On the Termination of the Chase Algorithm

Dissertation
zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat.)

vorgelegt dem Rat
der Technischen Fakultät
der Albert-Ludwigs-Universität Freiburg

von
Herrn Dipl.-Inf. bacc.math. Michael Meier
geboren am 14.11.1982
in Freiburg im Breisgau

Dekan:
Prof. Dr. Hans Zappe

Gutachter:
Prof. Dr. Georg Lausen
Prof. Dr. Bernhard Nebel

Datum der Einreichung: 16. Dezember 2009
Datum der Disputation: 21. Juni 2010

Danksagung

Ich danke den Dozenten des Graduiertenkollegs 806 für das zweijährige Promotionsstipendium. Außerdem danke ich meinem Betreuer Georg Lausen für die Freiheit mich das erforschen zu lassen, was ich wollte und für die Weiterfinanzierung durch eine Landesstelle nach Ablauf des Stipendiums.
Dem Mathematischen Institut der Universität Freiburg bin ich zutiefst zu Dank verpflichtet für die exzellente Ausbildung, die ich dort erhalten durfte. Insbesondere die Kenntnisse, die ich in den Lehrveranstaltungen von Jörg Flum erhalten habe, waren essentiell für meine Arbeit.
Ohne meine beiden Kollegen Michael Schmidt und Thomas Hornung hätte es diese Arbeit niemals gegeben. Die zahlreichen nichtwissenschaftlichen Diskussionen, die wir geführt und die Späße, welche wir zusammen gemacht haben, haben mir unendlich viel bedeutet. Ich hätte ohne die beiden niemals eine Dissertation in Informatik geschrieben. Besonders möchte ich Charlie dem Einhorn für seine Existenz und Nicht-Eindeutigkeit danken. Ich werde die Zeit mit den beiden sehr vermissen!
Ich möchte auch meinen anderen Kollegen Elisabeth Lott, Kerstin Pfeiffer, Fang Wei, Norbert Küchlin, Florian Schmedding, Kai Simon, Matthias Ihle, Martin Weber, Philipp Sorst und Liaquat Ali danken, auch wenn wir uns außer zu gelegentlichen gemeinsamen Essensgängen nicht so viel gesehen haben. Die Atmosphäre und der Umgang miteinander hier am Lehrstuhl ist sehr warm und herzlich.
Ich danke Assja Ignatova, Miryana Mircheva, Sascha Fröschl, Pablo Yanez Trujillo und Ingmar Berger für ihre Unterstützung in allen Lebenslagen. Außerdem gebührt meiner geliebten Assja ein weiterer Dank für die vielen Peperoni. Besbroi mnogo da be muci!
Zuguterletzt möchte ich meine geliebten Schwestern Ingelore und Marlene, meine Eltern und meine Tante Rita und ihren Mann Martin dankend erwähnen.

Zusammenfassung

Die vorliegende Arbeit behandelt das Problem der Terminierung des Chase-Algorithmus, einem wichtigen Hilfsmittel für viele Datenbank-Anwendungen wie z.B. die Optimierung konjunktiver Anfragen, Anfragebeantwortung mittels Sichten, Datenaustausch und Datenintegration. Die grundlegende Arbeitsweise des Chase-Algorithmus beruht darauf, gegeben eine Datenbankinstanz und eine Menge von Integritätsbedingungen, Verletzungen dieser Integritätsbedingungen auf der Instanz zu reparieren. Es ist seit langem bekannt, dass der Chase-Algorithmus nicht notwendigerweise für alle möglichen Eingaben terminiert. Im Allgemeinen ist dies sogar unentscheidbar. Diese Arbeit fasst bestehende Ansätze bezüglich hinreichender Terminierungsbedingungen für den Chase zusammen und entwickelt, darauf aufbauend, neue Techniken, die es uns erlauben schwächere hinreichende Terminierungsbedingungen aufzustellen. Insbesondere entwickeln wir, zum allerersten Mal überhaupt in der Literatur, Methoden, die es uns erlauben, die Terminierung mindestens einer Chase-Sequenz zu garantieren und nicht notwendigerweise von allen. Wir untersuchen, wie unsere Bedingungen zu den bestehenden Ansätzen in Beziehung stehen und geben obere Schranken für ihre algorithmische Komplexität an. Diese Analyse führt zu einem Algorithmus, der die Komplexität dieser Tests reduziert. Als weiterer Beitrag dieser Arbeit führen wir den Bereich der datenabhängigen Terminierungsbedingungen ein und präsentieren hinreichende Terminierungsbedingungen bezüglich festvorgegebener Datenbankinstanzen. Diese erlauben es uns, Aussagen über die Terminerung zu treffen, wenn die daten-unabhängigen Methoden keine Aussage erlauben. Als Anwendungen unserer Techniken transferieren wir unsere Methoden in den Bereich der semantischen Anfrageoptimierung und entwickeln eine Theorie zur regel-basierten Minimierung.

Sigillum:universitatis:studij:friburgensis:brisgaudie

On the Termination of the Chase Algorithm

Michael Meier

June 22, 2010

“Mathematics, rightly viewed, possesses not only truth, but supreme beauty - a beauty cold and austere, like that of sculpture.”

Bertrand Russell (1872 - 1970)

Abstract

We study the termination problem of the chase algorithm, a central tool in various database problems such as the constraint implication problem, conjunctive query optimization, rewriting queries using views, data exchange, and data integration. The basic idea of the chase is, given a database instance and a set of constraints as input, to fix constraint violations in the database instance. It is well-known that for an arbitrary set of constraints the chase does not necessarily terminate (in general, it is even undecidable if it does or not). Addressing this issue, we review the limitations of existing sufficient termination conditions for the chase and develop new techniques that allow us to establish weaker sufficient conditions. For the first time in the literature, we develop methods that allow us to ensure the termination of at least one chase sequence and not necessarily of all. We then study the interrelations of our termination conditions with previous conditions and the complexity of checking them. This analysis leads to an algorithm that reduces the complexity of checking our termination conditions. As another contribution, we study the problem of data-dependent chase termination and present sufficient termination conditions with respect to fixed instances. They might guarantee termination when our data-independent techniques cannot. As applications of our techniques beyond those already mentioned, we transfer our results into the field of semantic query optimization in the presence of types and develop the theory of rule-based minimization under constraints.

Contents

List of Figures

Chapter 1

Introduction

Riccardo: "What is this all about?"
Sergio: "I don't know."
Alice: "It's about a Swiss Army knife for database problems."

1.1 Motivation

The chase procedure is a fundamental algorithm that has been successfully applied in a variety of database applications [Maier et al., 1979; Johnson and Klug, 1982; Beeri and Vardi, 1984; Halevy, 2001; Deutsch et al., 2007; Lenzerini, 2002; Fagin et al., 2005; Fuxman et al., 2005; Deutsch et al., 2006; Olteanu et al., 2009]. Originally proposed to tackle the implication problem for data dependencies [Maier et al., 1979; Beeri and Vardi, 1984] and to optimize conjunctive queries (CQs) under data dependencies [Aho et al., 1979; Johnson and Klug, 1982], it has become a central tool in semantic query optimization (SQO) [Popa and Tannen, 1999; Deutsch et al., 2006; Schmidt et al., 2008]. For instance, the chase can be used to enumerate minimal CQs under a set of dependencies [Deutsch et al., 2006], thus supporting the search for more efficient query evaluation plans. Beyond SQO, it has been applied in many other contexts, such as data exchange [Fagin et al., 2005], peer data exchange [Fuxman et al., 2005], data integration [Lenzerini, 2002], query answering using views [Halevy, 2001; Deutsch et al., 2007], and probabilistic databases [Olteanu et al., 2009].

The core idea of the chase algorithm is simple: given a set of dependencies (also called constraints) over a database schema and a finite database instance as input, it fixes constraint violations in the instance. As a minimal and intuitive scenario we consider a database graph schema that provides a relation $\mathtt{E}(src, dst)$, which stores directed edges from node *src* to node *dst*, and a node relation $\mathtt{S}(s)$ containing nodes with some distinguished properties enforced by constraints. These constraints will vary from example to example and we will denote nodes in $\mathtt{S}$ as *special* nodes in the following. We sketch the idea of the chase algorithm using a single constraint

$$\alpha_1 := \forall x(\mathtt{S}(x) \rightarrow \exists y\ \mathtt{E}(x,y)),$$

stating that each special node has at least one outgoing edge. Now consider the sample database instance

$$I := \{\mathtt{S}(a),\, \mathtt{S}(b),\, \mathtt{E}(a,b)\}.$$

It is easy to see that I does not satisfy α_1 because it does not contain any outgoing edge for special node b. In its effort to fix the constraint violations in the database instance, the chase procedure would create the tuple

$$t_1 := \mathtt{E}(b, n_1),$$

where n_1 is a so-called fresh null value. The instance

$$I' := I \cup \{t_1\}$$

satisfies α_1, so the chase terminates and returns I' as result.
One major problem with the chase algorithm is that it does not terminate in the general case. To give the reader an idea of the problem, let us sketch a scenario that induces a non-terminating chase sequence. We replace the constraint α_1 by

$$\alpha_2 := \forall x(\mathtt{S}(x) \rightarrow \exists y\ \mathtt{E}(x,y),\, \mathtt{S}(y)),$$

which asserts that each special node links to another special node. Now consider the instance I from before. Obviously, I does not satisfy α_2 because special node b has no outgoing edge. In response, the chase fixes this constraints violation by adding the two tuples $\mathtt{E}(b, n_1)$ and $\mathtt{S}(n_1)$ to I, where n_1 is a fresh null value. Constraint α_2 is then fixed with respect to value b, but now the special node n_1 introduced in the last chase step violates α_2. In subsequent steps the chase would add

$$\mathtt{E}(n_1, n_2),\, \mathtt{S}(n_2),\, \mathtt{E}(n_2, n_3),\, \mathtt{S}(n_3),\, \ldots,$$

where n_2, n_3, ... are fresh null values. Hence, given I and α_2 as input, the chase will never terminate.
In case several constraints are violated, we arbitrarily choose one of them and repair it. Thus, the result of the chase depends on the order in which the constraints are applied. We call such an order a chase sequence. We point out that in general it is undecidable if the chase terminates for all or for at least one chase sequence, even for a fixed instance [Deutsch et al., 2008].

Addressing the problem of non-terminating chase sequences, several sufficient conditions for the input constraints have been proposed that guarantee termination on every database instance and every possible chase sequence [Fagin et al., 2005; Deutsch et al., 2008; Schmidt et al., 2008; Meier et al., 2009a]. The common idea

is to statically assert that there are no positions in the database schema where fresh null values are cyclically created in. The term *position* refers to a position in a relational predicate, e.g. `E`(*src*, *dst*) has two positions, namely *src*, denoted as $\mathtt{E}^1$, and *dst*, denoted as $\mathtt{E}^2$. The non-terminating chase sequence discussed before, for instance, cyclically creates fresh null values in both position $\mathtt{E}^1$ and $\mathtt{S}^1$.
One well-known termination condition for the chase is *weak acyclicity* [Fagin et al., 2005]. Roughly speaking, it implements a global study of the input constraints, to detect cyclically connected positions in the constraint set that introduce some fresh null values. In [Deutsch et al., 2008] it is claimed[1] that a newly developed condition called *stratification* improves weak acyclicity, showing that it suffices to assert weak acyclicity locally for subsets of constraints that might cyclically cause to fire each other. It is important to notice that the techniques introduced in [Fagin et al., 2005; Deutsch et al., 2008] take only the constraints into account and not the database instance. We call such termination conditions *data-independent*; their result is either the guarantee that the chase with these constraints terminates for *every* database instance or that no predictions can be made.

1.2 Contributions

A high-level description of our results with respect to chase termination is depicted in Figure 1.1. It shows that we divide the problem of chase termination into two branches. The first one studies data-independent methods. As the term data-independent suggests, these chase termination conditions work for every database instance. In contrast the second branch studies methods that depend on a given database instance, which is why they are called *data-dependent*. Both branches are divided again into two subbranches, namely sequence-independent and sequence-dependent methods. Sequence-independent chase termination conditions can guarantee chase termination no matter what chase sequence has been taken during the execution of the chase, in difference to that sequence-dependent conditions give termination guarantees only for (at least) one chase sequence. So far, the only branch that has been considered in the literature are the data- and sequence-independent methods. We want to point out that the current thesis is the first study of sequence-dependent methods and data-dependent termination conditions at all. In the following we summarize the key concepts and ideas of our analysis and survey the main results.

Data-independent chase termination. As discussed before, the source of non-terminating chase sequences are fresh null values that are cyclically created

---

[1] We will later show that this claim turns out to be wrong.

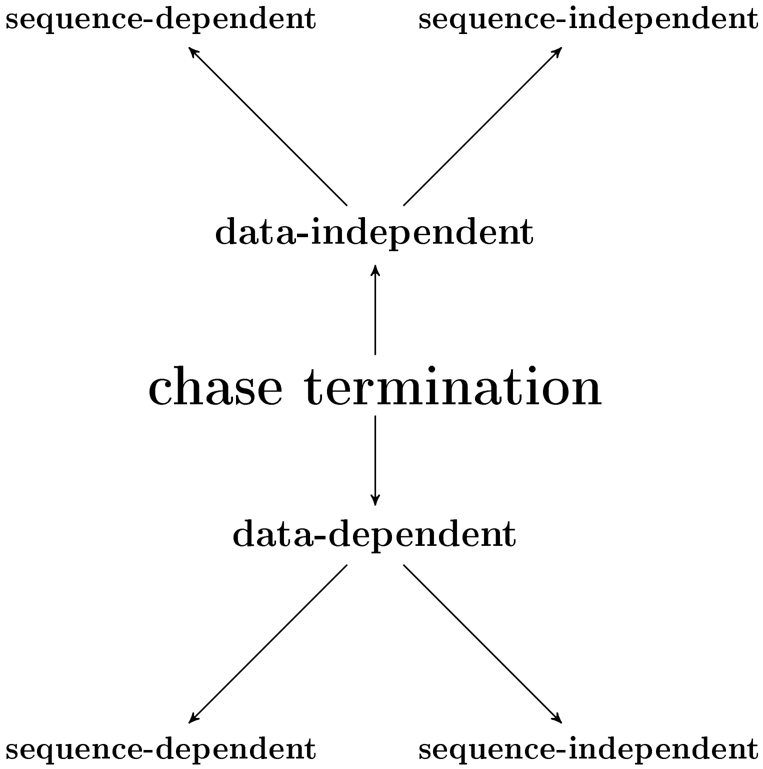


Figure 1.1: High-level summary of contributions.

at runtime in some position(s). We develop new techniques that allow us to statically approximate the set of positions where null values are created in or copied to during chase application, and use them to develop three hierarchies of sufficient termination conditions that are strictly more general than the weakest data-independent termination conditions known so far. The first hierarchy, the so-called ∀∀-T-hierarchy, takes sequence-independent termination into account, whereas the remaining ∀∃-T-hierarchy is sequence-dependent. Our results are based on the following ideas.

(1) Correction and exploration of the stratification condition: We show that stratification does not generally ensure termination of every chase sequence, as stated by the authors of [Deutsch et al., 2008], but of at least one chase sequence. Besides, we show that such a sequence can be statically determined independently of the input instance. This opens the door for the area of sufficient termination conditions for the chase that ensure, independently of the underlying data, the termination of at least one chase sequence and not necessarily of all. Furthermore, we propose

a possible correction of the stratification condition which ensures the termination for every chase sequence, as intended by the authors of [Deutsch et al., 2008], using the oblivious chase. This correction forms the basis for nearly all further results on sequence-independent chase termination.

(2) Identification of harmless null values: Often constraints introduce fresh null values in a certain position, but the (fixed) size of the database instance implies an upper bound on the number of null values that might be introduced in this position. To give an example, consider the constraint

$\alpha_3 := \forall x, y(\mathtt{S}(x),\, \mathtt{E}(x,y) \rightarrow \exists z\, \mathtt{E}(z,x)),$

which may create fresh null values in position E^1. Whenever α_3 is part of a constraint set that does not copy null values to or create null values in position $\mathtt{S}^1$, the number of fresh null values that might be introduced in position $\mathtt{E}^1$ by α_3 is implicitly fixed by the number of entries in $\mathtt{S}$.

(3) Analysis of the flow of null values: We statically approximate the set of positions to which null values might be copied during chase application, by a sophisticated study of the interrelations between the individual constraints. We illustrate the idea by a simple example. Consider the constraints

$\beta_1 := \forall x, y(\mathtt{S}(x),\, \mathtt{E}(x,y) \rightarrow \mathtt{E}(y,x))$ and
$\beta_2 := \forall x, y(\mathtt{S}(x),\, \mathtt{E}(x,y) \rightarrow \exists z\, \mathtt{E}(y,z),\, \mathtt{E}(z,x)),$

which assert that each special node with an outgoing edge has cycles of length 2 and 3 through all outgoing edges. We observe that none of these constraints inserts fresh null values into relation $\mathtt{S}$, so the chase will terminate as soon as β_1 and β_2 have been fixed for all special nodes with an outgoing edge, i.e. after a finite number of steps. Somewhat surprisingly, none of the existing conditions recognizes chase termination for the above scenario. The reason is that they do not analyse the flow of null values. Our approach exhibits such an analysis and guarantees chase termination for the two constraints above.

(4) Inductive decomposition of the constraint set: The constraint set in the previous example is not dangerous because no fresh null values are created in position $\mathtt{S}^1$. Let us, in addition to β_1 and β_2, consider the constraint

$\beta_3 := \exists x, y\, \mathtt{S}(x),\, \mathtt{E}(x,y),$

stating that there is at least one special node with an outgoing edge. Clearly, β_3 fires at most once, so the chase for the constraint set $\{\beta_1, \beta_2, \beta_3\}$ will still terminate. However, β_3 complicates the analysis because it "infects" position $\mathtt{S}^1$ in the

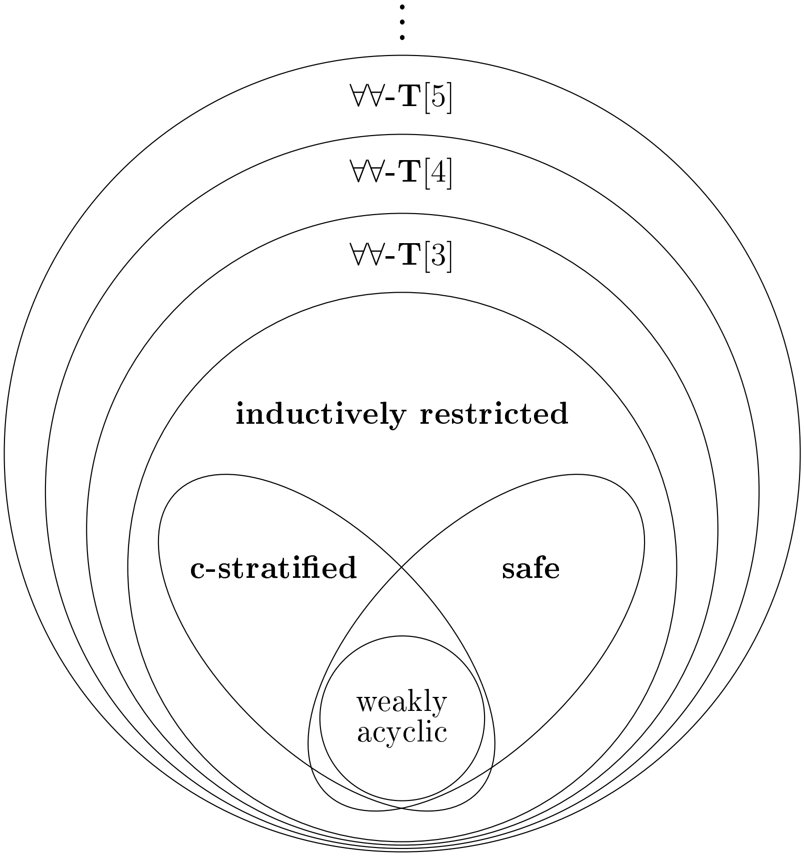


Figure 1.2: Data- and sequence-independent chase termination conditions.

sense that now null values may be created in this position. We resolve such situations by an (inductive) decomposition of the constraint set. When applied to the above example, our approach would recognize that β_3 is not cyclically connected with β_1 and β_2, and decompose the constraint set into the two subsets $\{\beta_1, \beta_2\}$ and $\{\beta_3\}$, which then are recursively inspected.

Based upon these ideas we develop two novel sufficient chase termination conditions, called *safety* and *inductive restriction*. Figure 1.2 surveys our main results in the field of sequence-independent termination conditions and relates them to the previous termination condition weak acyclicity and to c-stratification, the corrected version of stratification. All classes in the figure guarantee chase termination in polynomial-time data complexity and all inclusion relationships are strict. As it can be seen, safety generalizes weak acyclicity and is further generalized by inductive restriction. On top of inductively restricted constraints we ultimately define a hierarchy of sufficient termination conditions called ∀∀-T-hierarchy. To give an intuition for a fixed class in this hierarchy, say ∀∀-T[k], we study the flow and creation of fresh null values in detail for chains of up to k constraints that might cause to fire each other in sequence.

Based on the ∀∀-T-hierarchy and stratification, we develop a hierarchy of sequence-dependent termination conditions for the chase. The ∀∃-T-hierarchy combines sequence-dependent decomposition of the constraint with the ideas of the ∀∀-T-hierarchy. In particular, every level of the ∀∃-T-hierarchy is strictly contained in the same level of the ∀∀-T-hierarchy.

An algorithm. It can be checked in polynomial time if a constraint set is safe; in contrast, the recognition problem for inductively restricted constraints and all classes in the ∀∀-T-hierarchy is in coNP. We develop an efficient algorithm that accounts for the increasing complexity of the recognition problem. The algorithm can be used to test membership of a constraint set in a fixed level of the ∀∀-T-hierarchy. The basic idea is to combine the different sufficient termination conditions in order to reduce the complexity of the termination check where possible. We discuss that the algorithm can be easily adapted so that it yields an efficient membership test for the ∀∃-T-hierarchy.

Data-dependent chase termination. Whenever the input constraint set does not fall into some fixed level of the ∀∀-T-hierarchy, no termination guarantees for the general case can be derived. Arguably, reasonable applications should never risk non-termination, so the chase cannot be safely applied to *an arbitrary instance* in this case. Tackling this problem, we study *data-dependent chase termination:* given constraint set Σ and a fixed instance I, does the chase with Σ terminate on I for every chase sequence or at least for one? This setting makes particularly sense in the context of SQO, where the query – interpreted as database instance – is chased: typically, the size of the query is small, so the "data" part can be efficiently analyzed as opposed to the case where the input is a large database instance. We propose two complementary approaches.

(1) Static analysis: Our first, static, scheme relies on the observation that if the instance is fixed, we can ignore constraints in the constraint set which will never fire when chasing the instance, i.e. if general sufficient termination guarantees hold for those constraints that might fire. As a fundamental result, we show that in general, it is undecidable if a constraint will never fire on a fixed instance. Still, we give a *sufficient* condition that allows us to identify such constraints in many cases. We derive, based on the previous data-independent methods, sufficient data-dependent conditions for both the sequence-dependent and -independent side.

(2) Monitoring: Whenever the static approach fails, our second, dynamic, approach comes into play: we run the chase and track cyclically created fresh null values in a so-called monitor graph. We then fix the maximum depth of cycles in the monitor graph and stop the chase when this limit is exceeded: in such a case, no termination guarantees can be made. However, we show that each fixed search depth implicitly defines a class of constraint-instance pairs for which the chase terminates. Intuitively, the search depth limit can be seen as a natural condition that allows us to stop the chase when "dangerous" situations arise. Under these considerations, our approach adheres to situations that are likely to cause non-termination, so it is preferable to blindly running the chase and aborting after a fixed amount of time, or a fixed number of chase steps. Applications may fix the search depth following a pay-as-you-go principle.

Ultimately, the combination of our static and dynamic analysis often allows us to safely apply the chase procedure although no data-independent termination guarantees can be made.

Applications. Finally, we apply our methods in different contexts. First, we consider semantic query optimization in the presence of types and type hierarchies, then we look at an application in the Semantic Web. Both of these areas have in common that they use the chase as a central tool. Therefore, our results on chase termination directly apply to them.

(1) Semantic query optimization in the presence of types. Both semantic and type-based query optimization rely on the idea that queries often exhibit non-trivial rewritings if the state space of the database is restricted. Despite their close connection, these two problems to date have always been studied separately. We present a unifying, logic-based framework for query optimization in the presence of data dependencies and type information. It builds upon the classical chase algorithm and extends existing query minimization techniques to considerably larger classes of queries and dependencies. As a technical challenge, our setting involves chasing of (possibly unions of) conjunctive queries in the presence of constraints containing negation and disjunction.

In the following, we sketch the major contributions of our work in more detail.

- We develop a framework that allows to integrate type-based optimization into the semantic optimization process. In this framework, data dependencies (modeled as first-order sentences) coexist with a so-called type system, which is represented as

  - a set $\mathcal{T}$ of unary predicate symbols, one for each type,
  - a type interpretation for constants appearing in the query and the constraints, and
  - a set of full constraints (i.e., constraints without existential quantification) modeling interrelations between the types, such as inclusion or disjointness relationships.

  We are not aware of database-related type interrelations that cannot be encoded using this framework.

- On top of our framework, we develop a variant of the C&B algorithm from [Deutsch et al., 2006] that can be used to optimize and minimize unions of conjunctive queries with negation with respect to a constraint base, a type system, and a generic cost function. Whether or not this framework gives us the power to compute minimal rewritings of an input query lastly depends on the termination of the underlying chase, just like in the context of standard TGDs and EGDs. Although stated for our specific optimization framework, these results can be understood as consequent enhancements of previous results on containment testing and minimization under data dependencies using the chase.

- In response to the central role of chase termination in our setting, we develop novel chase termination conditions for constraint sets involving disjunction and negation. Rather than developing these termination conditions from scratch, our approach is to carry over existing sufficient termination conditions for standard TGDs and EGDs, i.e. we show how to make existing conditions applicable in the context of constraint sets involving disjunction and negation.

- We study the complexity of decision problems related to our type-based semantic optimization scheme. In particular, our results confirm that – whenever we can guarantee chase termination – important problems like testing query equivalence or minimality under constraints and types with respect to a generic cost function fall into low levels of the polynomial hierarchy. We therefore shall expect that our techniques are feasible in practice if the size of the queries is small.

(2) Rule-based minimization of RDF graphs under constraints: The Semantic Web [Berners-Lee et al., 2001] facilitates semantic interoperability and exchange of data between applications. The Resource Description Framework [World Wide Web Consortium, 2003a] was proposed by the World Wide Web consortium as a standard language for data in the Semantic Web. Conceptionally, RDF graphs are

collections of so-called triples of knowledge, where each knowledge triple in the database has the form $(subject, predicate, object)$ and models the binary relation predicate between the subject and the object. While structurally homogeneous, the triple format offers great flexibility and allows to establish statements about and connections between resources. As RDF has only very simple language constructs, RDF data often becomes large. There has been a line of research [World Wide Web Consortium, 2003c; Gutierrez et al., 2004, 2003; Iannone et al., 2005; Esposito et al., 2005] to minimize RDF graphs without losing any information, i.e. retaining homomorphic equivalence. This allows applications to exchange reduced data, thus minimizing storage cost, data-shipping and query evaluation time. In [World Wide Web Consortium, 2003c; Gutierrez et al., 2004, 2003] the notion of lean graphs was introduced as a minimal representation of an RDF graph. Basically, a lean graph eliminates triples which contain blank nodes that specify redundant information. For example, in the graph

$$\{(a_1, a_2, a_3), (X, a_2, Y)\}$$

the triple (X, a_2, Y) can be eliminated (X, Y are blank nodes) because both X and Y are treated like existentially quantified variables in the RDF semantics [World Wide Web Consortium, 2003c] and the triple (a_1, a_2, a_3) witnesses the existence of such a resource (X, a_2, Y). In [Iannone et al., 2005; Esposito et al., 2005] different algorithms are introduced that approximately compute a lean version of a given RDF graph. The notion of a lean graph is orthogonal to derivability by application-specific rules. If such rules exist, a lean graph may still contain triples that are redundant in the sense that they need not be explicitly stored because they could be derived by the rules as well. We propose a user-specific redundancy elimination technique based on rules. Usually, rules are interpreted generatively, i.e. if we have a rule of the form

$$\mathtt{P}(X, Y) \leftarrow \mathtt{R}(Y, X)$$

and we find $\mathtt{R}(a, b)$ in our data, we add $\mathtt{P}(b, a)$ to it. In our work, we use rules in the following sense: whenever $\mathtt{P}(b, a)$ and $\mathtt{R}(a, b)$ are in our data, we delete $\mathtt{P}(b, a)$. If later needed, we can recompute the tuple $\mathtt{P}(b, a)$ with the help of our rule, i.e. we minimize a given RDF graph such that all deleted triples can be reconstructed. It may not be surprising that we will use the chase as an important tool during the process of redundancy elimination.

1.3 How (Not) to Read this Thesis

Although we provide all relevant basics to the chase algorithm, we would not recommend this document for a beginner in this field. Furthermore, it is not

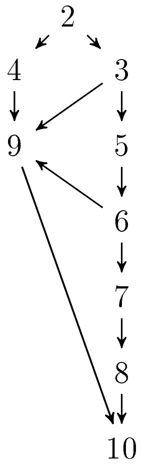


Figure 1.3: A reading graph.

suited for selective or cross-reading. This is due to the simple fact that some chapters depend on the techniques introduced in previous ones. A reading graph for this document is depicted in Figure 1.3. In order to further facilitate reading, we have included all proofs in the main text and not in an appendix. We expect the reader to be familiar with mathematical thinking, common notation, and proof techniques.

The thesis is structured as follows. Chapter 2 introduces the usual background of mathematical logic including some specializations to database theory. The chase algorithm, main subject of this thesis, is informally and formally presented in Chapter 3 together with its logical properties and previous results on termination. Afterward, we explore some of the chase's main applications like query optimization, data exchange and query answering using views. Chapter 5 explains four different aspects of chase termination, three of them not considered in the literature so far. Next, in Chapter 6, we come to the core part of this thesis, namely data- and sequence-independent chase termination, followed by data-independent and sequence-dependent termination in Chapter 7. In Chapter 8, we introduce data-dependent termination techniques. The subsequent Chapter 9 is devoted to our application scenarios. The last two Chapters 10 and 11 contain related work, a summary of the thesis and possible directions for future work.

1.4 Summary of Publications

Parts of the work that is discussed in the course of this thesis have been published at major database venues and as technical reports. In the following, we summarize publications that are related to this thesis and sketch their relations to this work.

Most of the author's publications are related to the chase algorithm. The first termination conditions were published in [Schmidt et al., 2008], followed by [Meier et al., 2009b] and its technical report [Meier et al., 2009a]. The main results of Chapters 6 and 8 come from the work in [Meier et al., 2009d] and its corresponding technical report [Meier et al., 2009c]. Our contributions to the field of semantic query optimization in the presence of types and sequence-dependent chase termination are from [Meier et al., 2010].

Another line of publications is related to the Semantic Web. In [Lausen et al., 2008] we demonstrated the use of integrity constraints for SPARQL processing in RDF databases. In the follow-up paper [Meier, 2008] we presented our idea of a rule-based minimization method under constraints from Section 9.2.
Our works in [Schmidt et al., 2008, 2010] related to the complexity of the SPARQL query language and possible optimization approaches are not covered in this thesis.

Chapter 2

Preliminaries

> **Riccardo:** "I'm getting out of patience!"
> **Alice:** "Calm down, we first need the very basics."

In this chapter we introduce the necessary background from mathematical logic and database theory, that forms the basis for this thesis. We mostly follow the standard definitions from these areas.

2.1 Mathematical Logic

We will introduce the usual background from mathematical logic as in [Ebbinghaus et al., 1996] making some specializations to database theory. We closely follow the presentation in [Ebbinghaus et al., 1996] and [Flum, 2002] and present the usual definitions such as some basic set-theoretic notations, signatures, formulas, structures and satisfaction of formulas.

2.1.1 Set-theoretic Basics

The natural numbers $\mathbb{N}$ do not include 0; $\mathbb{N}_0$ is used as a shortcut for $\mathbb{N} \cup \{0\}$. As usual in algebra, $\mathbb{N}[X]$ is the set of polynomials with coefficients in $\mathbb{N}$. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, ..., n\}$ and $[0] := \emptyset$.
Further, for a set M, we denote by 2^M its powerset and by $|M|$ its cardinality. For two sets M, N we write $M \subseteq N$ is M is a subset of N. We write $M \subset N$ if $M \subseteq N$ and $M \neq N$.
When we have two partial functions $f, f' : M \hookrightarrow N$, we write $f \subseteq f'$ iff[1] $\{(x, f(x)) | x \in M, f(x) \text{ is defined}\} \subseteq \{(x, f'(x)) | x \in M, f'(x) \text{ is defined}\}$.
In some proofs we will require basic knowledge on quotient sets, which are well-known in model theory. An equivalence relation is a binary relation that is reflexive, symmetrical, and transitive. The idea is that we take a set A and an

---

[1] We use "iff" as an abbreviation for "if and only if".

equivalence relation $B \subseteq A \times A$ and treat $a, b \in A$ as equal if $(a, b) \in B$. The resulting set A/B is called a quotient set. As a technical definition we set $[a]$ to be $\{b \in A | (a, b) \in B\}$. Then, we define A/B to be the set $\{[a] | a \in A\}$.
Abusing notation, we denote by $|s|$ also the length of a logical formula. Given a tuple $t = (t_1, \ldots, t_n)$ we define the tuple obtained by projecting on positions $1 \leq i_1 < \cdots < i_m \leq n$ as $p_{i_1,\ldots,i_m}(t) := (t_{i_1}, \ldots, t_{i_m})$.
We choose three pairwise disjoint infinite sets Δ, Δ_{null} and V. We will refer to Δ as the set of *constants*, to Δ_{null} as the set of *labeled nulls* and to V as the set of variables. Often we will denote a sequence of variables, constants or labeled nulls by $\overline{a}$ if the length of this sequence is understood from the context.

2.1.2 Signatures and Formulas

A *relational signature* $\mathcal{R}$ is a finite set of relational symbols $\{\mathtt{R}_1, ..., \mathtt{R}_n\}$. To every $\mathtt{R}_i \in \mathcal{R}$ we assign a natural number $\mathrm{ar}(\mathtt{R}_i) \in \mathbb{N}$, which we call the *arity* of $\mathtt{R}_i$. The arity of $\mathcal{R}$, denoted by $\mathrm{ar}(\mathcal{R})$, is defined as $\max\{ar(\mathtt{R}_i) | i \in [n]\}$. Throughout the rest of the thesis, we assume the database schema, the set of constants, and the set of labeled nulls to be fixed. This is why we will suppress these sets in our notation.

An $\mathcal{R}$-formula is an expression which is obtained by a finite number of applications of the following rules:

1. If $t_1, t_2 \in \Delta \cup V$, then $t_1 = t_2$ is an $\mathcal{R}$-formula.
2. If $\mathtt{R} \in \mathcal{R}$ and $t_1, ..., t_{ar(\mathtt{R})} \in \Delta \cup V$, then $\mathtt{R}(t_1, ..., t_{ar(\mathtt{R})})$ is an $\mathcal{R}$-formula.
3. If φ is an $\mathcal{R}$-formula, then $\neg\varphi$ is an $\mathcal{R}$-formula.
4. If φ, ψ are $\mathcal{R}$-formulas, then $(\varphi \vee \psi)$ and $(\varphi \wedge \psi)$ are $\mathcal{R}$-formulas.
5. If φ is an $\mathcal{R}$-formula and $x \in V$, then $\exists x\varphi$ and $\forall x\varphi$ are $\mathcal{R}$-formulas.

We also define abbreviations and write $(\varphi \rightarrow \psi)$ instead of the expression $(\neg\varphi \vee \psi)$ and $\varphi_1, ..., \varphi_n$ instead of $\bigwedge_{i \in [n]} \varphi_i$.
We assume that the reader is familiar with the concept of unique decomposition of formulas into its subexpressions. Therefore, we define the notion of free variables in an $\mathcal{R}$-formula by induction on its structure. For any $\mathcal{R}$-formula φ, the set of free variables , $\textsc{free}(\varphi)$, is inductively defined as follows:

1. $\textsc{free}(t_1 = t_2) := \{t_1, t_2\} \cap V$,
2. $\textsc{free}(\mathtt{R}(t_1, ..., t_{ar(R)})) := \{t_1, ..., t_{ar(\mathtt{R})}\} \cap V$,

3. $\textsc{free}(\neg\psi) := \textsc{free}(\psi)$,
4. $\textsc{free}((\psi_1 \vee \psi_2)) := \textsc{free}(\psi_1) \cup \textsc{free}(\psi_2)$,
5. $\textsc{free}((\psi_1 \wedge \psi_2)) := \textsc{free}(\psi_1) \cup \textsc{free}(\psi_2)$,
6. $\textsc{free}(\forall x\psi) := \textsc{free}(\psi)\backslash\{x\}$ and
7. $\textsc{free}(\exists x\psi) := \textsc{free}(\psi)\backslash\{x\}$.

If for some $\mathcal{R}$-formula φ, we have $\textsc{free}(\varphi) = \emptyset$, then we say that φ is a sentence. We denote the set of $\mathcal{R}$-formulas that are sentences by L_0.

2.1.3 Structures and Satisfaction

An $\mathcal{R}$-structure is a pair $I = (A, a)$, where A is a non-empty set and a is a function that is defined on $\mathcal{R}$ and assigns to every $\mathtt{R} \in \mathcal{R}$ a subset of $A^{ar(\mathtt{R})}$, i.e. $a(\mathtt{R}) \subseteq A^{ar(\mathtt{R})}$. Instead of $a(\mathtt{R})$, we will often write $\mathtt{R}^I$ and $(A,\mathtt{R}_1^I, ...,\mathtt{R}_n^I)$ instead of $I = (A, a)$. An assignment of variables in an $\mathcal{R}$-structure I is a mapping $\gamma : V \to A$. If γ is an assignment of variables in I, then $\gamma\frac{a}{x}$ is the assignment that maps x to a and otherwise coincides with γ. An $\mathcal{R}$-interpretation $\mathcal{I}$ is a pair (I, γ), where I is an $\mathcal{R}$-structure and γ an interpretation in I. We write $\mathcal{I}\frac{a}{x}$ for the interpretation $(I, \gamma\frac{a}{x})$.

We now turn toward the definition of satisfaction, but first we define the value of an $\mathcal{R}$-interpretation on variables and constants. Let $\mathcal{I} = (I, \gamma)$ be an $\mathcal{R}$-interpretation. For every $x \in V$ we set $\mathcal{I}(x) := \gamma(x)$ and for every $c \in \Delta$, we define $\mathcal{I}(c) := c$.

We define the notion of satisfaction of $\mathcal{R}$-formulas by $\mathcal{R}$-interpretations by induction on the structure of an $\mathcal{R}$-formula as follows:

1. $\mathcal{I} \models t_1 = t_2 \iff \mathcal{I}(t_1) = \mathcal{I}(t_2)$,
2. $\mathcal{I} \models \mathtt{R}(t_1, ..., t_n) \iff \mathtt{R}^I(\mathcal{I}(t_1), ..., \mathcal{I}(t_n))$,
3. $\mathcal{I} \models \neg\varphi \iff$ not $\mathcal{I} \models \varphi$,
4. $\mathcal{I} \models (\varphi \vee \psi) \iff \mathcal{I} \models \varphi$ or $\mathcal{I} \models \psi$,
5. $\mathcal{I} \models (\varphi \wedge \psi) \iff \mathcal{I} \models \varphi$ and $\mathcal{I} \models \psi$,
6. $\mathcal{I} \models \exists x\varphi \iff$ there is some $a \in \Delta \cup \Delta_{null}$ such that $\mathcal{I}\frac{a}{x} \models \varphi$ and
7. $\mathcal{I} \models \forall x\varphi \iff$ for all $a \in \Delta \cup \Delta_{null}$ it holds that $\mathcal{I}\frac{a}{x} \models \varphi$.

From the famous coincidence lemma in classical logic [Ebbinghaus et al., 1996] we can conclude that if φ is a sentence, then for all $\mathcal{R}$-structures I and assignments of variables γ_1, γ_2 in I it holds that

$$(I, \gamma_1) \models \varphi \Longleftrightarrow (I, \gamma_2) \models \varphi.$$

Therefore, we introduce the abbreviation $I \models \varphi$ which we use instead of $(I, \gamma) \models \varphi$ for any assignment of variables γ. For $\Phi \subseteq L_0$, we write $I \models \Phi$ iff $I \models \varphi$ for every $\varphi \in \Phi$. Given $\psi \in L_0$, we write $\Phi \models \psi$ iff for every $\mathcal{R}$-structure I such that $I \models \Phi$ it holds that $I \models \psi$.

2.1.4 Complexity Theory

As usual, we denote by PTIME (or P, for short) the complexity class comprising all problems that can be decided by a deterministic Turing machine (TM) in polynomial time, and by NP the set of problems that can be decided by a non-deterministic TM in polynomial time.

Given a complexity class C we denote by coC the set of decision problems whose complement can be decided by a TM in class C. Given complexity classes C_1 and C_2, the class $\mathrm{C}_1^{\mathrm{C}_2}$ captures all problems that can be decided by a TM M_1 in class C_1 enhanced by an oracle TM M_2 for solving problems in class C_2. Informally, engine M_1 can use M_2 to obtain a *yes*/*no*-answer for a problem in C_2 in a single step. We refer the interested reader to [Arora and Barak, 2009] for a more formal discussion of oracle machines. Finally, we define the classes Σ_i^P and Π_i^P inductively as

$$\Sigma_0^P = \Pi_0^P := \mathrm{P} \text{ and } \Sigma_{n+1}^P := \mathrm{NP}^{\Sigma_n^P}, \text{ and put } \Pi_{n+1}^P := \mathrm{coNP}^{\Sigma_n^P}.$$

The polynomial hierarchy PH [Stockmeyer, 1976] is then defined as

$$\mathrm{PH} = \bigcup_{i \in \mathbb{N}_0} \Sigma_i^P$$

It is folklore that $\Sigma_i^P = co\Pi_i^P$, and that $\Sigma_i^P \subseteq \Pi_{i+1}^P$ and $\Pi_i^P \subseteq \Sigma_{i+1}^P$ holds. Moreover, the following inclusion hierarchies for Σ_i^P and Π_i^P are known.

$$\mathrm{P} = \Sigma_0^P \subseteq \mathrm{NP} = \Sigma_1^P \subseteq \Sigma_2^P \subseteq \cdots \subseteq \mathrm{PSPACE}, \text{ and}$$
$$\mathrm{P} = \Pi_0^P \subseteq \mathrm{coNP} = \Pi_1^P \subseteq \Pi_2^P \subseteq \cdots \subseteq \mathrm{PSPACE}.$$

We consider hardness and completeness only with respect to polynomial-time many-one reductions. 3SAT is the problem to determine, given a formula from propositional logic in 3CNF as input, whether it is satisfiable. It is well-known that 3SAT is NP-complete [Arora and Barak, 2009]. The containment problem for conjunctive queries with inequality predicates was proved to be Π_2^P-complete in [Kolaitis et al., 1998].

2.2 Relational Databases

We will introduce the usual background from database theory while following the standard definition from this area. We present the definition of relational database instances, classes of relational constraints, an alternative characterization for their satisfaction and provide a short motivation for them.

2.2.1 Relational Database Instances

A *database schema* $\mathcal{R}$ is a finite set of relational symbols. A database position is a pair $(\mathtt{R}, i)$ where $\mathtt{R} \in \mathcal{R}$ and $i \in [ar(\mathtt{R})]$, for short we write $\mathtt{R}^i$, e.g. a three-ary predicate $\mathtt{S}$ has three positions $\mathtt{S}^1$, $\mathtt{S}^2$, $\mathtt{S}^3$.
An $\mathcal{R}$-atom is an expression of the form $\mathtt{R}(a_1, ..., a_n)$, where $a_1, ..., a_n \in \Delta \cup \Delta_{null} \cup V$. We say that a variable, labeled null, or constant c appears e.g. in a position $\mathtt{R}^1$ if there exists an $\mathcal{R}$-atom $\mathtt{R}(c, ...)$.

In the rest of the thesis, we assume the database schema, the set of constants and the set of labeled nulls to be fixed and therefore, we will omit them in our notations.

A *database instance* I is a set of equality-free $\mathcal{R}$-atoms that contains only elements from $\Delta \cup \Delta_{null}$ in its positions, i.e. it contains only $\mathcal{R}$-atoms of the form $\mathtt{R}(a_1, ..., a_n)$, where $a_1, ..., a_n \in \Delta \cup \Delta_{null}$. Please note that, by definition, a database may be infinite.
The domain of I, $dom(I)$, is the set of elements from $\Delta \cup \Delta_{null}$ that appear in I, i.e. $dom(I) = \{a_1, ..., a_n | \mathtt{R}(a_1, ..., a_n) \in I\} \subseteq \Delta \cup \Delta_{null}$.

For every sentence φ, we write $I \models \varphi$ if and only if $\mathcal{I} \models \varphi$, where the $\mathcal{R}$-structure $\mathcal{I}$ is defined as follows. For every $\mathtt{R} \in \mathcal{R}$ we set $\mathtt{R}^I := \{(a_1, ..., a_n) | \mathtt{R}(a_1, ..., a_n) \in I\}$. Then, $\mathcal{I} := (dom(I), (\mathtt{R}^I)_{\mathtt{R} \in \mathcal{R}})$.

2.2.2 Examples and Motivation

We give a small example. Consider a database schema {`flight`, `hasAirport`}, where `flight` has arity two and `hasAirport` has arity one. Intuitively, these predicates are supposed to contain information about flights between cities and if a city has an airport or not. Let the instance

{`flight`(Nuremberg, Amsterdam), `flight`(Amsterdam, New York),
`hasAirport`(New York)}

be given. It states the existence of an airport in New York and of flights from Nuremberg to Amsterdam and from Amsterdam to New York. Intuitively, this database is not really "complete" because there are flights scheduled for cities for which it is not known that they have an airport. So, if we know that there is an incoming or outgoing flight scheduled for a city, we should be able to deduce that this city must have an airport.

So, the definition of a database alone does not satisfy all of our needs. It is desirable to allow only *meaningful data*, reflecting the part of the world we want to model, to be inserted into the database. But what is meaningful data? This strongly depends on the application domain and is usually only known to the database schema designer or the users of the database. These people should have a way to specify what meaningful data is, i.e. to restrict the kind of data so that it reflects the part of the real world to be modeled. This is where relational constraints come into play.

2.2.3 Relational Constraints

Integrity constraints in databases have been intensively studied [Abiteboul et al., 1995; Deutsch et al., 2007; Deutsch and Nash, 2008a; Deutsch et al., 2006; Aho et al., 1979; Beeri and Vardi, 1984; Calì et al., 2008, 2009; Cheng et al., 1999; Cosmadakis and Kanellakis, 1986; Deutsch et al., 2008; Fagin et al., 2005; Fuxman et al., 2005; Gottlob and Nash, 2008; Johnson and Klug, 1982; Maier et al., 1979; Marnette, 2009]. Relational constraints allow the database schema designer to restrict the contents of a database to meaningful data. Integrity constraints ensure that the contents of the database obey certain real-life properties. E.g. a constraint of the form

$$\forall x, y(\texttt{flight}(x, y) \rightarrow \texttt{hasAirport}(x), \texttt{hasAirport}(y))$$

ensures that whenever there is a flight connection stored in the database, then the source and the destination of the flight both have an airport.

This section introduces the classes of relational constraints considered in this thesis, namely equality- and tuple-generating dependencies. These are able to express virtually all database constraints from the literature [Abiteboul et al., 1995; Deutsch and Nash, 2008a], e.g. functional dependencies, key dependencies, join dependencies, multi-valued dependencies, inclusion dependencies and foreign key dependencies. For a more thorough introduction to relational constraint types and their first-order representation, we recommend reading article [Deutsch and Nash, 2008a].

Whenever we use the term constraint in this thesis, we mean a mix of equality- and tuple-generating dependencies.

Equality-Generating Dependencies

Let $\overline{x}, \overline{y}$ be tuples of variables. An equality-generating dependency (EGD) is a first-order sentence

$$\varphi := \forall\overline{x}(\phi(\overline{x}) \to x_i = x_j) \in L_0,$$

where

- x_i, x_j either occur as universally quantified variables in ϕ or are elements from Δ and
- ϕ is a non-empty conjunction of equality-free $\mathcal{R}$-atoms (possibly with constants from Δ).

We denote by *body*(φ) the set of atoms in ϕ and by *head*(φ) the set $\{x_i = x_j\}$.

For brevity, we will often omit the $\forall$-quantifier and the respective list of universally quantified variables.

Tuple-Generating Dependencies

Let $\overline{x}, \overline{y}$ be tuples of variables. A tuple-generating dependency (TGD) is a first-order sentence

$$\varphi := \forall\overline{x}(\phi(\overline{x}) \to \exists\overline{y}\psi(\overline{x}, \overline{y})) \in L_0$$

such that

- both ϕ and ψ are conjunctions of atomic $\mathcal{R}$-formulas (possibly with constants from Δ),
- ψ is not empty,
- ϕ is possibly empty,
- both ϕ and ψ do not contain equality atoms, and
- all variables from $\overline{x}$ that occur in ψ must also occur in ϕ.

We denote by *body*(φ) the set of atoms in ϕ and by *head*(φ) the set of atoms in ψ.

2.2.4 Satisfaction and Homomorphisms

This section introduces a different characterization of constraint satisfaction in terms of homomorphisms, which is important to be understood because the chase algorithm is based on homomorphisms.

A homomorphism from a set of $\mathcal{R}$-atoms A_1 to a set of $\mathcal{R}$-atoms A_2 is a mapping

$$\mu : \Delta \cup \Delta_{null} \cup V \to \Delta \cup \Delta_{null},$$

such that the following conditions hold:

1. if $c \in \Delta$, then $\mu(c) = c$ and
2. if $\mathtt{R}(c_1, ..., c_n) \in A_1$, then $\mathtt{R}(\mu(c_1), ..., \mu(c_n)) \in A_2$.

We can use homomorphisms to characterize the satisfaction of EGDs and TGDs as follows.

Proposition 1.

- For every TGD φ and every database instance I it holds that $I \models \varphi \iff$ for every homomorphism h from $body(\varphi)$ to I it holds that there is a homomorphism $h' \supseteq h$ that maps $head(\varphi)$ to I.
- For every EGD φ with $head(\varphi) = \{x_i = x_j\}$ and every database instance I it holds that $I \models \varphi \iff$ for every homomorphism h from $body(\varphi)$ to I it holds that $h(x_i) = h(x_j)$.

Chapter 3

The Chase Algorithm

> **Riccardo:** "What are we going to do with it?"
> **Alice:** "Define our algorithm."

This chapter introduces the main tool of interest in this thesis: the chase algorithm. After an informal and a formal discussion, we will briefly review its most important logical properties before inspecting some classical areas of application. Although this chapter is self-contained, we recommend article [Deutsch and Nash, 2008b] for further reading and more references related to the chase and its applications.

3.1 Informal Description

The chase algorithm takes as input a set of constraints Σ and a finite database instance I. Let e.g. the following scenario be given: the database instance

$$I := \{\texttt{fly}(\text{Amsterdam,New York})\}$$

and the set of TGDs $\Sigma := \{\alpha, \beta\}$, where

$$\alpha := \texttt{fly}(x_1, x_2) \rightarrow \exists y\ \texttt{fly}(x_2, y) \text{ and}$$
$$\beta := \texttt{fly}(x_1, x_2) \rightarrow \texttt{fly}(x_2, x_1).$$

Constraint α states that if there is an incoming flight, then there must also be an outgoing flight scheduled, whereas constraint β ensures that flight connections are symmetrical. Obviously, it holds that if a database instance satisfies β, then it must also satisfy α, but for the sake of the discussion of the chase algorithm we will consider both constraints and observe the chase's different effects for these constraints.

Intuitively, the chase repairs violations of Σ in I. Consider the violation of β. We have that $I \models \texttt{fly}(\text{Amsterdam,New York})$, but $I \not\models \texttt{fly}(\text{New York,Amsterdam})$, which is why β is not satisfied in this setting. So, in this case the chase adds the tuple $\texttt{fly}(\text{New York,Amsterdam})$ to I resulting in the instance

$I' = \{$`fly`(Amsterdam,New York), `fly`(New York,Amsterdam)$\}$.

We note that $I' \models$ `fly`(New York,Amsterdam). Again, we look for constraint violations in I', but we will not find any because $I' \models \Sigma$. Here, the chase terminates and returns I' as result.
But there is no need for the chase to choose a violation for β to repair. It could also have chosen a violation of α. E.g. we have that $I \models$ `fly`(Amsterdam,New York), but $I \not\models \exists y$ `fly`(New York,y), which means that there is no outgoing flight scheduled for New York. In this case the chase adds a tuple of the form `fly`(New York,n_1) to I, where $n_1 \in \Delta_{null} \backslash dom(I)$, resulting in the instance

$I'' = \{$`fly`(Amsterdam,New York), `fly`(New York,n_1)$\}$.

We note that $I'' \models \exists y$ `fly`(New York,y), so the constraint is locally satisfied. We see that $I'' \not\models \Sigma$ because e.g. there is no outgoing flight scheduled for n_1 or both flight connections are not symmetrical. Again, we can choose whether we repair a violation of β or α. We decide to repair a violation of α, i.e. we specify an outgoing flight for n_1 and obtain as our new instance

$I''' = \{$`fly`(Amsterdam,New York), `fly`(New York,n_1), `fly`(n_1,n_2)$\}$,

where $n_2 \in \Delta_{null} \backslash dom(I'')$. We note that $I''' \models \exists y$ `fly`(n_1,y), so the constraint is locally satisfied.
The reader may observe that if we continue to repair violations of α, this process will never terminate, which means that the chase does not terminate in the general case.

3.2 Formal Description

Let Σ be a set of TGDs and EGDs and I an instance, represented as a set of $\mathcal{R}$-atoms. Further let $\alpha \in \Sigma$ be a TGD and $\overline{x}$ the list of universally quantified variables in α. We say that α is applicable to I if $I \not\models \alpha$, i.e. if there is a homomorphism μ from $body(\alpha)$ to I and μ cannot be extended to a homomorphism $\mu' \supseteq \mu$ from $head(\alpha)$ to I. In such a case the chase step

$I \overset{\alpha,\mu(\overline{x})}{\longrightarrow} J$

is defined as in the following way. We take a homomorphism ν such that:

- ν agrees with μ on all universally quantified variables in α,
- for every existentially quantified variable y in α we choose a labeled null $n_y \in \Delta_{null} \backslash dom(I)$ and define $\nu(y) := n_y$.

We set J to be $I \cup \nu(head(\alpha))$.

Let $\alpha \in \Sigma$ be an EGD and $\overline{x}$ the list of universally quantified variables in α. We say that α is applicable to I if there is a homomorphism μ from $body(\alpha)$ to I and $\mu(x_i) \neq \mu(x_j)$, where $head(\alpha) = \{x_i = x_j\}$. In such a case the chase step

$$I \xrightarrow{\alpha, a} J$$

is defined as follows. We set J to be

- I except that all occurrences of $\mu(x_j)$ are substituted by $\mu(x_i) =: a$, if $\mu(x_j)$ is a labeled null,
- I except that all occurrences of $\mu(x_i)$ are substituted by $\mu(x_j) =: a$, if $\mu(x_i)$ is a labeled null,
- undefined, if both $\mu(x_j)$ and $\mu(x_i)$ are constants. In this case we say that the chase fails.

A chase sequence is an exhaustive application of applicable constraints

$$I_0 \xrightarrow{\alpha_0, \overline{a}_0} I_1 \xrightarrow{\alpha_1, \overline{a}_1} \ldots,$$

where we impose no strict order what constraint must be applied in case several constraints apply. If this sequence is finite, say I_r being its final element, the chase terminates and its result I_0^{Σ} is defined as I_r. The length of this chase sequence is r. Note that different orders of application of applicable constraints may lead to a different chase result. However, as proved in [Fagin et al., 2005], two different finite chase orders lead to homomorphically equivalent results if these exist. Therefore, we can write I^{Σ} for the result of the chase on an instance I under constraints Σ. It has been shown in [Maier et al., 1979; Beeri and Vardi, 1984; Johnson and Klug, 1982] that $I^{\Sigma} \models \Sigma$ (cf. Proposition 1). In case that a chase step cannot be performed (e.g., because a homomorphism would have to equate two constants) the chase result is undefined. In case of an infinite chase sequence, we also say that the result is undefined.

Proviso. We will make a simplifying assumption. Let I be a database instance and Σ some constraint set. Without loss of generality we can assume that whenever two labeled nulls, say n_1, n_2, are equated by the chase and $n_1 \in dom(I)$, then all occurrences of n_2 are mapped to n_1 in the chase step. This does not affect chase termination as substituting n_1 with n_2 would lead to an isomorphic instance.

3.3 A Formal Example

We will review the example from Section 3.1 more formally with the actual definition of the chase at hand. Consider Σ and I from Section 3.1 again.
We have a homomorphism

$$h = \{x_1 \mapsto \text{Amsterdam}, \ x_2 \mapsto \text{New York}\}$$

from $body(\beta)$ to I. Note that the same variables occur in $body(\beta)$ and $head(\beta)$. It cannot be extended to a homomorphism that maps $head(\beta)$ to I because `fly`(New York, Amsterdam) $\notin I$. Therefore, β is applicable to I. The chase step yields the new instance

$$\begin{aligned} I' &= I \cup h(head(\beta)) \\ &= \{\texttt{fly}(\text{Amsterdam,New York}), \ \texttt{fly}(\text{New York,Amsterdam})\}. \end{aligned}$$

It holds that $I' \models \Sigma$, so the chase terminates and returns I'.

Like in Section 3.1 another option would have been to repair the violation of α instead of β. We have that h is also a homomorphism from $body(\alpha)$ to I and it cannot be extended in such a way that $head(\alpha)$ can be mapped to I. Therefore, we define

$$h \subseteq h' := \{x_1 \mapsto \text{Amsterdam}, \ x_2 \mapsto \text{New York}, \ y \mapsto n_1\},$$

where $n_1 \in \Delta_{null} \backslash dom(I)$, and obtain the instance

$$I'' = I \cup h'(head(\alpha)) = \{\texttt{fly}(\text{Amsterdam,New York}), \ \texttt{fly}(\text{New York},n_1)\}.$$

3.4 Logical Properties

Having introduced the basics of the chase algorithm, we now turn our attention to more general logical properties of it, namely its termination and its property to output universal solutions.
As we have already seen in the example in Section 3.1, the chase does not necessarily terminate on an arbitrary input. This is not a mere coincidence as chase termination is an undecidable property:

Theorem 2. (see [Deutsch et al., 2008]) Consider an instance I and a set Σ of TGDs.

- It is undecidable whether there is some chase sequence of I and Σ that terminates.

- It is undecidable whether all chase sequences of I and Σ terminate.

This still holds if $I = \emptyset$.

Because of Theorem 2 there is no algorithm that can decide chase termination, therefore we focus on sufficient conditions for chase termination, which is the central topic of this thesis.

The most important property of the chase is that it produces universal instances with respect to its input (strongly universal solutions in [Deutsch et al., 2008]). This is explained in the next theorem:

Theorem 3. (see [Beeri and Vardi, 1984]) Consider an instance I and a set Σ of constraints. Let I^Σ be defined. For all database instances J such that $J \models \Sigma$ and there is a homomorphism from I to J, there is a homomorphism from I^Σ to J.

Intuitively, this theorem states that a chase result is the most general instance that satisfies Σ and I, which is why I^Σ is called a universal solution [Fagin et al., 2005; Deutsch et al., 2008]. It is exactly this property that makes the chase useful for so many applications.
As an easy corollary we obtain:

Theorem 4. (see [Beeri and Vardi, 1984]) Consider an instance I and a set Σ of constraints. Let J, J' be two chase results obtained by a different order of constraint application. It holds that J and J' are homomorphically equivalent, i.e. there is a homomorphism from J to J' and vice versa.

3.5 The Oblivious Chase

We will not only use the chase but also a variant of it, called the oblivious chase. Basically, the idea is that an oblivious chase step applies even if a constraint is satisfied. We illustrate this by an example.

Example 5. Consider the constraint set $\Sigma = \{\mathtt{R}(x_1, x_2) \rightarrow \exists y\ \mathtt{R}(x_1, y)\}$ and note that this constraint is a tautology. To be more explicit, we examine the constraint set together with the instance $\{\mathtt{R}(a, b)\}$. Although it holds that $I \models \Sigma$, the oblivious chase makes a modification here, that is to say, an atom of the form $\mathtt{R}(a, n_1)$ is added to the instance, where $n_1 \in \Delta_{null}$. Yet, the oblivious chase does not terminate. It will successively add atoms $\mathtt{R}(a, n_2)$, $\mathtt{R}(a, n_3)$, $\mathtt{R}(a, n_4) \ldots$, where $n_2, n_3, n_4, \ldots \in \Delta_{null}$.

A more formal definition for an oblivious chase step is the following one. Consider a TGD of the syntactic form $\forall \overline{x} \varphi$. The oblivious step applies to an instance I if there is a homomorphism μ from $body(\forall \overline{x} \varphi)$ to I. In such a case the oblivious chase step $I \overset{*, \forall \overline{x} \varphi, \mu(\overline{x})}{\longrightarrow} J$ is defined as follows. We define a homomorphism ν such that:

- ν agrees with μ on all universally quantified variables in φ,
- for every existentially quantified variable y in $\forall \overline{x} \varphi$ we choose a "fresh" labeled null $n_y \in \Delta_{null}$ and define $\nu(y) := n_y$.

We set J to be $I \cup \nu(head(\forall \overline{x} \varphi))$. An oblivious chase step for an EGD is a chase step for an EGD except that we also add an $*$ on the arrow (like in the case of TGDs) that indicates the step. Intuitively, an oblivious chase step always applies when the body of a constraint can be mapped to an instance even if the constraint is satisfied.

The essential point here is to keep in mind that an oblivious chase step is applicable to an instance even if the head of the constraint is satisfied by the instance. This thesis will not be about the oblivious chase, but we will often use it for technical reasons.

3.6 Previous Results on Chase Termination

In the past, research effort has been spent on chase termination. More precisely, conditions on the constraint set have been found such that if the constraint set satisfies these conditions then every chase sequence starting with every finite database is finite. This section explains the methods used in these previous works.

3.6.1 Cascading of Labeled Nulls

In order to be able to understand how the techniques that ensure chase termination work, we first need to clarify why the chase may not terminate. To be more precise, we identify a bad event that occurs in non-terminating chase sequences over and over again. The absence of infinitely many of such bad events then ensures the termination of every chase sequence. How can such a bad event be structured? We resume the example from the Introduction. Consider the set of TGDs $\Sigma := \{\alpha\}$ and the instance I^0, where

$$\begin{aligned} \alpha &:= \texttt{fly}(x_1, x_2) \rightarrow \exists y\ \texttt{fly}(x_2, y), \\ I^0 &:= \{\texttt{fly}(\text{Amsterdam,New York})\}. \end{aligned}$$

As pointed out earlier, we have an infinite chase sequence

$I^1 = \{\texttt{fly}(\text{Amsterdam,New York}), \texttt{fly}(\text{New York}, n_1)\}$,
$I^2 = \{\texttt{fly}(\text{Amsterdam,New York}), \texttt{fly}(\text{New York}, n_1), \texttt{fly}(n_1, n_2)\}$,
$I^3 = \{\texttt{fly}(\text{Amsterdam,New York}), \texttt{fly}(\text{New York}, n_1), \texttt{fly}(n_1, n_2), \texttt{fly}(n_2, n_3)\}$,
...

We observe that in this chase sequence it will infinitely many times occur that a labeled null, which is not in $dom(I^0)$, is copied from the body to the head of a constraint, while a new labeled null is created by some existential quantifier. We call a situation in which this happens *cascading of labeled nulls.* For instance, consider the chase step from I^2 to I^3. We have that $I^2 \models \texttt{fly}(n_1, n_2)$, but $I^2 \not\models \exists y\ \texttt{fly}(n_2, y)$. Applying this step, we see that n_2 is copied from the body to the head of the constraint, while n_3 is newly created. So, we have a cascading of labeled nulls here.

We formalize the notion of cascading of labeled nulls as follows:

Definition 6. Let $\mathcal{S} = I_0 \xrightarrow{\alpha_0, \bar{a}_0} I_1 \xrightarrow{\alpha_1, \bar{a}_1} \ldots$ be a chase sequence. Then, $C_{\mathcal{S}} := \{\ i \in \mathbb{N}_0 \mid dom(I_{i+1}) \backslash dom(I_i) \neq \emptyset \neq (dom(I_{i+1} \backslash I_i) \backslash dom(I_0)) \cap dom(I_i)\ \}$.

It is easy to observe that if we have such as cascading only finitely many times in a chase sequence, then this is equivalent to the chase sequence itself being finite.

Proposition 7. Let $\mathcal{S} = I_0 \xrightarrow{\alpha_0, \bar{a}_0} I_1 \xrightarrow{\alpha_1, \bar{a}_1} \ldots$ be a chase sequence. Then, it holds that $\mathcal{S}$ is finite $\Longleftrightarrow$ $C_{\mathcal{S}}$ is finite.

With this instrument at hand we will be able to better understand the following sections.

3.6.2 Weak Acyclicity

In the following, we are only interested in constraints for which every chase sequence is finite. In [Fagin et al., 2005] weak acyclicity was introduced, which is the starting point for our work.

The key idea of weak acyclicity is to statically estimate the flow of data between the database positions during the execution of the chase. It syntactically asserts that there is no infinite cascading of labeled nulls, i.e. $C_{\mathcal{S}}$ being finite for every chase sequence $\mathcal{S}$.

Definition 8. (see [Fagin et al., 2005]) The dependency graph dep(Σ) of a set of constraints Σ is the directed graph defined as follows. The set of vertices is the set of positions that occur in some TGD in Σ. There are two kinds of edges. Add them as follows: for every TGD $\forall\overline{x}(\phi(\overline{x}) \rightarrow \exists\overline{y}\psi(\overline{x},\overline{y})) \in \Sigma$ and for every x in $\overline{x}$ that occurs in ψ and every occurrence of x in ϕ in position π_1

- for every occurrence of x in ψ in position π_2, add an edge $\pi_1 \rightarrow \pi_2$ (if it does not already exist).
- for every existentially quantified variable y and for every occurrence of y in a position π_2, add a special edge $\pi_1 \overset{*}{\rightarrow} \pi_2$ (if it does not already exist).

A set Σ of TGDs and EGDs is called *weakly acyclic* iff dep(Σ) has no cycles through a special edge.

Intuitively, normal edges track the flow of data between the database positions and special edges cover the case of cascading of null values. Hence, if the dependency graph contains no cycles through a special edge, it cannot happen that fresh null values are added to the instance over and over again. It was shown in [Fagin et al., 2005] that the chase terminates in polynomial-time data-complexity, i.e. in time polynomial in the size of the input instance's domain, for weakly acyclic constraint sets. Note that weak acyclicity can be decided in polynomial time. We illustrate the definition by an example for a situation in which weak acyclicity cannot guarantee the termination of the chase.

Example 9. The dependency graph for the constraint set $\Sigma := \{\alpha_1, \alpha_2, \alpha_3\}$ from Figure 3.1 is shown in Figure 3.2. We observe that Σ is not weakly acyclic, as witnessed by the self-loop special edge for position $\texttt{fly}^2$.

3.6.3 Stratification

In [Deutsch et al., 2008] stratification was set on top of the definition of weak acyclicity. The main idea of stratification is to decompose the given constraint set into independent subsets that are separately tested for weak acyclicity. More precisely, the decomposition splits the constraint set into subsets of constraints that may cyclically cause to fire each other. Termination guarantees for the full constraint set follow if weak acyclicity holds for each subset in the decomposition. However, we will prove that the stratification condition does not generally ensure chase termination, which is why we will discuss it in Section 7.1.

Sample Schema: `hasAirport`(c_id)
`fly`($c\_id1, c\_id2, dist$)
`rail`($c\_id1, c\_id2, dist$)

Constraint Set: $\Sigma = \{\alpha_1, \alpha_2, \alpha_3\}$, where

α_1 : If there is a flight connection between two cities, both of them have an airport:
`fly`(c_1, c_2, d) → `hasAirport`(c_1), `hasAirport`(c_2)

α_2 : Rail-connections are symmetrical:
`rail`(c_1, c_2, d) → `rail`(c_2, c_1, d)

α_3 : Each city that is reachable via plane has at least one outgoing flight scheduled:
`fly`(c_1, c_2, d) → $\exists c_3, d'$ `fly`(c_2, c_3, d')

Figure 3.1: Sample database schema and constraints.

3.6.4 Super-weak Acyclicity

In [Marnette, 2009], in the context of data exchange, weak acyclicity was improved to super-weak acyclicity. It is important to notice that this new termination condition is limited to sets of TGDs, i.e. EGDs are not covered by it. The overall idea is to check whether a constraint α can trigger a firing of another constraint β. A constraint set Σ is super-weakly acyclic if this trigger relation is acyclic. We now formally introduce super-weak acyclicity and follow closely the presentation in [Marnette, 2009].

Let a set of TGDs Σ be given. Let $\mathcal{P}(\Sigma)$ be a skolemization of Σ. A place (a, i) is a pair, where a is an atom of $\mathcal{P}(\Sigma)$ and $1 \leq i \leq ar(a)$. Two places (a, i) and (a', i') are called unifiable, $(a, i) \sim (a', i')$, iff $i = i'$ and there are substitutions s, s' such that $s(a) = s'(a')$, where a substitution is a mapping from logical terms to logical terms.

Example 10. Let the places $(\mathtt{B}(f(x), x), 1)$ and $(\mathtt{B}(x, x), 1)$ be given. It holds that $(\mathtt{B}(f(x), x), 1) \not\sim (\mathtt{B}(x, x), 1)$ because x should be mapped to itself and at the same time to $f(x)$.

Given a constraint $\alpha \in \Sigma$ and an existential variable y in the head of α, we denote

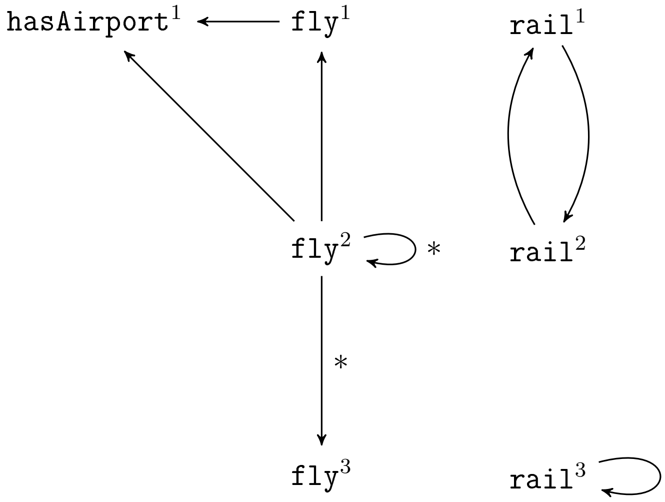


Figure 3.2: Dependency graph for Σ from Example 9.

by $\mathrm{Out}(\alpha, y)$ the set of places in the head of $\mathcal{P}(\alpha)$ where a term of the form $f_y^\alpha(...)$ occurs.

Example 11. Let the TGD $\alpha := \mathtt{A}(x) \to \exists y\ \mathtt{B}(x,y),\ \mathtt{B}(y,x),\ \mathtt{C}(y)$ be given. Then,

$$\mathcal{P}(\alpha) = \mathtt{A}(x) \to \mathtt{B}(x, f_y^{\alpha_1}(x)),\ \mathtt{B}(f_y^{\alpha_1}(x), x),\ \mathtt{C}(f_y^{\alpha_1}(x)).$$

It holds that $\mathrm{Out}(\alpha, y) = \{(\mathtt{B}(x, f_y^{\alpha_1}(x)), 2), (\mathtt{B}(f_y^{\alpha_1}(x), x), 1), (\mathtt{C}(f_y^{\alpha_1}(x)), 1)\}$.

Given a universally quantified variable x in the body of α, we denote by $\mathrm{In}(\alpha, x)$ the set of places in the body of α in which x occurs.

Example 12. Consider α from Example 11. It holds that $\mathrm{In}(\alpha, x) = \{(\mathtt{A}(x), 1)\}$.

Given two sets of places Q, Q' we write $Q \lhd Q'$ iff for all $q \in Q$ there is some $q' \in Q'$ such that $q \sim q'$.

Example 13. Consider the sets of places

$$P_1 := \{(\mathtt{C}(x), 1), (\mathtt{B}(f(x), y), 2)\} \text{ and}$$
$$P_2 := \{(\mathtt{C}(y), 1), (\mathtt{B}(x, f(y)), 2)\}.$$

Clearly,

$(\mathtt{C}(x), 1) \sim (\mathtt{C}(y), 1)$ and
$(\mathtt{B}(f(x), y), 2) \sim (\mathtt{B}(x, f(y)), 2)$.

Therefore, $P_1 \lhd P_2$.

We set $\mathrm{Move}(\Sigma, Q)$ as the smallest set of places Q' such that $Q \subseteq Q'$ and for all expressions $r = B_r \to H_r \in \mathcal{P}(\Sigma)$ and for all variables x it holds that if $\gamma_x(B_r) \lhd Q'$ then $\gamma_x(H_r) \subseteq Q'$, where $\gamma_x(B_r)$ and $\gamma_x(H_r)$ denote the set of places in B_r and H_r where x occurs.

Example 14. Consider the constraint $\Sigma = \{\alpha_1, \alpha_2\}$, where

$\alpha_1 := \mathtt{A}(x, y) \to \exists z\ \mathtt{B}(x, z), \mathtt{B}(z, y)$ and
$\alpha_2 := \mathtt{B}(x, y), \mathtt{B}(y, z) \to \mathtt{C}(x, z)$.

The logic program $\mathcal{P}(\Sigma)$ consists of the two constraints

$\mathtt{A}(x, y) \to \mathtt{B}(x, f_z^{\alpha_1}(x, y)), \mathtt{B}(f_z^{\alpha_1}(x, y), y)$ and
$\mathtt{B}(x, y), \mathtt{B}(y, z) \to \mathtt{C}(x, z)$.

It follows from the definitions that $\mathrm{Move}(\Sigma, \{(\mathtt{A}(z_1, z_2), 1)\})$ is the set of places

$$\begin{array}{lll} (\mathtt{A}(z_1, z_2), 1), & (\mathtt{B}(x, f_z^{\alpha_1}(x, y)), 1), & (\mathtt{B}(x, f_z^{\alpha_1}(x, y)), 2), \\ (\mathtt{B}(f_z^{\alpha_1}(x, y), y), 1), & (\mathtt{B}(f_z^{\alpha_1}(x, y), y), 2), & (\mathtt{C}(x, z), 1), (\mathtt{C}(x, z), 2). \end{array}$$

Definition 15. [Marnette, 2009] Given a set of TGDs Σ and $\alpha, \beta \in \Sigma$, we say that α triggers β, $\alpha \hookrightarrow_\Sigma \beta$, if there exists an existentially quantified variable y in the head of α and a universally quantified variable x occurring both in the head and body of β such that $\mathrm{In}(\beta, x) \lhd \mathrm{Move}(\Sigma, \mathrm{Out}(\alpha, y))$.

Please note that if $\alpha \hookrightarrow_\Sigma \beta$, then α contains existentially quantified variables. Intuitively, the trigger relation holds for two TGDs α and β if a firing of α can cause a firing of β (in the case that β could not fire before). This definition is the main ingredient for super-weak acyclicity.

Definition 16. [Marnette, 2009] Given a set of TGDs Σ. We say that Σ is super-weakly acyclic iff the trigger relation $\hookrightarrow_\Sigma$ is acyclic.

It was proved in [Marnette, 2009] that super-weak acyclicity guarantees the termination of the chase for every database and every chase sequence taken and is strictly weaker than weak acyclicity. The method of super-weak acyclicity is strongly related to the termination conditions we introduce in the subsequent chapters. We defer this discussion to Chapter 10 and finish this chapter with an example that was obtained as a result of an email discussion with the author of [Marnette, 2009].

Example 17. Let $\Sigma = \{\alpha_1, \alpha_2\}$, where

$\alpha_1 := \texttt{A}(x) \to \exists y\ \texttt{B}(x,y),\ \texttt{B}(y,x),\ \texttt{C}(y)$ and
$\alpha_2 := \texttt{B}(x,x),\ \texttt{C}(y) \to \texttt{A}(x),\ \texttt{C}(y)$.

The logic program $\mathcal{P}(\Sigma)$ consists of the two constraints

$\texttt{A}(x) \to \texttt{B}(x, f_y^{\alpha_1}(x)),\ \texttt{B}(f_y^{\alpha_1}(x), x),\ \texttt{C}(f_y^{\alpha_1}(x))$ and
$\texttt{B}(x,x),\ \texttt{C}(y) \to \texttt{A}(x),\ \texttt{C}(y)$.

It holds that $\hookrightarrow_\Sigma = \emptyset$. The tuple (α_1, α_2) is not in $\hookrightarrow_\Sigma$ because the relation symbols in the body and the head are disjoint and $(\alpha_1, \alpha_1) \notin \hookrightarrow_\Sigma$ because $\texttt{B}(x, f_y^{\alpha_1}(x))$ and $\texttt{B}(f_y^{\alpha_1}(x), x)$ cannot unify with $\texttt{B}(x,x)$. Therefore, it is super-weakly acyclic although it is not weakly acyclic.

We want to point out that the constraint α_2 is unnatural because the atom $\texttt{C}(y)$ appears both in its body and head. Nevertheless, this example will be quite significant for showing that there are constraint sets which we cannot recognize with the help of some of our chase termination conditions that we develop throughout this thesis. The main reason for this constraint not being contained in some of our termination conditions is $\texttt{C}(y)$ appearing both in α_2's body and head.

Chapter 4

Classical Areas of Application

> **Riccardo:** "What is all this good for?"
> **Alice:** "Guess what a Swiss Army knife can be good for?"
> **Riccardo:** "Having a meal?"

This chapter clarifies why the chase can be compared to a Swiss Army knife tool for many database problems. We will sketch the most important applications of the chase and show that all of them ultimately depend only on the termination of the chase, i.e. these methods can always be applied if the chase algorithm terminates. If we can make the chase terminate in strictly more cases, then this is a contribution to all research sub-areas mentioned next.

We will explain how the implication problem for TGDs and EGDs can be solved, how it can be applied to query optimization in the Chase & Backchase algorithm. Finally, we will explicate its importance in data exchange and in answering queries using views.

The goal of this chapter is not to make the reader an expert in these research areas, but to show him how the chase is used in these applications and to make him understand that they solely depend on the termination of the chase.

We also want to mention that there are several other application scenarios we do not explain in this thesis which use the chase as a subroutine, like data integration [Lenzerini, 2002], peer data exchange [Fuxman et al., 2005] and probabilistic databases [Olteanu et al., 2009].

4.1 The Implication Problem

The implication problem is fundamental for many applications. It is defined as follows. Given a set of constraints Σ and a single constraint α, decide whether $\Sigma \models \alpha$ holds. So, the implication problem is to decide whether a given constraint is logically implied by a constraint set.

Unfortunately, the implication problem for keys and foreign keys is undecidable (see [Fan and Siméon, 2000; Fan and Libkin, 2002]). This is why we need to restrict ourselves to meaningful subclasses of constraints, which guarantee the decidability of the implication problem. The following result can be obtained using the chase.

Theorem 18. (see [Beeri and Vardi, 1984]) Consider a set Σ of constraints and a constraint α such that $body(\alpha)^{\Sigma}$ is defined.

If α is a TGD, then

$$\Sigma \models \alpha \iff \text{there is a homomorphism from } head(\alpha) \text{ to } body(\alpha)^{\Sigma}.$$

If α is an EGD and $head(\alpha) = \{x_i = x_j\}$, then

$$\begin{aligned}\Sigma \models \alpha \iff & \text{ for every homomorphism } h \text{ from } body(\alpha) \text{ to } body(\alpha)^{\Sigma} \\ & \text{ it holds that } h(x_i) = h(x_j).\end{aligned}$$

The theorem illustrates how the implication problem for TGDs and EGDs can be reduced to the chase if it terminates.

4.2 Query Optimization: Chase & Backchase

The chase is a central tool in logic-based optimization of conjunctive queries. It can be used to reduce the problem of query containment under constraints to general query containment. This method is used in the well-known Chase & Backchase (C&B) algorithm [Popa, 2000; Deutsch et al., 2006]. For the case of relational databases an implementation and an evaluation of C&B can be found in [Popa et al., 2000]. It turned out that it can post significant time reductions in query execution.

It has also been applied in the context of XML [World Wide Web Consortium, 2003d] data and XQuery [World Wide Web Consortium, 2000] in the MARS system [Deutsch and Tannen, 2003]. Here, the idea is to translate the XQuery expression to a relational query, apply C&B, and translate the result back to an XQuery expression.

We start the presentation of C&B by defining conjunctive queries and their containment problem, which forms the basis of the algorithm.

A conjunctive query q is an expression of the form

$$ans(\overline{x}) \leftarrow \varphi(\overline{x}, \overline{z}),$$

where φ is a conjunction of relational atoms, $\overline{x}$, $\overline{z}$ are sequences of variables and constants, and it holds that every variable in $\overline{x}$ also occurs in φ. If $\overline{x}$ is empty we call the query boolean. We denote by $db(q)$ the set of atoms in φ and by $head(q)$ the atom $ans(\overline{x})$.
The semantics of such a query q on database instance I is defined as

$$q(I) := \{\overline{a} \in \Delta^{|\overline{x}|} | I \models \exists \overline{z} \varphi(\overline{a}, \overline{z})\}.$$

Let q, q' be conjunctive queries and Σ a set of constraints. We say that q is contained in q' under Σ, $q \sqsubseteq_\Sigma q'$, if and only if for all database instances I such that $I \models \Sigma$ it holds that $q(I) \subseteq q'(I)$. We write $q \sqsubseteq q'$ if and only if $q \sqsubseteq_\emptyset q'$ as an abbreviation. The following result is folklore.

Theorem 19. (see [Chandra and Merlin, 1977]) Let q, q' be two conjunctive queries. It holds that $q \sqsubseteq q'$ if and only if there is a homomorphism h that maps $db(q') \cup \{head(q')\}$ to $db(q) \cup \{head(q)\}$.

So far, the chase is not involved. However, when non-empty constraint sets Σ come into play, then the chase can be used to reduce the containment problem under constraints to the containment problem under an empty constraint set as the next theorem states.

Theorem 20. (see [Johnson and Klug, 1982]) Let q, q' be two conjunctive queries and Σ a set of constraints such that $db(q)^\Sigma$ is defined. It holds that $q \sqsubseteq_\Sigma q'$ if and only if $(\{head(q)\} \cup db(q))^\Sigma \sqsubseteq q'$.

We see that this method solely depends on the termination of the chase. But how can we apply this in query optimization? The next theorem answers this question, but before that, we need to introduce what a minimal rewriting of a conjunctive query is. A Σ-equivalent rewriting of a query q is a query q' such that $q \sqsubseteq_\Sigma q'$ and $q' \sqsubseteq_\Sigma q$. A minimal rewriting of q under Σ is a conjunctive query q' such that q' is a Σ-equivalent rewriting of q and q' is minimal with respect to the number of subgoals in its body. Minimal rewritings are considered the most efficient representation of a given query because they usually have lower execution cost than the original query. Note that this is not a general law. In [Cheng et al., 1999] it was shown that introducing semantics-preserving join operations into a query can reduce the execution cost in some cases.

Theorem 21. (see [Deutsch et al., 2006]) Let q be a conjunctive query and Σ a set of constraints such that the chase with Σ has a terminating chase sequence for every input database. There is an algorithm (called C&B) that precisely outputs all minimal rewritings (up to isomorphism) of q under Σ.

The chase is used as a subprocedure of this algorithm. The first step in the Chase & Backchase algorithm is the computation of $(\{head(q)\} \cup db(q))^{\Sigma}$. All minimal rewritings of q are then subqueries of $db(q)^{\Sigma}$ that are Σ-equivalent to q. Σ-equivalence of the queries can be tested via Theorem 20. Again, the only property that is important here, is that the chase terminates. If it does, we can obtain minimal rewritings for conjunctive queries.

4.3 Data Exchange

Data exchange was first considered in [Fagin et al., 2005]. It is the problem of migrating a data instance from a source schema to a target schema such that the materialized data on the target schema satisfies the integrity constraints specified by it. Data exchange is very similar to data integration [Lenzerini, 2002] but the main difference is that the data is indeed materialized at the target schema, which is not always the case for data integration settings.

The mapping of the data from the source to the target schema is given by source-to-target TGDs. Additionally, the target schema specifies target constraints in form of EGDs and TGDs, which the imported data must satisfy. Exchanging the data from source to target can be seen as the execution of the chase with source-to-target plus target constraints. The resulting data instance is called a universal solution. Of course, the termination of the chase is a big issue there.
We now define what a data exchange problem is. Given a source database schema $\mathcal{R}_1$, a target database schema $\mathcal{R}_2$ such that $\mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset$, a set of constraints Σ over $\mathcal{R}_1 \cup \mathcal{R}_2$ and a database instance I over the source schema, find a database J over the target schema such that $I \cup J \models \Sigma$. Our constraint set Σ is the disjoint union of Σ_{st} and Σ_t. Σ_t is a set of constraints over $\mathcal{R}_2$. The TGDs in Σ_{st} are of the form $\varphi(\overline{x}) \to \exists \overline{y} \psi(\overline{x}, \overline{y})$, where φ contains only predicate symbols from the source schema and ψ only from the target schema. It was shown in [Fagin et al., 2005] that $I^{\Sigma} \backslash I$ is a solution to the data exchange problem if it is defined. As the result of the chase is not unique, solutions to data exchange problems are not unique either. What does that mean for conjunctive queries? Does the answer of a conjunctive query depend on the chase result? As it turns out, this is not the case. It was proved in [Fagin et al., 2005] that we always get the same answers, independently of the chase result materialized at the target. Again, we see that the data exchange problem crucially depends on the termination of the chase.

We also want to mention that the data exchange framework was extended in [Fuxman et al., 2005] to peer-to-peer networks. The computation of solutions to the peer data exchange problem also uses the chase.

4.4 Query Answering Using Views

In query answering using views we try to answer a database queries using materialized views, accessing non-view relations in the database only if necessary. The idea is that the definition of a view may involve some costly join operations that need to be computed by the query to be answered, too. If the view is materialized, we can probably use it as an intermediate result for the computation of the query answer, thus saving time and memory consumption. An exhaustive overview on this topic can be found in [Halevy, 2001; Deutsch et al., 2007].

In [Deutsch et al., 2006] it was shown that in some cases it is not possible to compute equivalent rewritings of the input query using views only, but a minimally-containing one. A minimally-containing rewriting is characterized as follows. Usually, the views are defined in terms of a set of TGDs Σ that explicate how the tuples in the views are obtained from the base relations. Let q be a conjunctive query. A conjunctive query q' is a minimally-containing rewriting of q under Σ and a set of views $\mathcal{V}$ if and only if $q \sqsubseteq_\Sigma q'$ and for all conjunctive queries q'' such that $q \sqsubseteq_\Sigma q''$ it follows that $q'' \sqsubseteq_\Sigma q'$.

Theorem 22. (see [Deutsch et al., 2006]) There is an algorithm that, given a conjunctive query q and a set of constraints Σ, computes a minimally-containing rewriting of q under Σ and $\mathcal{V}$ if $db(q)^\Sigma$ exists.

How can a minimally-containing rewriting under Σ be obtained from q? First, we compute $db(q)^\Sigma$ using the chase if this is possible. Then, we drop all atoms from $db(q)^\Sigma$ that do not belong to a view predicate in $\mathcal{V}$, except for the head predicate *ans*. We call the remaining part $head(q) \leftarrow view(db(q)^\Sigma)$. If this is a valid conjunctive query, i.e. all variables in the *ans* predicate appear also in some other predicate that is different from *ans*, then it is also a minimally-containing rewriting of q under Σ and $\mathcal{V}$.
Again, we see that the applicability of this method depends only on the termination of the chase algorithm.

Chapter 5

Four Flavors of Chase Termination

Sergio: "What next?"
Alice: "Let's spice things up."

The chase algorithm [Maier et al., 1979; Beeri and Vardi, 1984; Johnson and Klug, 1982] has had a tremendous success since the time of its proposal. Initially, it was only used to decide logical implication of first-order constraints. Today, it is the key technique in all application areas mentioned in Section 4. The basic idea of this algorithm is simple: as input it takes a database and a set of constraints and then fixes constraint violations of the database instance. But unfortunately, the chase does not terminate for every input. Even worse, it has just recently been proved that, given a set of constraints, it is undecidable whether the chase terminates for every database instance, see Section 3.4. Although this result was just recently obtained, in the past much research effort has been spent to find syntactic restrictions for the constraints such that the chase terminates on every database instance and every chase sequence [Abiteboul et al., 1995; Johnson and Klug, 1982; Cosmadakis and Kanellakis, 1986; Deutsch and Tannen, 2005; Deutsch et al., 2006; Fagin et al., 2005; Deutsch et al., 2008].

So far, the whole research on chase termination has concentrated on finding syntactic restrictions on the constraint set that ensure the termination of the chase for every database instance and every chase sequence taken. To the best of our knowledge, there has been no work to ensure chase termination for at least one chase sequence. To be more precise, we distinguish four flavors of chase termination, namely the termination of the chase

- for every database instance and every chase sequence,
- for every database instance and at least one chase sequence,
- for a given database instance and every chase sequence, and

- for a given database instance and at least one chase sequence.

The first flavor of termination is the usual one as considered by weak acyclicity or super-weak acyclicity. The latter three flavors have not yet been considered. They are less restrictive than the first one and may allow us to derive termination guarantees in cases where the first approach fails. We proceed with a more formal definition.

Definition 23. Let J denote a finite database instance. We denote our four flavors of chase termination as follows:

- $\mathrm{CT}_{\forall\forall} := \{\ \Sigma \mid$ for every finite database instance I and for every chase sequence the chase with Σ and I terminates $\}$,
- $\mathrm{CT}_{\forall\exists} := \{\ \Sigma \mid$ for every finite database instance I there exists a chase sequence such that the chase with Σ and I terminates $\}$,
- $\mathrm{CT}_{J,\forall} := \{\ \Sigma \mid$ for every chase sequence the chase with Σ and J terminates $\}$, and
- $\mathrm{CT}_{J,\exists} := \{\ \Sigma \mid$ there exists a chase sequence such that the chase with Σ and J terminates $\}$.

The core part of this thesis will be the study of these classes. As already mentioned above, it holds that weak acyclicity and super-weak acyclicity are both contained in $\mathrm{CT}_{\forall\forall}$. We were rather surprised to find that $\mathrm{CT}_{\forall\exists}, \mathrm{CT}_{J,\forall}$ and $\mathrm{CT}_{J,\exists}$ have not yet been considered and therefore, this thesis constitutes the first study of them. We start by repeating a result that we already mentioned in the preliminaries, now stating it in the light of the previous definition.

Theorem 24. (see Theorem 2) It holds that $\mathrm{CT}_{\emptyset,\forall}$ and $\mathrm{CT}_{\emptyset,\exists}$ are undecidable.

There are some obvious subset-relationships between these four classes. We state them in the following. Their proof follows straightforwardly from the definitions and is therefore omitted here.

Proposition 25. Let J be a finite database instance.

1. $\mathrm{CT}_{\forall\forall} \subseteq \mathrm{CT}_{J,\forall} \subseteq \mathrm{CT}_{J,\exists}$,
2. $\mathrm{CT}_{\forall\forall} \subseteq \mathrm{CT}_{\forall\exists} \subseteq \mathrm{CT}_{J,\exists}$,

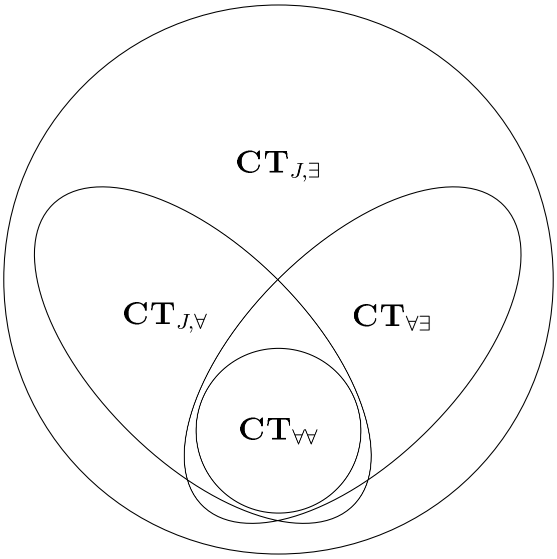

$\mathbf{CT}_{J,\exists}$
$\mathbf{CT}_{J,\forall}$
$\mathbf{CT}_{\forall\exists}$
$\mathbf{CT}_{\forall\forall}$

Figure 5.1: A summary of Proposition 25 and Theorem 26.

3. $\bigcap_{I \text{ finite instance}} \mathrm{CT}_{I,\forall} = \mathrm{CT}_{\forall\forall}$, and
4. $\bigcap_{I \text{ finite instance}} \mathrm{CT}_{I,\exists} = \mathrm{CT}_{\forall\exists}$.

Apart from these obvious consequences of the definitions, we can also show that $\mathrm{CT}_{\forall\exists}, \mathrm{CT}_{J,\forall}$ and $\mathrm{CT}_{J,\exists}$ contain constraint sets which are not already in $\mathrm{CT}_{\forall\forall}$, i.e. we will show that all other containment relationships of our four classes do not hold.

Theorem 26.

1. There is some J such that $\mathrm{CT}_{J,\forall} \backslash \mathrm{CT}_{\forall\forall} \neq \emptyset$.
2. There is some J such that $\mathrm{CT}_{J,\exists} \backslash \mathrm{CT}_{J,\forall} \neq \emptyset$.
3. It holds that $\mathrm{CT}_{\forall\exists} \backslash \mathrm{CT}_{\forall\forall} \neq \emptyset$.
4. There is some J such that $\mathrm{CT}_{J,\exists} \backslash \mathrm{CT}_{\forall\forall} \neq \emptyset$.
5. There is some J such that $\mathrm{CT}_{J,\forall} \backslash \mathrm{CT}_{\forall\exists} \neq \emptyset$.
6. There is some J such that $\mathrm{CT}_{\forall\exists} \backslash \mathrm{CT}_{J,\forall} \neq \emptyset$.

Proof.

1. We can set $J := \{\texttt{E}(a,a)\}$, $I := \{\texttt{E}(a,b)\}$ and $\Sigma := \{\texttt{E}(x_1,x_2) \to \exists y\, \texttt{E}(x_2,y)\}$. We see that the chase will terminate for J and run forever on I.

2. We can set $J := \{\texttt{E}(a,b)\}$ and $\Sigma := \{\texttt{E}(x_1,x_2) \to \exists y\ \texttt{E}(x_2,y);\ \texttt{E}(x_1,x_2) \to \texttt{E}(x_2,x_1)\}$. We see that the chase will terminate if we apply the second constraint on J. We can create an infinite chase sequence by never applying the second constraint.

3. The constraint set $\Sigma = \{\alpha_1, \alpha_2\}$ can be used where

   $\alpha_1 = \texttt{E}(x_1,x_2) \to \exists y\ \texttt{E}(x_2,y)$, and
   $\alpha_2 = \texttt{E}(x_1,x_2) \to \texttt{E}(x_2,x_1)$.

   It holds that $\Sigma \notin \mathrm{CT}_{\forall\forall}$ because $\{\alpha_1\}$ has no terminating chase sequence on the instance $\{\texttt{E}(a,b)\}$. But we have that $\Sigma \in \mathrm{CT}_{\forall\exists}$ because the chase starting with α_2 alone terminates for every input and $\alpha_2 \models \alpha_1$.

4. Take J to be $\{\texttt{E}(a,a)\}$ and $\Sigma := \{\texttt{E}(x_1,x_2) \to \exists y\ \texttt{E}(x_2,y)\}$. Note that $\Sigma \notin \mathrm{CT}_{\forall\forall}$ because there is no terminating chase sequence for the instance $\{\texttt{E}(a,b)\}$.

5. Analogously to the previous bullet's proof we take J to be $\{\texttt{E}(a,a)\}$ and $\Sigma := \{\texttt{E}(x_1,x_2) \to \exists y\ \texttt{E}(x_2,y)\}$. Note that $\Sigma \notin \mathrm{CT}_{\forall\exists}$ because there is no terminating chase sequence for the instance $\{\texttt{E}(a,b)\}$.

6. The constraint set $\Sigma = \{\alpha_1, \alpha_2\}$ can be used where

   $\alpha_1 = \texttt{E}(x_1,x_2) \to \exists y\ \texttt{E}(x_2,y)$, and
   $\alpha_2 = \texttt{E}(x_1,x_2) \to \texttt{E}(x_2,x_1)$.

   We have that $\Sigma \in \mathrm{CT}_{\forall\exists}$ because the chase with α_2 alone terminates for every input and $\alpha_2 \models \alpha_1$. However, given the start instance $\{\texttt{E}(a,b)\}$, we obtain a non-terminating chase sequence by chasing with α_1 alone. □

A graphical overview of all containments is given in Figure 5.1. Note that all containments are strict. We can see that by taking $\mathrm{CT}_{\forall\exists}, \mathrm{CT}_{J,\forall}$ and $\mathrm{CT}_{J,\exists}$ into account, and not only $\mathrm{CT}_{\forall\forall}$, we can detect more situations in which the chase can be safely applied without the risk of non-termination.

As we have already mentioned, $\mathrm{CT}_{\emptyset,\exists}$ and $\mathrm{CT}_{\emptyset,\forall}$ are undecidable. It is open whether $\mathrm{CT}_{\forall\forall}$ and $\mathrm{CT}_{\forall\exists}$ are decidable or not. This is why we focus on decidable subsets. The subsequent chapters will introduce decidable fragments of all four flavors of chase termination.

Chapter 6

A Study of $\mathrm{CT}_{\forall\forall}$

> **Sergio:** "What next?"
> **Alice:** "Let's do some fancy theory stuff."
> **Riccardo:** "I'm getting a schnitzel."

As a minimalistic motivating example for our study of novel chase termination conditions in $\mathrm{CT}_{\forall\forall}$ let us consider the constraint set Σ from Figure 6.1, which is settled in our graph database schema from the Introduction. As we shall see later, the chase with Σ terminates for every database instance. Still, none of the existing termination conditions is able to recognize termination for this constraint set, i.e. Σ is neither weakly acyclic nor super-weakly acyclic. With the techniques and tools that we develop within this section, we will be able to guarantee chase termination for Σ on every database instance and every chase sequence taken.

In this chapter we discuss the sufficient data-independent chase termination conditions presented in Figure 1.2 and put them into context. First, we will introduce the novel classes of *c-stratified*[1] and *safe* constraints, which strictly generalizes weak acyclicity, but is different from super-weak acyclicity. Building upon the definition of safety and c-stratification, we will then introduce *safely restricted* and *inductively restricted* constraints as a consequent advancement of our ideas. The latter class strictly subsumes safe restriction. Finally, we will define a hierarchy of sufficient termination condition on top of inductively restricted constraints, the so-called $\forall\forall$-T-hierarchy. Each level $\forall\forall\text{-T}[k]$ in this hierarchy is strictly contained in the next level $\forall\forall\text{-T}[k+1]$. Our novel sufficient termination conditions extend the applicability of the chase algorithm, as they guarantee chase termination for larger classes of constraints than previous conditions. We show that they all guarantee polynomial length for every chase sequence like weak acyclicity, thus a chase result can be efficiently computed.

---

[1] C-stratification is the corrected version of stratification in [Deutsch et al., 2008].

Schema: $S(n)$, $E(src, dst)$
Constraint Set: $\Sigma := \{\alpha\}$, where

α : If x_2 is a special node and has some predecessor x_1, then x_1 has itself a predecessor:
$S(x_2),\ E(x_1, x_2) \rightarrow \exists y\ E(y, x_1)$

Figure 6.1: A sample constraint.

6.1 C-Stratification

In Chapter 7 we will show that the stratification condition introduced in [Deutsch et al., 2008] does not ensure chase termination in the sense of $\mathrm{CT}_{\forall\forall}$, but in the sense of $\mathrm{CT}_{\forall\exists}$. However, according to this thesis's structure, we already propose a possible correction of stratification, called *c-stratification*, which has the property of ensuring the termination of the chase independently of the data and the chase sequence used. The underlying ideas are similar to [Deutsch et al., 2008]: decompose the constraint set such that if a termination guarantee can be made for every subset, then the same guarantee can be made for the overall set.

Definition 27. Given two TGDs or EGDs $\alpha, \beta \in \Sigma$ we define $\alpha \prec_c \beta$ iff there exists a relational database instance I and $\overline{a}, \overline{b}$ such that

1. $I \models \beta(\overline{b})$,
2. $I \overset{*,\alpha,\overline{a}}{\rightarrow} J$, and
3. $J \not\models \beta(\overline{b})$.

Note that in this definition different from [Deutsch et al., 2008] we use an oblivious chase step and not a standard chase step. We give two examples to illustrate this definition.

Example 28. Let the constraint set $\Sigma = \{\alpha_1, \alpha_2\}$ be given, where and let the constraint

$\alpha_1 := \mathtt{S}(x_1, x_2) \rightarrow \exists z\ \mathtt{T}(x_2, z)$ and
$\alpha_2 := \mathtt{T}(x_1, x_2),\ \mathtt{T}(x_1, x_3),\ \mathtt{T}(x_3, x_1) \rightarrow \mathtt{R}(x_2)$.

We can observe that $\alpha_1 \prec_c \alpha_2$ because the instance $\{\mathtt{S}(c,d),\ \mathtt{T}(c,d),\ \mathtt{T}(d,c)\}$ together with the tuples $\overline{a} = c,d$ and $\overline{b} = d,n_1,c$ satisfy the requirements in Definition 27. Note that n_1 is the null value which is created by the oblivious chase step $\{\mathtt{S}(c,d),\ \mathtt{T}(c,d),\ \mathtt{T}(d,c)\} \overset{*,\alpha_1,\overline{a}}{\rightarrow} \{\mathtt{S}(c,d),\ \mathtt{T}(c,d),\ \mathtt{T}(d,c),\ \mathtt{T}(d,n_1)\}$.

Example 29. (see [Deutsch et al., 2008]) Let predicate E store the edge relation of a graph and let the constraint

$$\alpha := \mathtt{E}(x_1,x_2),\ \mathtt{E}(x_2,x_1) \rightarrow \exists y_1,y_2\ \mathtt{E}(x_1,y_1),\ \mathtt{E}(y_1,y_2),\ \mathtt{E}(y_2,x_1)$$

be given, stating that each node having a cycle of length 2 also has a cycle of length 3. A 3-cycle cannot be homomorphically mapped into 2-cycle again, so it holds that $\alpha \not\prec_c \alpha$.

The actual definition of c-stratification then relies, as outlined before, on the notion of weak acyclicity.

Definition 30. (see [Deutsch et al., 2008]) The c-chase graph $G_c(\Sigma) = (\Sigma, E)$ of a set of constraints Σ contains a directed edge (α, β) between two constraints iff $\alpha \prec_c \beta$. We call Σ c-stratified iff the constraints in every cycle of $G_c(\Sigma)$ are weakly acyclic.

Example 31. Given the set of TGDs $\Sigma = \{\alpha_1, ..., \alpha_4\}$, where

$$\begin{array}{lcl}
\alpha_1 & := & \mathtt{R}(x_1) \rightarrow \mathtt{S}(x_1,x_1),\\
\alpha_2 & := & \mathtt{S}(x_1,x_2) \rightarrow \exists z\ \mathtt{T}(x_2,z),\\
\alpha_3 & := & \mathtt{S}(x_1,x_2) \rightarrow \mathtt{T}(x_1,x_2),\ \mathtt{T}(x_2,x_1) \text{ and}\\
\alpha_4 & := & \mathtt{T}(x_1,x_2),\ \mathtt{T}(x_1,x_3),\ \mathtt{T}(x_3,x_1) \rightarrow \mathtt{R}(x_2).
\end{array}$$

We will give now an instance for which the chase does not necessarily terminate. Consider the database $\{\mathtt{R}(a)\}$ and the chase sequence which applies the constraints in the order $\alpha_1, ..., \alpha_4, \alpha_1, ..., \alpha_4, ...$ and so on. The first steps of the resulting chase sequence look as follows:

$$\begin{array}{ll}
 & \{\mathtt{R}(a)\}\\
\xrightarrow{\alpha_1,a} & \{\mathtt{R}(a),\ \mathtt{S}(a,a)\}\\
\xrightarrow{\alpha_2,a,a} & \{\mathtt{R}(a),\ \mathtt{S}(a,a),\ \mathtt{T}(a,n_1)\}\\
\xrightarrow{\alpha_3,a,a} & \{\mathtt{R}(a),\ \mathtt{S}(a,a),\ \mathtt{T}(a,n_1),\ \mathtt{T}(a,a)\}\\
\xrightarrow{\alpha_4,a,n_1,a} & \{\mathtt{R}(a),\ \mathtt{S}(a,a),\ \mathtt{T}(a,n_1),\ \mathtt{T}(a,a),\ \mathtt{R}(n_1)\}\\
\xrightarrow{\alpha_1,n_1} & \{\mathtt{R}(a),\ \mathtt{S}(a,a),\ \mathtt{T}(a,n_1),\ \mathtt{T}(a,a),\ \mathtt{R}(n_1),\ \mathtt{S}(n_1,n_1)\}
\end{array}$$

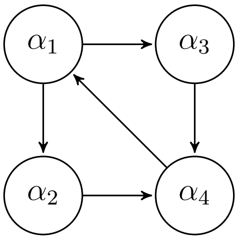


Figure 6.2: C-chase graph for Example 31.

$\xrightarrow{\alpha_2,n_1,n_1}$ $\{\mathtt{R}(a),\ \mathtt{S}(a,a),\ \mathtt{T}(a,n_1),\ \mathtt{T}(a,a),\ \mathtt{R}(n_1),\ \mathtt{S}(n_1,n_1),\ \mathtt{T}(n_1,n_2)\}$

$\xrightarrow{\alpha_3,n_1,n_1}$ $\{\mathtt{R}(a),\ \mathtt{S}(a,a),\ \mathtt{T}(a,n_1),\ \mathtt{T}(a,a),\ \mathtt{R}(n_1),\ \mathtt{S}(n_1,n_1),\ \mathtt{T}(n_1,n_2),\ \mathtt{T}(n_1,n_1)\}$

$\xrightarrow{\alpha_4,n_1,n_2,n_1}$ $\{\mathtt{R}(a),\ \mathtt{S}(a,a),\ \mathtt{T}(a,n_1),\ \mathtt{T}(a,a),\ \mathtt{R}(n_1),\ \mathtt{S}(n_1,n_1),\ \mathtt{T}(n_1,n_2),\ \mathtt{T}(n_1,n_1),$
$\mathtt{R}(n_2)\},$

$\xrightarrow{\alpha_1,n_2}$ $\ldots$

where n_1, n_2 are fresh null values. It can be easily seen that this sequence is infinite. The c-chase graph for this constraint set is depicted in Figure 6.2. The only strongly connected component in it is Σ itself, which is not weakly acyclic. So, Σ is not c-stratified, as witnessed by the non-terminating chase sequence above. In order to illustrate why $\alpha_2 \prec_c \alpha_4$ holds, we set $I_0 = \{S(a,b), T(a,b), T(b,a)\}$, $\bar{a} = a, b$ and $\bar{b} = b, n_1, a$. Then, we can apply an oblivious chase step to obtain $I_1 = I_0 \cup \{T(b, n_1)\}$ and see that $I_1 \not\models \alpha_4(\bar{b})$.

From the definition of $\prec_c$ it is not immediately clear that it is decidable, however the test for membership in $\prec_c$ can be done with linearly-sized databases.

Proposition 32. It can be decided in coNP whether a set of constraints is c-stratified.

Proof Sketch. We start with an additional claim: let α, β be constraints. Then, the mapping $(\alpha, \beta) \mapsto \alpha \prec_c \beta$? can be computed by an NP-algorithm. The proof of this claim proceeds like the proof of Theorem 3 in [Deutsch et al., 2008]. It is enough to consider candidate databases for I of size at most $|\alpha| + |\beta|$, i.e. unions of homomorphic images of the premises of α's and β's bodies. To prove that Σ is not c-stratified, guess some strongly connected component Σ' of the c-chase graph and verify that it is not weakly acyclic. $\square$

C-stratification ensures the termination of the chase in the sense of $\mathrm{CT}_{\forall\forall}$, i.e. independently of the data and the chase sequence, as the following theorem states.

Theorem 33. Let Σ be a fixed set of c-stratified constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for any database instance I, the length of every chase sequence is bounded by $Q(|dom(I)|)$. Thus, $\Sigma \in \mathrm{CT}_{\forall\forall}$.

To prepare the proof of this theorem, we will show two lemmas.

Lemma 34. Let α and β be two constraints. If there exists $1 \leq r \in \mathbb{N}$, finite databases $I_1, ..., I_{r+1}$, and $(\alpha_1, \overline{a}_1), ..., (\alpha_r, \overline{a}_r)$ such that $I_1 \overset{\alpha_1,\overline{a}_1}{\longrightarrow} ... \overset{\alpha_r,\overline{a}_r}{\longrightarrow} I_{r+1}$ and $(I_2 \backslash I_1) \cap body(\alpha_r)(\overline{a}_r) \neq \emptyset$, then there is a directed path from α_1 to α_r in $G_c(\Sigma)$.

Proof Sketch. We introduce some additional notation needed for this proof. We say that every atom $t \in I_1$ has rank zero, i.e. $rank(t) = 0$. For every atom $t \in I_2 \backslash I_1$ we set $rank(t) = 1$. For $i \in [r]$ every atom in $t \in I_{i+1} \backslash I_i$ we set $rank(t) = 1 + max\{rank(t') \mid t' \in body(\alpha_i)(\overline{a}_i)\}$ if there is some $t' \in body(\alpha_i)(\overline{a}_i)$ such that $rank(t') > 0$. Otherwise, we set $rank(t) = 0$. For a set of atoms A we set $rank(A) = max\{rank(t) \mid t \in A\}$.
If $rank(body(\alpha_r)(\overline{a}_r)) = 1$, then the claim of this lemma follows immediately from the chase sequence from the prerequisites. We set $I := body(\alpha_1)(\overline{a}_1) \cup body(\alpha_r)(\overline{a}_r) \backslash (I_2 \backslash I_1)$, $\overline{a} = \overline{a}_1$, $\overline{b} = \overline{a}_r$ in order ro satisfy the conditions of Definition 27.
Otherwise $rank(body(\alpha_r)(\overline{a}_r)) > 1$. Let $t \in body(\alpha_r)(\overline{a}_r)$ of maximal rank. Say, t was created by the chase step $(\alpha_i, \overline{a}_i)$. Then, $I := body(\alpha_i)(\overline{a}_i) \cup body(\alpha_r)(\overline{a}_r) \backslash (I_{i+1} \backslash I_i)$ together with $\overline{a} = \overline{a}_i$, $\overline{b} = \overline{a}_r$ satisfies the conditions of Definition 27. Thus, $\alpha_i \prec_c \alpha_r$. We have that $rank(body(\alpha_i)(\overline{a}_i)) < rank(body(\alpha_r)(\overline{a}_r))$. By induction hypothesis we can conclude that there is a directed path from α_1 to α_i in $G_c(\Sigma)$, which finishes this proof. ☐

Intuitively, this lemma can be understood as follows: if α is able to create some atom(s) that can be “plugged” into the body of β, then there is a path from α to β in $G_c(\Sigma)$. This does not hold for the case of stratification.

Lemma 35. Let Σ be a set of constraints and $C_1, .., C_n$ all strongly connected components of $G_c(\Sigma)$. If for all $i \in [n]$ it holds that $C_i \in \mathrm{CT}_{\forall\forall}$, then $\Sigma \in \mathrm{CT}_{\forall\forall}$.

Proof Sketch. Assume that we have a database instance I_0 such that the chase does not terminate. We will construct an infinite chase sequence that uses only constraints from some of the C_i.

We have an infinite chase sequence $S = I_0 \xrightarrow{\alpha_1,\overline{a}_1} I_1 \xrightarrow{\alpha_2,\overline{a}_2} \ldots$.
We construct a "partial c-chase graph" (Σ, E_S) as follows. For every $i, j \in \mathbb{N}$ such that $I_{i-1} \xrightarrow{\alpha_i,\overline{a}_i} I_i \xrightarrow{\alpha_{i+1},\overline{a}_{i+1}} \ldots \xrightarrow{\alpha_{j-1},\overline{a}_{j-1}} I_{j-1} \xrightarrow{\alpha_j,\overline{a}_j} I_j$ and $(I_i \backslash I_{i-1}) \cap \mathit{body}(\alpha_j)(\overline{a}_j) \neq \emptyset$, we add the edges to E_S according to the previous lemma. Let $G_c(\Sigma) = (\Sigma, E)$ the c-chase graph for Σ. It follows from Lemma 34 that $E_S \subseteq E$.
Without loss of generality, we can assume that every constraint from Σ fires infinitely often and that for every $j \in \mathbb{N}$ there is some $i > j$ such that $I'_{i-1} \models \alpha_i(\overline{a}_i)$, where $I'_0 := I_0$, $I_{l-1} \overset{\alpha_l,\overline{a}_l}{\to} J_l$ for $l \neq j$ and $J_j := J_{j-1}$.
Therefore, for every $i \in \mathbb{N}$ there exists $r \in \mathbb{N}_0$ such that $I_{i-1} \xrightarrow{\alpha_i,\overline{a}_i} I_i \xrightarrow{\alpha_{i+1},\overline{a}_{i+1}} \ldots \xrightarrow{\alpha_{r+i-3},\overline{a}_{r+i-3}} I_{r+i-3} \xrightarrow{\alpha_{r+i-2},\overline{a}_{r+i-2}} I_{r+i-1}$ and $(I_i \backslash I_{i-1}) \cap \mathit{body}(\alpha_{r+i-2})(\overline{a}_{r+i-2}) \neq \emptyset$, so every constraint is contained in some cycle in E_S. If we have only one strongly connected component in E_S, we are done. Otherwise, we choose a strongly connected component C_{i_0} such that every $\alpha \in C_{i_0}$ has predecessors only in C_{i_0}. It follows that there is a non-terminating chase sequence for the instance I_0 and the constraint set C_{i_0}. □

We now turn toward the proof of the theorem.

Proof of Theorem 33. Let Σ be the set of constraints under consideration. Let $C_1, ..., C_n$ be the strongly connected components of $G_c(\Sigma)$. The proof of the theorem's termination part follows by application of Lemma 35.
The polynomial time data complexity is derived as follows. First, note that by assumption, C_i is weakly acyclic and therefore terminates terminates in time $Q_i(|\mathit{dom}(I)|)$ for some $Q_i \in \mathbb{N}[X]$. It holds that every constraint outside $\bigcup_{j \in [n]} C_j$ can fire at most polynomially many times from which the rest of the claim follows. □

6.2 Importance of Null Positions

Before going on, we want to shortly discuss the main idea of all following termination conditions in this chapter. As we have already seen, the reason for infinite chase sequences is an infinite cascading of labeled nulls (see Subsection 3.6.1). Therefore our idea is as follows. Given a constraint violation

$$I \not\models \alpha(\overline{a})$$

during the application of the chase algorithm, we try to estimate what values in $\overline{a}$ are null values that were newly introduced by the chase algorithm and have not been in the input instance.

With this idea at hand we are not only able to improve the weak acyclicity condition but also the method that lifts weak acyclicity to c-stratification, finally leading to the introduction of the ∀∀-T-hierarchy.

6.3 Safe Constraints

This section introduces the class of so-called *safe* constraints. We will show that this class guarantees termination of the chase and that it extends the notion of weak acyclicity but does not coincide with stratification as depicted in Figure 1.2. It will be later our main tool to define more powerful sufficient termination conditions for the chase.

The basic idea of our first new termination condition safety is to estimate the set of positions where labeled nulls may be copied to and statically analyze the data flow only between these positions. As a useful tool, we borrow the notion of so-called *affected positions* from [Calì et al., 2008], which is an overestimation of the positions in which a null value that was introduced during the chase may occur.

Definition 36. (see [Calì et al., 2008]) Let Σ be a set of TGDs over $\mathcal{R}$. The set of affected positions $\mathrm{aff}(\Sigma)$ of Σ is inductively defined as follows. Let π be a position in the head of a TGD $\sigma \in \Sigma$.

- If an existentially quantified variable appears in π, then $\pi \in \mathrm{aff}(\Sigma)$.
- If the same universally quantified variable x appears both in position π and in the body of σ in affected positions only, then $\pi \in \mathrm{aff}(\Sigma)$.

Although we borrow this definition from [Calì et al., 2008], our focus is different. We use affected positions to extend known classes of constraints for which the chase terminates, whereas [Calì et al., 2008] investigates query answering in cases the chase may not terminate. Our work neither subsumes [Calì et al., 2008] nor the other way around.

We motivate the safety termination condition using the single constraint

$$\beta := \mathtt{R}(x_1, x_2, x_3),\ \mathtt{S}(x_2) \rightarrow \exists y\ \mathtt{R}(x_2, y, x_1).$$

The dependency graph of constraint set $\{\beta\}$ is shown in Figure 6.3 (left). As can be seen, there is a cycle going through a special edge, so the set is not weakly acyclic. We next study the affected positions in β:

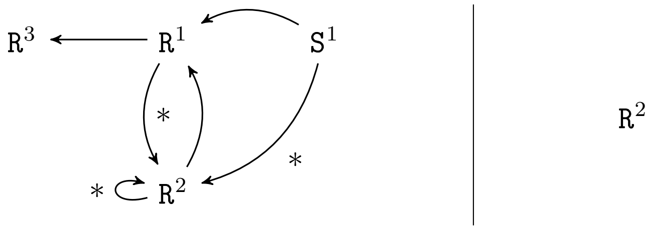


Figure 6.3: On the left hand side is the dependency graph and on the right hand side the propagation graph (it has no edges).

Example 37. Let us consider the constraint set $\Sigma := \{\beta\}$. Clearly, position $\texttt{R}^2$ is affected because it contains an existentially quantified variable. $\texttt{S}^1$ is not affected because $\texttt{S}$ does not occur in the head of β. Finally, we observe that also $\texttt{R}^1$ is not affected because x_2 occurs not only in $\texttt{R}^2$ but also in $\texttt{S}^1$, which is not an affected position. It follows that $\texttt{R}^3$ cannot be affected, too. We conclude that position $\texttt{R}^2$ is the only affected position in constraint set Σ.

We now argue that for constraint β a cascading of labeled nulls cannot occur, i.e. no fresh labeled null can repeatedly create new labeled nulls in position $\texttt{R}^2$ while copying itself to position $\texttt{R}^1$. The reason is that β cannot be violated with a fresh labeled null in $\texttt{R}^2$, i.e. if $\texttt{R}(a_1, a_2, a_3)$ and $\texttt{S}(a_2)$ hold, but $\exists y\ \texttt{R}(a_2, y, a_1)$ does not, then a_2 is never a newly created labeled null. This is due to the fact that a_2 also occurs in $\texttt{S}^1$, but $\texttt{S}^1$ is not an affected position. Hence, the chase sequence always terminates. We will later see that this is not a mere coincidence: the constraint is safe.

Like in the case of weak acyclicity, we define the safety condition with the help of the absence of cycles containing special edges in some graph. In order to distinguish it from the dependency graph, we call this graph the *propagation graph*.

Definition 38. Given a set of constraints Σ, the propagation graph $\text{prop}(\Sigma) := (\text{aff}(\Sigma), E)$ is the directed graph defined as follows. There are two kinds of edges in E. Add them as follows: for every TGD $\forall \overline{x}(\phi(\overline{x}) \rightarrow \exists \overline{y} \psi(\overline{x}, \overline{y})) \in \Sigma$ and for every x in $\overline{x}$ that occurs in ψ and every occurrence of x in ϕ in position π_1

- if x occurs only in affected positions in ϕ then, for every occurrence of x in ψ in position π_2, add an edge $\pi_1 \rightarrow \pi_2$ (if it does not already exist).
- if x occurs only in affected positions in ϕ then, for every existentially quantified variable y and for every occurrence of y in a position π_2, add a *special* edge $\pi_1 \overset{*}{\rightarrow} \pi_2$ (if it does not already exist).

As an improvement over weak acyclicity, in the propagation graph we do not supervise the whole data flow but only the flow of labeled nulls that might be introduced at runtime. Consequently, the graph contains edges only for null values that stem exclusively from affected positions. We now can easily define the safety condition on top of the propagation graph.

Definition 39. A set Σ of constraints is called *safe* iff $\mathrm{prop}(\Sigma)$ has no cycles going through a special edge.

Note that given a set of constraints it can be decided in polynomial time whether it is safe.

The intuition of these definitions is that we forbid an unrestricted cascading of null values, i.e. with the help of the propagation graph we impose a partial order on the affected positions such that any newly introduced null value can only be created in a position that has a higher rank in that partial order in comparison to null values that may occur in the body of a TGD. To state this more precisely, assume a TGD of the form $\forall\overline{x}(\phi(\overline{x}) \rightarrow \exists\overline{y}\psi(\overline{x},\overline{y}))$ is violated. Then, $I \models \phi(\overline{a})$ must hold, but $I \not\models \exists\overline{y}\psi(\overline{a},\overline{y}))$. The safety condition ensures that any position in the body that has a newly created labeled null from $\overline{a}$ in itself and also occurs in the head of the TGD has a strictly lower rank in our partial order than any position in which some element from $\overline{y}$ occurs. The main difference in comparison to weak acyclicity is that we look in a refined way (see affected positions) on where a labeled null can be propagated to.

Note that if Σ is safe, then every subset of Σ is safe, too. We will now compare safety to other termination conditions. We start with weak acyclicity and consider an example.

Example 40. Consider the TGD $\mathtt{R}(x_1, x_2, x_3),\ \mathtt{S}(x_2) \rightarrow \exists y\ \mathtt{R}(x_2, y, x_1)$ from earlier. The dependency graph and the propagation graph are depicted in Figure 6.3. The only affected position is $\mathtt{R}^2$. From the respective definitions it follows that this constraint is safe, but not weakly acyclic.

In the last example, the propagation graph was a subgraph of the dependency graph. This turns out not to be a mere coincidence.

Proposition 41. Let Σ be a set of constraints. Then, $\mathrm{prop}(\Sigma) \subseteq \mathrm{dep}(\Sigma)$. Therefore, it holds that if Σ is weakly acyclic, then it is also safe.

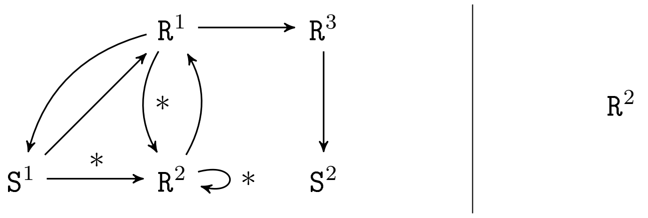


Figure 6.4: On the left hand side is the dependency graph and on the right hand side the propagation graph (it has no edges).

The proposition follows directly from the definitions of safety and weak acyclicity. In the propagation graph stronger conditions have to be satisfied than in the dependency graph in order to add special or non-special edges.

We give two more examples for safety, respectively non-safety. They show that neither c-stratification implies safety nor the other way round.

Example 42. Let the TGDs

$\alpha := \mathtt{S}(x_2, x_3), \mathtt{R}(x_1, x_2, x_3) \to \exists y\ \mathtt{R}(x_2, y, x_1)$ and
$\beta := \mathtt{R}(x_1, x_2, x_3) \to \mathtt{S}(x_1, x_3)$

be given. It can easily be shown that $\alpha \prec_c \beta$ and $\beta \prec_c \alpha$. Together with the fact that $\{\alpha, \beta\}$ is not weakly acyclic (cf. Figure 6.4) it follows that $\{\alpha, \beta\}$ is not c-stratified. However, $\{\alpha, \beta\}$ is safe (cf. Figure 6.4).

Example 43. (see Example 29) Let the constraint

$$\gamma := \mathtt{E}(x_1, x_2), \mathtt{E}(x_2, x_1) \to \exists\ y_1, y_2\ \mathtt{E}(x_1, y_1), \mathtt{E}(y_1, y_2), \mathtt{E}(y_2, x_1)$$

be given. It was argued earlier that $\{\gamma\}$ is c-stratified. However, it is not safe because both $\mathtt{E}^1$ and $\mathtt{E}^2$ are affected and therefore $\text{dep}(\{\gamma\}) = \text{prop}(\{\gamma\})$. It was seen in [Deutsch et al., 2008] that it is not weakly acyclic.

The next result shows that safety guarantees termination in the sense of $\text{CT}_{\forall\forall}$ while retaining the property of polynomial time data complexity.

Theorem 44. Let Σ be a fixed set of safe constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for every database instance I, the length of every chase sequence is bounded by $Q(|dom(I)|)$. Thus, $\Sigma \in \text{CT}_{\forall\forall}$.

Proof. First we introduce some additional notation. We denote constraints in the form $\phi(\overline{x_1}, \overline{x_2}, \overline{u}) \rightarrow \exists \overline{y} \psi(\overline{x_1}, \overline{x_2}, \overline{y})$, where $\overline{x_1}, \overline{x_2}, \overline{u}$ are all the universally quantified variables and

- $\overline{u}$ are those variables that do not occur in the head,
- every element in $\overline{x_1}$ occurs in a non-affected position in the body, and
- every element in $\overline{x_2}$ occurs only in affected positions in the body.

The proof is inspired by the proof of Theorem 3.9 in [Fagin et al., 2005], especially the notation and some introductory definitions are taken from there. In a first step we will give the proof for TGDs only, i.e. we do not consider EGDs. Later, we will see what changes when we add EGDs.

Note that Σ is fixed. Let (V, E) be the propagation graph prop(Σ). For every position $\pi \in V$ an incoming path is a, possibly infinite, path ending in π. We denote by $rank(\pi)$ the maximum number of special edges over all incoming paths. It holds that $rank(\pi) < \infty$ because prop(Σ) contains no cycles through a special edge. Define $r := max\{\ rank(\pi) \mid \pi \in V\ \}$ and $p := |V|$. It is easily verified that $r \leq p$, thus r is bounded by a constant. This allows us to partition the positions into sets $N_0, ..., N_p$ such that N_i contains exactly those positions π with $rank(\pi) = i$. Let n be the number of values in I. We define $dom(\Sigma)$ as the set of constants in Σ.

Choose some $\alpha := \phi(\overline{x_1}, \overline{x_2}, \overline{u}) \rightarrow \exists \overline{y} \psi(\overline{x_1}, \overline{x_2}, \overline{y}) \in \Sigma$. Let $I \rightarrow \ldots \rightarrow \overline{G} \xrightarrow{\alpha, \overline{a_1}\overline{a_2}\overline{b}} G'$ and let $\overline{c}$ be the newly created null values in the step from $\overline{G}$ to G'. Then

1. newly introduced labeled nulls occur only in affected positions,
2. $\overline{a_1} \subseteq dom(I) \cup dom(\Sigma)$ and
3. for every labeled null $Y \in \overline{a_2}$ that occurs in π in ϕ and every $c \in \overline{c}$ that occurs in some position ρ in ψ it holds that $rank(\pi) < rank(\rho)$.

This intermediate claim is easily proved by induction on the length of the chase sequence. Now we show by induction on i that the number of values that can occur in any position in N_i in G' is bounded by some polynomial Q_i in n that depends only on i (and, of course, Σ). As $i \leq r \leq p$, this implies the theorem's statement because the maximal arity $ar(\mathcal{R})$ of a relation is fixed. We denote by $body(\Sigma)$ the number of characters of the largest body of all constraints in Σ.

Case 1: $i = 0$. We claim that $Q_0(n) := n + |\Sigma| \cdot n^{ar(\mathcal{R}) \cdot body(\Sigma)}$ is sufficient for our needs. We consider a position $\pi \in N_0$ and an arbitrary TGD from Σ such that π occurs in the head of α. For simplicity we assume that it has the syntactic form of α. In case that there is a universally quantified variable in π, there can occur at most n distinct elements in π. Therefore, we assume that some existentially quantified variable occurs in π in ψ. Note that as $i = 0$ it must hold that $|x_2| = 0$. Every value in I can occur in π. But how many labeled nulls can be newly created in π? For every choice of $\overline{a_1} \subseteq dom(\overline{G})$ such that $\overline{G} \models \phi(\overline{a_1}, \lambda, \overline{b})$ and $\overline{G} \not\models \exists \overline{y} \psi(\overline{a_1}, \lambda, \overline{y})$ at most one labeled null can be added to π by α. Note that in this case it holds that $\overline{a_1} \subseteq dom(I)$ due to (1). So, there are at most $n^{ar(\mathcal{R}) \cdot body(\Sigma)}$ such choices. Over all TGDs at most $|\Sigma| \cdot n^{ar(\mathcal{R}) \cdot body(\Sigma)}$ labeled nulls are created in π.

Case 2: $i \to i+1$. We claim that

$$Q_{i+1}(n) := \sum_{j=0}^{i} Q_i(n) + |\Sigma| \cdot \left(\sum_{j=0}^{i} Q_i(n)\right)^{ar(\mathcal{R}) \cdot body(\Sigma)}$$

is such a polynomial. Consider the fixed TGD α. Let $\pi \in N_{i+1}$. Values in π may be either copied from a position in $N_0 \cup ... \cup N_i$ or may be a new labeled null. Therefore w.l.o.g. we assume that some existentially quantified variable occurs in π in ψ. In case a TGD, say α, is violated in G' there must exist $\overline{a_1}, \overline{a_2} \subseteq dom_{G'}(N_0, ..., N_i)$ and $\overline{b} \subseteq dom(G')$ such that $G' \models \phi(\overline{a_1}, \overline{a_2}, \overline{b})$, but $G' \not\models \exists \overline{y} \psi(\overline{a_1}, \overline{a_2}, \overline{y})$. If newly introduced labeled null occur in $\overline{a_2}$, say in some position ρ, then $\rho \in \bigcup_{j=0}^{i} N_j$. As there are at most $(\sum_{j=0}^{i} Q_i(n))^{ar(\mathcal{R}) \cdot body(\Sigma)}$ many such choices for $\overline{a_1}, \overline{a_2}$, at most $(\sum_{j=0}^{i} Q_i(n))^{ar(\mathcal{R}) \cdot body(\Sigma)}$ many labeled nulls can be newly created in π.

When we allow EGDs among our constraints, we have that the number of values that can occur or be created in any position in N_i in G' can be bounded by the same polynomial Q_i. The difference to the situation of TGDs only is that Q_i bounds the number of different values in a position over all instances in our chase sequence, i.e. $|\{a \in \Delta \cup \Delta_{null} | a$ appears in some position of N_i of some database instance after some finite number of chase steps$\}| \leq Q_i(n)$. The reason for this to work is that equating labeled nulls does not increase the number of labeled nulls and the use of the same partial order on the database positions like in the case of TGDs only. □

6.4 Safely Restricted Constraints

In this section we generalize the method of c-stratification to a condition which we call *safe restriction.* The c-chase graph from Section 6.1 will be a special case of our new notion. We show that the chase always terminates in the sense of $\mathrm{CT}_{\forall\forall}$ for constraints obeying this new condition. We begin with a small example.

Example 45. Let the constraints

$$\alpha := \mathtt{S}(x_2, x_3), \mathtt{R}(x_1, x_2, x_3) \rightarrow \exists y\, \mathtt{R}(x_2, y, x_1) \text{ and}$$
$$\beta := \mathtt{R}(x_1, x_2, x_3) \rightarrow \mathtt{S}(x_1, x_3)$$

from Example 42 be given again. We have seen that this constraint set is safe and therefore $\{\alpha, \beta\} \in \mathrm{CT}_{\forall\forall}$. Still, we will make an observation that will be useful for the development of further chase termination conditions. Intuitively, the chase will always terminate with these two constraints because a firing of α may cause a null value to appear in position $\mathtt{R}^2$, but a firing of β will never introduce null values in the head of α although $\beta \prec_c \alpha$ holds. This is the key motivation for the upcoming definitions.

First, we will refine the relation $\prec_c$, which will help us to detect if during the execution of the chase null values might be copied to the head of some constraint.

Let $pos(\Sigma)$ denote the set of positions that occur in the body of some constraint in Σ. In order to simplify the definition of $\prec_c$'s refinement, we introduce the notion of null-pos:

Definition 46. Let Σ be a set of constraints, I be a fixed database instance and $N \subseteq \Delta_{null}$. Then, we define null-pos(N, I) as $\{\pi \in pos(\Sigma) \mid a \in N, a \text{ occurs in position } \pi \text{ in } I\}$.

Informally spoken, null-pos(A, I) is the set of positions in I that also appear in Σ and in which elements (i.e. labeled nulls) from A occur. We are now ready to define the refinement of relation $\prec_c$:

Definition 47. Let Σ be a set of constraints and $P \subseteq pos(\Sigma)$. For all $\alpha, \beta \in \Sigma$, we define $\alpha \prec_P \beta$ iff there are tuples $\bar{a}, \bar{b}$ and a database instance I_0 such that

1. $I_0 \overset{*,\alpha,\bar{a}}{\rightarrow} I_1$,
2. $I_1 \not\models \beta(\bar{b})$,
3. $I_0 \models \beta(\bar{b})$, and
4. there is $n \in \bar{b} \cap \Delta_{null}$ in the head of $\beta(\bar{b})$ such that null-pos$(\{n\}, I_0) \subseteq P$.

The main differences to $\prec_c$ is that we check if a “possibly dangerous null value” is copied via the last bullet. We next introduce a notion for affected positions relative to a constraint and a set of positions.

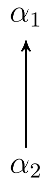


Figure 6.5: Graph of the 2-restriction system for Example 51.

Definition 48. For any set of positions P and a TGD α let aff-cl(α, P) be the set of positions π from the head of α such that

- for every universally quantified variable x in π: x occurs in the body of α only in positions from P or
- π contains an existentially quantified variable.

The latter definition and the refinement of $\prec_c$ will help us to define the notion of a restriction system, which is a strict generalization of the c-chase graph.

Definition 49. A 2-*restriction system* is a pair $(G'(\Sigma), f)$, where $G'(\Sigma) := (\Sigma, E)$ is a directed graph and $f \subseteq pos(\Sigma)$ such that

1. for all TGDs α and for all $(\alpha, \beta) \in E$: aff-cl$(\alpha, f) \cap pos(\Sigma) \subseteq f$,
2. for all TGDs β and for all $(\alpha, \beta) \in E$: aff-cl$(\beta, f) \cap pos(\Sigma) \subseteq f$, and
3. for all $\alpha, \beta \in \Sigma$: $\alpha \prec_f \beta \implies (\alpha, \beta) \in E$.

A 2-restriction system is *minimal* if it is obtained from $((\Sigma, \emptyset), \emptyset)$ by a repeated application of the constraints from bullets one and two (until all constraints hold) s.t., in case of the first bullet, f is extended only by those positions that are required to satisfy the condition.

We illustrate this definition by two examples. The first one also shows that 2-restriction systems always exist.

Example 50. Let Σ a set of constraints. Then, $((\Sigma, \Sigma \times \Sigma), f)$, where $f := pos(\Sigma)$ is a 2-restriction system for Σ.

Example 51. Let predicate E(x,y) store graph edges and predicate S(x) store some nodes. The constraints $\Sigma = \{\alpha_1, \alpha_2\}$ with

$\alpha_1 :=$S(x), E(x,y) $\rightarrow$ E(y,x) and
$\alpha_2 :=$S(x), E(x,y) $\rightarrow \exists z$ E(y,z), E(z,x)

assert that all nodes in S have a cycle of length 1 and 2. It holds that aff(Σ) = $\{\mathtt{E}^1,\mathtt{E}^2\}$ and it is easy to verify that Σ is neither safe nor c-stratified.
We will now illustrate that the minimal restriction system for Σ is G'(Σ):=(Σ,{(α_2,α_1)},f) with $f := \{\mathtt{E}^1,\mathtt{E}^2\}$. The 2-restriction system's graph is depicted in Figure 6.5.
Starting from $((\Sigma, \emptyset), \emptyset)$, we check whether $\alpha_2 \prec_\emptyset \alpha_1$ holds according to Definition 47. A possible start instance I_0 can be chosen as {S(c),S(d),E(c,d)}, $\overline{a} := c, d$ and $\overline{b} := d, n_1$, where n_1 is a fresh null value and $c, d \in \Delta$. We observe that $I_0 \xrightarrow{\alpha_2,c,d} I_1$, where $I_1 = I_0 \cup \{\mathtt{E}(d,n_1),\mathtt{E}(n_1,c)\}$. Thus, we have that $I_0 \xrightarrow{*,\alpha_2,c,d} I_1$. We see that $I_1 \not\models \alpha_1(\overline{b})$ and $I_0 \models \alpha_1(\overline{b})$. We can conclude that $\alpha_2 \prec_\emptyset \alpha_1$ holds. According to bullet three in Definition 49 we obtain as an intermediate result in the computation of the minimal 2-restriction system $((\Sigma, \{(\alpha_2, \alpha_1)\}), \emptyset)$. Applying bullets one and two, we result in $((\Sigma, \{(\alpha_2, \alpha_1)\}), f)$.
For convenience we will call a null value that occurs in position S *harmless*. Note that S is unchanged by our constraints and therefore cannot contain any newly introduced null values.
We now argue that $\alpha_1 \not\prec_f \alpha_1$ holds. Assume that $\alpha_1 \prec_f \alpha_1$ holds and choose a starting instance I_0 and $\overline{a}$, $\overline{b}$ according to Definition 47. Say $\overline{a} = c, d$, where $c, d \in \Delta \cup \Delta_{null}$ may not be distinct. The relation S contains only harmless null values or elements from Δ. Thus, c is either in Δ or harmless. In either case, we see that $\overline{b} = d, c$ which implies that d is in Δ or harmless, too.
With the same argumentation we see that $\alpha_1 \not\prec_f \alpha_2$ holds. In order to prove that $\alpha_2 \not\prec_f \alpha_2$ holds, we observe that an oblivious chase step with α_2 completes a cycle through an outgoing edge of a node in S. Obviously, α_2 cannot obliviously fire again with the help of some tuples created in the step just before.
To see that $\alpha_2 \prec_f \alpha_1$ holds we can use the same argumentation as we used to prove that $\alpha_2 \prec_\emptyset \alpha_1$ is valid. But this does not change our partial 2-restriction system anymore.
We end up with the final minimal 2-restriction system $((\Sigma, \{(\alpha_2, \alpha_1)\}), f)$.

Based on the novel technical notion of 2-restriction systems we can easily define a new class of constraints.

Definition 52. Σ is called *safely restricted* if and only if there is a restriction system $(G'(\Sigma), f)$ for Σ such that every strongly connected component in $G'(\Sigma)$ is safe.

Example 53. Recall the constraint set from Example 51. Its minimal 2-restriction system does not contain any cycle. It follows from the previous definition that this constraint set is safely restricted.

The next theorem shows that safe restriction strictly extends the notion of c-stratification and safety but is different from super-weak acyclicity.

Theorem 54.

- If Σ is c-stratified or safe, then it is also safely restricted.
- There is some Σ that is safely restricted but neither safe nor c-stratified.
- There is some set of TGDs Σ that is safely restricted but not super-weakly acyclic.
- There is some set of TGDs Σ that is super-weakly acyclic but not safely restricted.

Proof.

- Bullets one and two follow from Theorem 59.
- For bullet three observe that the constraint set from Example 51 is not super-weakly acyclic because $\mathrm{In}(\alpha_2, y) \lhd \mathrm{Move}(\Sigma, \mathrm{Out}(\alpha_2, z))$ and therefore we have that $\hookrightarrow_\Sigma = \{(\alpha_2, \alpha_2)\}$.
- For bullet four consider the two constraints $\mathtt{A}(x) \rightarrow \exists y\, \mathtt{B}(x, y), \mathtt{B}(y, x), \mathtt{C}(y)$ and $\mathtt{B}(x, x), \mathtt{C}(y) \rightarrow \mathtt{A}(x), \mathtt{C}(y)$. We have already argued in Example 17 that these two TGDs are super-weakly acyclic. It is easy to see that they are not safely restricted because of the atom $\mathtt{C}(y)$ in the second constraint. $\square$

Definition 52 implies that safely restricted constraints can be recognized by a Σ_2^P-algorithm. However, with the help of a canonical restriction system, we can show that safe restriction can be decided, like c-stratification, in coNP.

Theorem 55. Given constraint set Σ it can be checked by a coNP-algorithm whether Σ is safely restricted.

Before we prove Theorem 55, we introduce an additional tool. In general, a set of constraints may have several 2-restriction systems, but as the next lemma shows, we can always restrict ourselves to minimal 2-restriction systems.

Lemma 56. Let Σ be a set of constraints, $(G'(\Sigma), f)$ a restriction system for Σ and $(G'_{min}(\Sigma), f_{min})$ a minimal one.

- Let P be a set of positions and α, β constraints. Then, the mapping $(P, \alpha, \beta) \mapsto \alpha \prec_P \beta$? can be computed by an NP-algorithm.
- The minimal restriction system for Σ is unique. It can be computed from Σ in non-deterministic polynomial time.
- It holds that Σ is safely restricted if and only if every strongly connected component in $G'_{min}(\Sigma)$ is safe.

Proof. The proof of part one of the lemma proceeds like the proof of Theorem 3 in [Deutsch et al., 2008]. It is enough to consider candidate databases for A of size at most $|\alpha| + |\beta|$, i.e. unions of homomorphic images of the premises of α and β s.t. null values occur only in positions from P. This concludes part one.
Uniqueness holds by definition. It can be computed via successive application of the constraints (note that f and E are changed in each step) in definition 49 by a Turing machine that guesses answers to the question $\alpha \prec_P \beta$?. As the mapping $(P, \alpha, \beta) \mapsto \alpha \prec_P \beta$? can be computed by an NP-algorithm and the fixedpoint is reached after polynomially many applications of the constraints from definition 49, this implies the second claim.
Concerning the third claim, observe that every strongly connected component in $G'_{min}(\Sigma)$ is contained in a single strongly connected component of any other restriction system. This implies the third claim. □

Proof of Theorem 55. By the previous lemma it suffices to check the conditions from Definition 52 only for the minimal restriction system. To decide whether Σ is not safely restricted, compute the minimal restriction system, guess a strongly connected component and check if it is not safe. Clearly, this can be done in non-deterministic polynomial time. □

The next theorem is the main contribution of this section. It states that the chase will always terminate in the sense of $\text{CT}_{\forall\forall}$ in polynomial time data complexity for safely restricted constraints. We refer the interested reader to Theorem 75 for a formal proof of this theorem.

Theorem 57. Let Σ be a fixed set of safely restricted constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for every database instance I, the length of every chase sequence is bounded by $Q(|dom(I)|)$. Thus, $\Sigma \in \mathrm{CT}_{\forall\forall}$.

Finally, we compare the c-chase graph to 2-restriction systems because the reader might wonder what happens if we substitute weak acyclicity with safety in the definition of c-stratification.

Definition 58. We call Σ *safely c-stratified* iff the constraints in every cycle of $G_c(\Sigma)$ are safe.

We obtain the following result, showing that with the help of restriction systems, we have strictly extended the method of the c-chase graph.

Theorem 59. Let Σ be a set of constraints.

- If Σ is weakly acyclic or safe, then it is safely c-stratified.
- If Σ is safely c-stratified, then it is safely restricted.
- There is some set of constraints that is safely restricted, but not safely c-stratified.

Proof of Theorem 59.

- Let Σ be weakly acyclic. Every cycle in $G_c(\Sigma)$ is safe, because Σ is safe and weak acyclicity implies safety. Let Σ be safe. Every cycle in $G_c(\Sigma)$ is safe, because Σ is.
- Follows from Example 50 and the following proposition. Its proof follows directly from the definition of $\prec_P$ and $\prec_c$ and is therefore omitted here.

  Proposition 60. Let $P \subseteq P' \subseteq pos(\Sigma)$. If $\alpha \prec_P \beta$, then $\alpha \prec_{P'} \beta$. It holds that if $\alpha \prec_P \beta$, then $\alpha \prec_c \beta$.

- Consider the following TGDs. $\Sigma := \{\alpha, \beta, \chi, \delta\}$.

$$\begin{aligned}
\alpha &:= \mathtt{R}_1(x_1,x_2) \to \exists y\ \mathtt{S}(x_1,x_2,y),\\
\beta &:= \mathtt{R}_1(x_1,x_2) \to \exists y\ \mathtt{T}(x_1,x_2,y),\\
\chi &:= \mathtt{S}(x_1,x_2,x_3),\ \mathtt{T}(x_4,x_5,x_6) \to \mathtt{T}(x_5,x_1,x_4), \text{ and}\\
\delta &:= \mathtt{S}(x_1,x_2,x_3),\ \mathtt{T}(x_4,x_5,x_3) \to \mathtt{T}(x_1,x_3,x_3), R_1(x_3,x_1), R_2(x_3,x_1).
\end{aligned}$$

It can be seen that $\alpha \prec_c \chi$, $\beta \prec_c \chi$, $\chi \prec_c \delta$, $\delta \prec_c \alpha$ and $\delta \prec_c \beta$ holds. Thus , there is a cycle in the chase graph that involves all constraints. Unfortunately, the constraint set is not safe. Therefore, it is also not safely c-stratified. The minimal restriction system is $((\Sigma, E), f)$, where $E = \emptyset$ and $f = \emptyset$. Obviously, every cycle in (Σ, E) is safe. Hence, Σ is safely restricted. □

Note that we used safety instead of safe c-stratification in the definition of safe restriction although safely c-stratified constraints are the provably larger class. This is due to the fact that safety is easily checkable and using c-stratification would not change the class of constraints. The next proposition clarifies this issue.

Proposition 61. Σ is safely restricted iff there is a restriction system $(G'(\Sigma), f)$ for Σ such that every strongly connected component in $G'(\Sigma)$ is safely c-stratified.

Proof of Proposition 61. Let $(G'(\Sigma), f)$ be a restriction system for Σ such that every strongly connected component in $G'(\Sigma)$ is safely c-stratified. Choose some strongly connected component C and two constraints $\alpha, \beta \in C$ such that $\alpha \prec_P \beta$ for some set of positions P. By Proposition 60, $\alpha \prec_c \beta$ holds. As C is safely c-stratified, this means that C must also be safe. So, every cycle in $G'(\Sigma)$ is also safe. □

6.5 Inductively Restricted Constraints

In this section we generalize the method that lifts safety to safety restriction with the help of a more sophisticated use of 2-restriction systems. We define a new sufficient termination condition called *inductive restriction*, whose main idea is to decompose a given constraint set into smaller subsets (in a more refined way than safe restriction). We then use the safety condition from before to check the termination of every subset and, whenever all subsets are safe, the termination for the full constraint set can be guaranteed. Ultimately, we show that inductive restriction (like all the classes discussed before) guarantees chase termination in the sense of $\mathrm{CT}_{\forall\forall}$ in polynomial-time data complexity. This section also lays the foundations for the $\forall\forall$-T-hierarchy (cf. Figure 1.2), which will be defined

```
part($\Sigma$: Set of TGDs and EGDs, $k$: not equal to 1) {
    compute the strongly connected components (as
    sets of constraints) $C_1, \ldots, C_n$ of the minimal
    $k$-restriction system of $\Sigma$;
    $D \leftarrow \emptyset$
    if (n == 1) then
        if ($C_1 \neq \Sigma$) then
            return part($C_1$,$k$);
        endif
        return {$\Sigma$};
    endif
    for i=1 to n do
        $D \leftarrow D \cup$ part($C_i$,$k$);
    endfor
    return $D$; }
```

Figure 6.6: Algorithm to compute subsets of Σ.

subsequently in Section 6.6. We motivate our study with a constraint set that is neither safe nor safely restricted.

Example 62. We extend the constraint set from Example 51 to $\Sigma' := \Sigma \cup \{\alpha_3\}$, where

$$\alpha_3 := \exists x, y\ \mathtt{S}(x),\ \mathtt{E}(x, y).$$

Then $G'(\Sigma'):=(\Sigma',\{(\alpha_1,\alpha_2),(\alpha_2,\alpha_1),(\alpha_3,\alpha_1),(\alpha_3,\alpha_2)\},f)$ with $f = \{\mathtt{E}^1,\mathtt{E}^2,\mathtt{S}^1\}$ is the minimal 2-restriction system. It contains the strongly connected component $\{\alpha_1,\alpha_2\}$. Note that Σ' is neither safe, nor stratified, nor safely restricted. Hence, using the sufficient termination conditions discussed so far no chase termination guarantees can be made for Σ'.

Intuitively, in the example above the constraint α_3 "infects" position $\mathtt{S}^1$ in the 2-restriction system. Still, null values cannot be repeatedly created in $\mathtt{S}^1$: α_3 fires at most once, so it does not affect chase termination. Our novel termination condition resolves such situations by recursively computing the minimal 2-restriction systems of the strongly connected components. We formalize this computation in Algorithm 1, called $part(\Sigma, k)$[2], and define the class of inductively restricted constraint sets by help of this algorithm.

---

[2] In this section we will always set $k = 2$. The case $k \neq 2$ will be treated later.

Definition 63. Let Σ be a set of constraints. We call Σ *inductively restricted* iff every $\Sigma' \in part(\Sigma, 2)$ is safe.

Compared to safe restriction, inductive restriction does not increase the complexity of the recognition problem:

Lemma 64. Let Σ be a set of constraints. The recognition problem for inductive restriction is in coNP.

Proof Sketch. We start with an additional claim: let P be a set of positions and α, β constraints. Then, the mapping $(P, \alpha, \beta) \mapsto \alpha \prec_P \beta?$ can be computed by an NP-algorithm. The proof of this claim proceeds like the proof of Theorem 3 in [Deutsch et al., 2008]. It is enough to consider candidate databases for I_0 of size at most $|\alpha| + |\beta|$, i.e. unions of homomorphic images of the premises of α and β s.t. null values occur only in positions from P. Because of this claim, the minimal 2-restriction system of a set of constraints can be computed by an NP-algorithm (only polynomially many steps must be performed to reach the fixedpoint). Computing $part(\Sigma, 2)$ can also be done in non-deterministic polynomial time. To prove that Σ is not inductively restricted, guess some $\Sigma' \in part(\Sigma, 2)$ and verify that it is not safe. ☐

We give an example for an inductively restricted constraint set, which – as argued in Example 62 – is neither safe nor safely restricted.

Example 65. Referring back to Example 62, we have seen that the minimal 2-restriction system of Σ' contains the only strongly connected component $\{\alpha_1,\alpha_2\}$, which by Example 62 is not safe. Therefore we compute the minimal 2-restriction system of $\{\alpha_1,\alpha_2\}$ and see that it does not contain a cycle. This argumentation proves that $part(\Sigma', 2) = \emptyset$, so we conclude that constraint set Σ' is inductively restricted.

As depicted in Figure 1.2, the inductive restriction condition generalizes safe restriction. The following proposition formally states this result and shows that the respective inclusion relationship is proper. Please note that, as we will show later, inductive restriction ensures chase termination independently of the database and the chase sequence, therefore it cannot extend stratification, see Example 86.

Proposition 66. The following claims hold.

- If Σ is safely restricted, then it is inductively restricted.
- There is some Σ that is stratified, but not inductively restricted.

- There is some Σ that is inductively restricted, but not safely restricted.
- There is some set of TGDs Σ that is inductively restricted but not super-weakly acyclic.
- There is some set of TGDs Σ that is super-weakly acyclic but not inductively restricted.

Proof Sketch. We start with bullet one. It follows from the fact that every strongly connected component of Σ's minimal 2-restriction system is safe and so every subset of it must be safe, too. Bullet two follows from Example 86. Bullet three is proved by the constraint set from Examples 62 and 65. Finally, bullets four and five can be proved like bullets three and four in Theorem 54. ☐

The next theorem gives the main result concerning inductive restriction, showing that it guarantees chase termination in the sense of $\mathrm{CT}_{\forall\forall}$ in polynomial time data complexity. We refer the interested reader to Theorem 75 for a formal proof of this theorem.

Theorem 67. Let Σ be a fixed set of inductively restricted constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for any database instance I, the length of every chase sequence is bounded by $Q(|dom(I)|)$. Thus, $\Sigma \in \mathrm{CT}_{\forall\forall}$.

We conclude with the remark that our motivating constraint set from Figure 6.1 is not inductively restricted: the constraint α can cause itself to fire, so its minimal 2-restriction system contains an edge from α to α, which forms a strongly connected component; further, α is not safe. To show that the chase with α terminates, we need weaker termination conditions than inductive restriction.

6.6 The $\forall\forall$-T-Hierarchy

This section introduces the $\forall\forall$-T-hierarchy, which is our main result regarding $\mathrm{CT}_{\forall\forall}$. Its lowest level, $\forall\forall$-T[2], corresponds to inductive restriction. Every level in the hierarchy is decidable and strictly contains all lower levels. As we shall see, also the constraint from Figure 6.1 is a member of some level in this hierarchy.
We start by defining the k-ary relation $\prec_{k,P}$ which is a generalization of $\prec_P$. The definition naturally extends the $\prec_P$ relation to a fixed number k of constraints.

Definition 68. *Let $k \geq 2$, Σ a set of constraints and $P \subseteq pos(\Sigma)$. For all $\alpha_1, ..., \alpha_k \in \Sigma$, we define $\prec_{k,P}$ $(\alpha_1, ..., \alpha_k)$ iff there are tuples $\bar{a}_1, ..., \bar{a}_k$ and a database instance I_0 such that*

- *for all $i \in [k-1]$ it holds that $I_{i-1} \overset{*,\alpha_i,\bar{a}_i}{\rightarrow} I_i$,*
- *$I_{k-1} \not\models \alpha_k(\bar{a}_k)$,*
- *there is $n \in \bar{a}_k \cap \Delta_{null}$ in the head of $\alpha_k(\bar{a}_k)$ such that null-pos$(\{n\}, I_0) \subseteq P$,*
- *$I_0 \models \alpha_k(\bar{a}_k)$, and*
- *for every $i \in [k-1]$ it holds that J_{k-1} is defined and $J_{k-1} \models \alpha_k(\bar{a}_k)$, where $J_0 := I_0$, $J_{l-1} \overset{*,\alpha_l,\bar{a}_l}{\rightarrow} J_l$ for $k > l \neq i$ and $J_i := J_{i-1}$.* ☐

Note that $\prec_{2,P}$ corresponds exactly to $\prec_P$ introduced in Definition 47. It can be shown that, for a fixed value of k, membership in this relation is decidable in NP:

Proposition 69. Let $k \geq 2$ be fixed. Then, there exists a NP-algorithm that decides for every set of positions P and every $\alpha_1, ..., \alpha_k \in \Sigma$ whether $\prec_{k,P}$ $(\alpha_1, ..., \alpha_k)$ holds.

Proof: Let $k \geq 2$ be fixed and $\alpha_1, ..., \alpha_k$ be TGDs and EGDs. Assume $\prec_{k,P}$ $(\alpha_1, ..., \alpha_k)$. Choose a database instance I_0 and sequences $\bar{a}_1, ..., \bar{a}_k$ such that the definition of $\prec_{k,P}$ $(\alpha_1, ..., \alpha_k)$ holds. For all $i \in [k-1]$ set $I_{i-1} \overset{*,\alpha_i,\bar{a}_i}{\rightarrow} I_i$ and $I_{k-1} \overset{*,\alpha_k,\bar{a}_k}{\rightarrow} I_k$. There is a sequence of homomorphisms $h_1, ..., h_k$ such that $h_i : body(\alpha_i) \rightarrow I_{i-1}$ for all $i \in [k]$. Let $J_0 \subseteq I_0$ be the minimal subinstance (with respect to set cardinality) such that for all $i \in [k]$ $J_{i-1} \overset{\alpha_i,\bar{a}_i}{\rightarrow} J_i$ and $J_i \subseteq I_i$. Then J_0 and $\bar{a}_1, ..., \bar{a}_k$ also satisfy the conditions from the definition of $\prec_{k,P}$ $(\alpha_1, ..., \alpha_k)$. Furthermore. it must hold that $|J_0| \leq \sum_{i \in [k]}(|body(\alpha_i)| + |head(\alpha_i)|) \leq \sum_{i \in [k]} |\alpha_i|$, where $|\alpha_i|$ denotes the length of the sequence of symbols of the formula α_i. So only finitely many candidate databases have to be examined, which completes the proof. ☐

We next use the relation $\prec_{k,P}$ to define k-restriction systems, which naturally generalize the 2-restriction systems defined over relation $\prec_P$ (cf. Definition 49).

Definition 70. Let $k \in \mathbb{N}_{>1}$. A k-restriction system $G'_k(\Sigma)$ is a pair (G', f), where $G' = (\Sigma, E)$ is a graph and $f \subseteq pos(\Sigma)$ such that

- for all TGDs α and for all $(\alpha, \beta) \in E$: aff-cl$(\alpha, f) \cap pos(\Sigma) \subseteq f$,
- for all TGDs β and for all $(\alpha, \beta) \in E$: aff-cl$(\beta, f) \cap pos(\Sigma) \subseteq f$, and
- for all $\alpha_1, ..., \alpha_k \in \Sigma$: $\prec_{k,f} (\alpha_1, ..., \alpha_k)$ then $(\alpha_1, \alpha_2), ..., (\alpha_{k-1}, \alpha_k) \in E$.

A k-restriction system is *minimal* if it is obtained from $((\Sigma, \emptyset), \emptyset)$ by a repeated application of the constraints from bullets one and two (until all constraints hold) such that, in case of the first or second bullet, f is extended only by those positions that are required to satisfy the condition. In case the third bullet is applied, E is extended.

Note that for $k = 2$ this definition corresponds exactly to the definition of 2-restriction systems used to define inductive restriction. Like 2-restriction systems, minimal k-restriction systems are unique and can be computed by a coNP-algorithm:

Proposition 71. Let $k \geq 2$ be fixed and Σ a set of constraints. The minimal k-restriction system for Σ is unique and can be computed by an NP-algorithm.

Proof. Uniqueness follows directly from the definition: the computation is monotone and bounded. The computation takes polynomially many steps and each step requires at most one guess to check whether $\prec_{k,f} (\alpha_1, ..., \alpha_k)$ holds. Clearly, this algorithm runs in non-deterministic polynomial time. ☐

We are now in the position to define the ∀∀-T-hierarchy:

Definition 72. Let $k \geq 2$ and Σ be a set of constraints. Then $\Sigma \in$ ∀∀-T$[k]$ iff there is $k' \in [k] \backslash \{1\}$ such that for every $\Sigma' \in part(\Sigma, k')$ it holds that Σ' is safe.

We call ∀∀-T$[k]$ the k-th level of the ∀∀-T-hierarchy. As a corollary from Proposition 71 we obtain that we can decide whether a set of constraints is in ∀∀-T$[k]$ by a coNP-algorithm. We next give an example for constraints in the ∀∀-T-hierarchy.

Example 73.
We set $\Sigma_{k+1} := \{\alpha_{k+1}\}$, where

$$\alpha_{k+1} := \ \mathtt{S}(x_{k+1}), \mathtt{R}_k(x_1, ..., x_{k+1}) \rightarrow \exists y\ \mathtt{R}_k(y, x_1, ..., x_k).$$

In order to prove that $\prec_{k+1,\emptyset} (\alpha, ..., \alpha)$ holds, observe that on the starting instance $I_0 = \{\ \mathtt{S}(a_1), ..., \mathtt{S}(a_{k+1}, \mathtt{R}_k(a_1, ..., a_{k+1})\}$ we can apply $k + 1$ (oblivious) chase steps

resulting in the instances $I_i = I_{i-1} \cup \{\mathtt{R}_k(n_i, a_1, ..., a_{k+1-i})\}$ for $i \in [k]$ and $I_{k+1} = I_k \cup \{\mathtt{R}_k(n_{k+1}, ..., n_1)\}$.
We also see that $\prec_{k+2,\emptyset} (\alpha, ..., \alpha)$ cannot hold because there cannot exist $k+2$ oblivious chase steps such that bullet five in Definition 68 is satisfied.
So the minimal $(k+2)$-restriction system does not contain any cycle, but the minimal $k+1$-restriction system does. Therefore $\Sigma_{k+1} \in$ ∀∀-T$[k+2]$. On the other hand, we observe that the constraint is not safe, so it is not contained in ∀∀-T$[k+1]$. Also note that the constraint in Figure 6.1 exactly corresponds to Σ_2, so it is contained in level ∀∀-T[3].

The following proposition relates the levels of the ∀∀-T-hierarchy to each other and to inductive restriction.

Proposition 74. Let $k \geq 2$.

- Σ is inductively restricted iff $\Sigma \in$ ∀∀-T[2]
- ∀∀-T$[k] \subseteq$ ∀∀-T$[k+1]$.
- There is some Σ such that $\Sigma \in$ ∀∀-T$[k+1] \setminus$∀∀-T$[k]$.
- There is some set of TGDs Σ that is in $T[k]$ but not super-weakly acyclic.

Proof Sketch.

- To prove bullet one, note that both definitions coincide exactly.
- Bullet two follows by definition.
- For bullet three we refer back to Example 73.
- It is easy to see that the constraint set in Example 73 is not super-weakly acyclic. □

It is unclear to us if super-weak acyclicity is different from the T-hierarchy, but as we have already mentioned several times before, we can easily define a decidable fragment of $\mathrm{CT}_{\forall\forall}$ that is a superset of super-weak acyclicity by allowing in the definition of our classes that every strongly connected component of the k-restriction system may be either safe or super-weakly acyclic. As super-weak acyclicity is not our main object of interest in this thesis, we will not explicitly do this.

The next result is our main contribution concerning data- and sequence-independent chase termination. It states that, for a fixed value of k, membership in ∀∀-T$[k]$ guarantees polynomial time data complexity for the chase.

Theorem 75. Let $k \geq 2$ and $\Sigma \in \forall\forall\text{-T}[k]$ be a fixed set of constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for any database instance I, the length of every chase sequence is bounded by $Q(|dom(I)|)$. Thus, $\forall\forall\text{-T}[k] \subseteq \text{CT}_{\forall\forall}$.

In order to prepare the proof of this theorem, we need two lemmas. For an infinite chase sequence $S = I_1 \xrightarrow{\alpha_1,\overline{a}_1} I_2 \xrightarrow{\alpha_2,\overline{a}_2} \ldots$ and $k \geq 2$ we define $\delta_{S,k}$ to be the minimal natural number such that for every $l \geq \delta_{S,k}$ there are $\alpha_{i_1}, ..., \alpha_{i_k} = \alpha_l$ such that $(I_{i_j+1} \backslash I_{i_j}) \cap body(\alpha_{i_{j+1}})(\overline{a}_{i_{j+1}}) \neq \emptyset$.

Lemma 76. Let an infinite chase sequence $S = I_1 \xrightarrow{\alpha_1,\overline{a}_1} I_2 \xrightarrow{\alpha_2,\overline{a}_2} \ldots$, $P \subseteq \text{pos}(\Sigma)$ and $i \geq \delta_{S,k}$ be given. If there is $n \in \Delta_{null} \cap (I_{i+1} \backslash I_i)$ and $n \in dom(body(\alpha_j)(\overline{a}_j)) \cap dom(head(\alpha_j)(\overline{a}_j))$ for some $j > i$, then there is a directed path from α_i to α_j in $G'_k(\Sigma)$.

Sketch of proof. Recall the definition of *rank* from the Proof of Lemma 34. We adapt this notion for our needs here. Let $(\alpha_h, \overline{a}_h)$ be the chase step in which n was freshly created by the chase. We say that every atom $t \in I_h$ has rank zero, i.e. $rank(t) = 0$. For every atom $t \in I_{h+1} \backslash I_h$ in which n occurs we set $rank(t) = 1$. For $i' \geq h+1$ every atom in $t \in I_{i'+1} \backslash I_{i'}$ we set $rank(t) = 1 + max\{rank(t') \mid t' \in body(\alpha_{i'})(\overline{a}_{i'})\}$ if there is some $t' \in body(\alpha_{i'})(\overline{a}_{i'})$ such that $rank(t') > 0$ and there is some $m \in \Delta_{null} \cap body(\alpha_{i'})(\overline{a}_{i'}) \cap head(\alpha_{i'})(\overline{a}_{i'})$ such that $null - pos\{m\}, I_h \subseteq P$. Otherwise, we set $rank(t) = 0$. For a set of atoms A we set $rank(A) = max\{rank(t) \mid t \in A\}$.

Consider $J := body(\alpha_j)(\overline{a}_j)$. We prove our claim by induction on $rank(J)$. If $rank(J) = 1$ then the claim of this lemma follows immediately from the chase sequence from the prerequisites. We set $I := body(\alpha_i)(\overline{a}_i) \cup body(\alpha_j)(\overline{a}_j) \backslash (I_{i+1} \backslash I_i)$, $\overline{a}_{k-1} = \overline{a}_i$, $\overline{a}_k = \overline{a}_j$ in order ro satisfy the conditions of Definition 68. As we have chosen $i \geq \delta_{S,k}$, we can conclude that there must be $\beta_1, ..., \beta_{k-2} \in \Sigma$ such that $\prec_{k,P} (\beta_1, ..., \beta_{k-2}, \alpha_i, \alpha_j)$, from which follows the claim.

If $rank(J) > 1$. Choose $t \in J$ such that $rank(t)$ is maximal and t was added to I_j by the chase step $(\alpha_l, \overline{a}_l)$ with $l < j$ and l maximal with this property. Then, we have that $J' \xrightarrow{*,\alpha_l,\overline{a}_l} J$, where $J' := body(\alpha_l)(\overline{a}_l) \cup J \backslash (I_{l+1} \backslash I_l)$. We can conclude that there must be $\beta_1, ..., \beta_{k-2} \in \Sigma$ such that $\prec_{k,P} (\beta_1, ..., \beta_{k-2}, \alpha_l, \alpha_j)$ becauise we have $j > l > i \geq \delta_{S,k}$. By induction hypothesis we can also conclude that there is a directed path from α_i to α_l in $G'_k(\Sigma)$, which completes this proof. □

The previous lemma is used to obtain a second lemma.

Lemma 77. If every strongly connected component of the minimal k-restriction system of Σ is in $\mathrm{CT}_{\forall\forall}$, so is Σ.

Proof Sketch. Assume that we have a database instance I_0 such that the chase does not terminate. We will construct an infinite chase sequence that uses only constraints from some strongly connected component C_i of Σ's minimal k-restriction system $G' = ((\Sigma, E'), f')$.

We have an infinite chase sequence $S = I_1 \overset{\alpha_1,\overline{a}_1}{\longrightarrow} I_2 \overset{\alpha_2,\overline{a}_2}{\longrightarrow} \dots$. Without loss of generality, we can assume that

1. every constraint from Σ fires infinitely often, and
2. that for every $j \in \mathbb{N}$ there is some $i > j$ such that $I'_i \models \alpha_i(\overline{a}_i)$, where $I'_1 := I_1$, $J_l \overset{\alpha_l,\overline{a}_l}{\to} J_{l+1}$ for $l \neq j$ and $J_{j+1} := J_j$.

This infinite chase sequence will serve as a witness for the fact that some strongly connected component of the minimal k-restriction system has already an infinite chase sequence. We construct a partial k-restriction system $G = ((\Sigma, E), f)$, i.e. $E \subseteq E'$ and $f \subseteq f'$, in which we only add edges according to the proof of Lemma 76 are satisfied. Clearly, G has a non-empty strongly connected component. If it has exactly one such strongly connected component, then we are done. Otherwise, we choose some strongly connected component C of G that has only predecessors in C itself. It holds that there is some strongly connected component C' in G' such that $C \subseteq C'$ due to monotonicity of G''s computation. Let $j \in \mathbb{N}$ such that $\alpha_j \in C$ is a TGD with existential quantifiers (which must exist because otherwise α_j would have a predecessor not in C). We can conclude that the chase with I_j and C has an infinite chase sequence or otherwise some element in C would have a predecessor not in C. $\square$

We can now turn our attention to the theorem's proof, which concludes this section.

Proof Sketch of Theorem 75. The proof of Theorem 75 now is by induction on the depth d of recursive calls of $\mathit{part}(\Sigma, k)$. If $d = 0$, then Σ is safe and the claim follows from Theorem 44. If $d > 1$, then we consider the strongly connected components $C_1, ..., C_n$ of the minimal restriction system of Σ and apply Lemma 77. We obtain $\Sigma \in \mathrm{CT}_{\forall\forall}$.

By induction hypothesis, chasing with C_i terminates in time $Q_i(|\mathit{dom}(I)|)$ for some polynomial Q_i that depends only on C_i. If $C_i \neq C_j$ and there is a path from C_i to C_j, then there can be only polynomially many chase steps with constraints from C_i. These can trigger polynomially many chase steps with constraints in

```
check($\Sigma$: Set of TGDs and EGDs, $k$: not equal to 1) {
    if ($\Sigma$ is safe) then
        return true;
    endif
    compute the strongly connected components (as
    sets of constraints) $C_1, \ldots, C_n$ of the minimal
    $k$-restriction system of $\Sigma$;
    if (n == 0) then
        return true;
    endif
    if (n == 1) then
        if ($C_1 \neq \Sigma$) then
            return check($C_1$,$k$);
        endif
        return false;
    endif
    for i=1 to n do
        if (not check($C_i$,$k$)) then
            return false;
        endif
    endfor
    return true; }
```

Figure 6.7: Algorithm to decide membership in $\forall\forall$-T[$\cdot$].

C_j but not the other way round. If $C_i \neq C_j$ and there is no path from C_i to C_j, then every atom that is created by some $\alpha \in C_i$ can never be used to copy null values by some $\beta \in C_j$. Thus, a cascading of labeled nulls can happen only polynomially many times because it can happen only polynomially many times in every strongly connected component. Thus, the overall computation takes only polynomially many chase steps. $\square$

6.7 An Algorithmic Approach

This section aims to develop an efficient algorithm to test membership in $\forall\forall$-T[k]. We have seen before that the computation of k-restriction systems is costly because we need NP time to compute the relation $\prec_{k,P}$. For this reason, we present an algorithm that avoids the computation of k-restriction systems if possible. It relies on the idea that the stronger condition safety can be checked in polynomial time

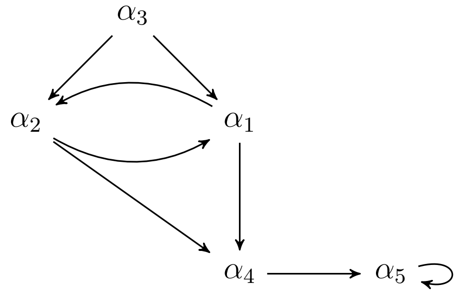


Figure 6.8: 2-restriction system for Example 79.

(cf. Section 6.3). Before computing the k-restriction system, we always check for safety and, whenever safety holds, we conclude that the chase for the respective constraint set terminates and omit the computation of the k-restriction system. We illustrate this idea by a simple example.

Example 78. Consider the constraint from $\Sigma := \{\beta\}$ Examples 37 and 40, where

$$\beta := \mathtt{R}(x_1, x_2, x_3),\ \mathtt{S}(x_2) \rightarrow \exists y\ \mathtt{R}(x_2, y, x_1).$$

Assume that we want to test if this constraint falls into some (fixed) level k of the ∀∀-T-hierarchy. Computing a k-restriction system is superfluous because we already know that this constraint is safe and its membership in ∀∀-T[k] trivially follows from the satisfaction of the safety condition.

In general, the situation is, of course, not that simple.

Example 79. Consider for instance the constraint set $\Sigma' = \{\alpha_1, \alpha_2, \alpha_3\}$, where

$$\begin{aligned} \alpha_1 :=&\quad \mathtt{S}(x),\ \mathtt{E}(x,y) \rightarrow \mathtt{E}(y,x),\\ \alpha_2 :=&\quad \mathtt{S}(x),\ \mathtt{E}(x,y) \rightarrow \exists z\ \mathtt{E}(y,z) \text{ and}\\ \alpha_3 :=&\quad \exists x, y\ \mathtt{S}(x),\ \mathtt{E}(x,y) \end{aligned}$$

from Example 62 extended by $\{\alpha_4, \alpha_5\}$, where

$$\alpha_4 := \mathtt{E}(x_1, x_2) \rightarrow \mathtt{T}(x_1, x_2),$$
$$\alpha_5 := \mathtt{T}(x_1, x_2),\ \mathtt{T}(x_2, x_3) \rightarrow \mathtt{T}(x_1, x_3),$$

and call the resulting constraint set Σ''.

Assume we want to show that Σ'' is inductively restricted (i.e. membership in $\forall\forall$-T[2]). It follows from Example 62 that Σ'' is not safe because Σ' is not.
In direct correspondence to Example 62 it follows that the minimal 2-restriction system for Σ'' is G'(Σ''):=(G',f), where G' is depicted in Figure 6.8 and $f := \{\texttt{E}^1, \texttt{E}^2, \texttt{S}^1, \texttt{T}^1, \texttt{T}^2\}$.
This 2-restriction system contains the strongly connected components $\{\alpha_1,\alpha_2\}$ and $\{\alpha_5\}$. For $\{\alpha_1,\alpha_2\}$ we must compute its minimal 2-restriction system because it is not safe, but for $\{\alpha_5\}$ we can avoid this additional complexity because we know that α_5 is safe as it is a full TGD and therefore the chase terminates.

We implement the scheme described above in algorithm *check*, provided in Figure 6.7.

Proposition 80. Algorithm *check* terminates and correctly decides membership in the $\forall\forall$-T-hierarchy, i.e. there exists $k' \in [k]$ such that $\mathit{check}(\Sigma, k')$ returns true if and only if $\Sigma \in \forall\forall\text{-T}[k]$.

Proof Sketch. The algorithm terminates because all recursive calls are made on constraint sets with size smaller than the input constraint set. What the algorithm does is trying to avoid the computation of k-restriction systems by testing for safeness. The correctness follows from the proof of Theorem 75 because the only property we need to show is that for all $\Sigma' \in \mathit{part}(\Sigma, k)$ the chase terminates in the sense of $\text{CT}_{\forall\forall}$, which is ensured by the additional safety checks. ☐

Chapter 7

Exploring $\mathrm{CT}_{\forall\exists}$

> **Sergio:** "What would Alan say?"
> **Alice:** "Oh, he'd say that he likes Theorem 87."
> **Riccardo:** "I'm getting another schnitzel."

$\mathrm{CT}_{\forall\exists}$ ensures the existence of at least one terminating chase sequence for every database instance. The literature on the chase lacks a systematic study of $\mathrm{CT}_{\forall\exists}$ and concentrates solely on $\mathrm{CT}_{\forall\forall}$. We fill this gap by identifying decidable fragments of it. To the best of our knowledge, this is the first study of sufficient termination conditions for the chase that do not ensure the termination of all chase sequences but of at least one.

But how can we safely apply the chase and be sure that the chase sequence we follow terminates? How can such a sequence be found? A general solution seems to be quite simple. We apply the chase in a breadth-first manner[1]. Using this technique, we can always guarantee termination. However, this method is quite inefficient because it needs a lot of runtime to follow all chase sequences and it also uses a lot of memory.

The results of our study on sequence-dependent chase termination have an important additional property. We cannot only ensure that there is a terminating chase sequence, but we can statically determine it, while checking our termination conditions. This has an important implication. We do not have to apply the chase in the breadth-first, but in the usual depth-first manner, thus saving much time and space.

As a starting point for our work, we make the observation that the stratification condition introduced in [Deutsch et al., 2008] does not belong to $\mathrm{CT}_{\forall\forall}$, as stated by the authors of [Deutsch et al., 2008], but to $\mathrm{CT}_{\forall\exists}$. It is the cornerstone for all further sequence-dependent termination conditions for the chase. We start with an introduction to stratified constraint sets.

---

[1] Applying the chase breadth-first means that for all $n = 1, 2, 3, ...$ we generate all possible chase sequences of length bounded by n and we abort this process when we have found a terminating sequence.

7.1 Stratification

In [Fagin et al., 2005] a syntactic restriction was introduced, called weak acyclicity, which guarantees chase termination in the sense of $\mathrm{CT}_{\forall\forall}$. It can be checked in polynomial time whether a set of constraints is weakly acyclic. At the time weak acyclicity was found by the authors of [Fagin et al., 2005] it comprised every other known termination condition for the chase. In [Deutsch et al., 2008], stratification was introduced which meant to improve the former weak acyclicity condition. The main idea behind stratification is to decompose the constraint set into independent subsets that are then separately tested for weak acyclicity. More precisely, the decomposition splits the input constraint set into subsets of constraints that may cyclically cause to fire each other. The idea is that the termination guarantee for the full constraint set should follow if weak acyclicity holds for each subset in the decomposition.
The basis for such a decomposition is the binary relation $\prec$ on the constraint set.

Definition 81. (see [Deutsch et al., 2008]) Given two TGDs or EGDs $\alpha, \beta \in \Sigma$ we define $\alpha \prec \beta$ iff there exists a relational database instance I and $\overline{a}, \overline{b}$ such that

1. $I \not\models \alpha(\overline{a})$,
2. $I \models \beta(\overline{b})$,
3. $I \overset{\alpha,\overline{a}}{\rightarrow} J$, and
4. $J \not\models \beta(\overline{b})$.

Intuitively, $\alpha \prec \beta$ means that if α fires, it can cause β to fire (in the case that β could not fire before). In contrast to c-stratification, we do not use an oblivious but a standard chase step in the definition. We give an example to illustrate this definition.

Example 82. Consider the following constraints, where $c, d \in \Delta$:

$$\begin{aligned} \alpha &:= \mathtt{E}(x_1, x_2) \rightarrow \exists y\ \mathtt{R}(x_2, y, c) \text{ and} \\ \beta &:= \mathtt{R}(x_1, x_2, d) \rightarrow \mathtt{E}(x_1, x_2). \end{aligned}$$

It holds that $\alpha \not\prec \beta$ because α can only create atoms which have the constant c in their third position, whereas β requires the constant d in its first position. We have that $\beta \prec \alpha$ because we can take the instance $\{\mathtt{R}(a, b, d)\}$ and the tuples $\overline{a} = \overline{b} = a, b$. Besides, $\alpha \not\prec \alpha$ and $\beta \not\prec \beta$ hold.

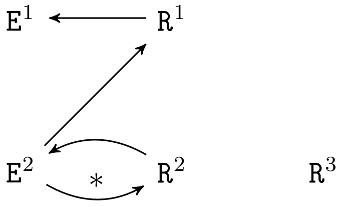


Figure 7.1: Dependency graph for Example 84.

The actual definition of stratification then relies, as outlined before, on the notion of weak acyclicity.

Definition 83. (see [Deutsch et al., 2008]) The chase graph $G(\Sigma) = (\Sigma, E)$ of a set of constraints Σ contains a directed edge (α, β) between two constraints iff $\alpha \prec \beta$. We call Σ stratified iff the constraints in every cycle of $G(\Sigma)$ are weakly acyclic.

Stratification strictly generalizes weak acyclicity (see [Deutsch et al., 2008]) in the sense that

1. if Σ is weakly acyclic, then it is also stratified and
2. there are constraint sets that are stratified but not weakly acyclic (cf. Example 84).

Example 84. Consider the constraint $\{\alpha, \beta\}$ from Example 82. It holds that $\prec = \{(\beta, \alpha)\}$, so $\{\alpha, \beta\}$ is stratified. The dependency graph of $\{\alpha, \beta\}$ is depicted in Figure 7.1 and contains a cycle through a special edge, so $\{\alpha, \beta\}$ is not weakly acyclic.

As shown in [Deutsch et al., 2008] it can be decided in coNP whether a set of constraints is stratified. The authors of [Deutsch et al., 2008] claim the following result:

Statement 85. (cf. [Deutsch et al., 2008]) Let Σ be a fixed set of stratified constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for every database instance I, the length of every chase sequence is bounded by $Q(|dom(I)|)$. Thus, $\Sigma \in \mathrm{CT}_{\forall\forall}$.

Unfortunately, as we show in the following example, this statement is not true. Please note that we introduced the example (cf. Example 31) already earlier in Chapter 6. For convenience, we repeat it here again.

Example 86. We consider the constraint set from Example 31 again. We repeat it here for convenience as it stems from the beginning of Chapter 6. Given the set of TGDs $\Sigma = \{\alpha_1, ..., \alpha_4\}$, where

$$\begin{array}{lll}
\alpha_1 & := & \mathtt{R}(x_1) \rightarrow \mathtt{S}(x_1, x_1), \\
\alpha_2 & := & \mathtt{S}(x_1, x_2) \rightarrow \exists z\ \mathtt{T}(x_2, z), \\
\alpha_3 & := & \mathtt{S}(x_1, x_2) \rightarrow \mathtt{T}(x_1, x_2),\ \mathtt{T}(x_2, x_1) \text{ and} \\
\alpha_4 & := & \mathtt{T}(x_1, x_2),\ \mathtt{T}(x_1, x_3),\ \mathtt{T}(x_3, x_1) \rightarrow \mathtt{R}(x_2).
\end{array}$$

We will give now an instance for which the chase does not necessarily terminate. Consider the database $\{\mathtt{R}(a)\}$ and the chase sequence which applies the constraints in the order $\alpha_1, ..., \alpha_4, \alpha_1, ..., \alpha_4, ...$ and so on. The first steps of the resulting chase sequence look as follows:

$$\begin{array}{ll}
 & \{\mathtt{R}(a)\} \\
\xrightarrow{\alpha_1, a} & \{\mathtt{R}(a),\ \mathtt{S}(a, a)\} \\
\xrightarrow{\alpha_2, a, a} & \{\mathtt{R}(a),\ \mathtt{S}(a, a),\ \mathtt{T}(a, n_1)\} \\
\xrightarrow{\alpha_3, a, a} & \{\mathtt{R}(a),\ \mathtt{S}(a, a),\ \mathtt{T}(a, n_1),\ \mathtt{T}(a, a)\} \\
\xrightarrow{\alpha_4, a, n_1, a} & \{\mathtt{R}(a),\ \mathtt{S}(a, a),\ \mathtt{T}(a, n_1),\ \mathtt{T}(a, a),\ \mathtt{R}(n_1)\} \\
\xrightarrow{\alpha_1, n_1} & \{\mathtt{R}(a),\ \mathtt{S}(a, a),\ \mathtt{T}(a, n_1),\ \mathtt{T}(a, a),\ \mathtt{R}(n_1),\ \mathtt{S}(n_1, n_1)\} \\
\xrightarrow{\alpha_2, n_1, n_1} & \{\mathtt{R}(a),\ \mathtt{S}(a, a),\ \mathtt{T}(a, n_1),\ \mathtt{T}(a, a),\ \mathtt{R}(n_1),\ \mathtt{S}(n_1, n_1),\ \mathtt{T}(n_1, n_2)\} \\
\xrightarrow{\alpha_3, n_1, n_1} & \{\mathtt{R}(a),\ \mathtt{S}(a, a),\ \mathtt{T}(a, n_1),\ \mathtt{T}(a, a),\ \mathtt{R}(n_1),\ \mathtt{S}(n_1, n_1),\ \mathtt{T}(n_1, n_2),\ \mathtt{T}(n_1, n_1)\} \\
\xrightarrow{\alpha_4, n_1, n_2, n_1} & \{\mathtt{R}(a),\ \mathtt{S}(a, a),\ \mathtt{T}(a, n_1),\ \mathtt{T}(a, a),\ \mathtt{R}(n_1),\ \mathtt{S}(n_1, n_1),\ \mathtt{T}(n_1, n_2),\ \mathtt{T}(n_1, n_1), \\
 & \mathtt{R}(n_2)\}, \\
\xrightarrow{\alpha_1, n_2} & \ldots
\end{array}$$

where n_1, n_2 are fresh null values. It can be easily seen that this sequence is infinite. The chase graph for Σ is depicted in Figure 7.2. The only cycle in it is constituted by full TGDs only and therefore, it is weakly acyclic. Hence, it follows that Σ is stratified.

In order to show that $\alpha_2 \not\prec \alpha_4$ holds assume for a moment $\alpha_2 \prec \alpha_4$. Choose a database instance I and $\overline{a}, \overline{b}$ according to Definition 81. Say, $\overline{a} = c, d$, where c and d may not be distinct. α_2 creates a fresh labelled null n_1 in our instance. Therefore, $\overline{b} = d, n_1, e$, where d, e may not be distinct. We see that $T(d, e) \in I$, which implies $I \models \alpha_2(\overline{a})$. This contradicts our assumption.

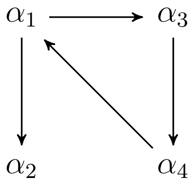


Figure 7.2: Chase graph for Example 86.

As a consequence, unlike weak acyclicity, stratification does not ensure termination in the sense of $\mathrm{CT}_{\forall\forall}$.
However, we can prove another, equally useful result with the definition of stratification as in [Deutsch et al., 2008]. If a set of constraints is stratified, we cannot ensure termination in the sense of $\mathrm{CT}_{\forall\forall}$ but $\mathrm{CT}_{\forall\exists}$ as stated in the following theorem. We want to emphasize that this result was not stated by the authors of [Deutsch et al., 2008] but is our own finding.

Theorem 87. Let Σ be a fixed set of stratified constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for any database instance I there is a terminating chase sequence whose length is bounded by $Q(|dom(I)|)$. Thus, $\Sigma \in \mathrm{CT}_{\forall\exists}$.

As we are going to generalize this statement on the following pages we will omit its formal proof here and refer the reader to the more general Theorem 89. The polynomial data complexity follows immediately from the polynomial data complexity of weakly acyclic constraint sets.

But how can we use this result in practice? The first idea is to apply the chase in a breadth-first manner, i.e. generating a tree whose root is the start instance, its children are obtained by applying one chase step on the start instance and the tree is expanded in a breadth-first manner. This ensures that if there is a terminating chase sequence, then we will find it. Unfortunately, this is rather ineffective because in some intermediate instance there may be many constraints violated and therefore, the degree and the depth of the tree may be high.
As it turns out, we are in a much better situation here. We can use the chase graph to statically construct the order in which the constraints can be applied to ensure a terminating chase sequence. We give an example to illustrate this.

Example 88. Consider the constraint set Σ from Example 86 again and the instance $\{\mathtt{R}(a),\, \mathtt{T}(b,b)\}$. We give a chase sequence that terminates.

$$\{\mathtt{R}(a),\, \mathtt{T}(b,b)\}$$

$$\begin{array}{ll}
\xrightarrow{\alpha_1,a} & \{\mathtt{R}(a),\ \mathtt{T}(b,b),\ \mathtt{S}(a,a)\} \\
\xrightarrow{\alpha_3,a,a} & \{\mathtt{R}(a),\ \mathtt{T}(b,b),\ \mathtt{S}(a,a),\ \mathtt{T}(a,a)\} \\
\xrightarrow{\alpha_4,b,b,b} & \{\mathtt{R}(a),\ \mathtt{T}(b,b),\ \mathtt{S}(a,a),\ \mathtt{T}(a,a),\ \mathtt{R}(b)\} \\
\xrightarrow{\alpha_1,b} & \{\mathtt{R}(a),\ \mathtt{T}(b,b),\ \mathtt{S}(a,a),\ \mathtt{T}(a,a),\ \mathtt{R}(b),\ \mathtt{S}(b,b)\}
\end{array}$$

It holds that $\{\mathtt{R}(a),\ \mathtt{T}(b,b),\ \mathtt{S}(a,a),\ \mathtt{T}(a,a),\ \mathtt{R}(b),\ \mathtt{S}(b,b)\} \models \Sigma$. We obtained a terminating chase sequence by first chasing with the constraints in the cycle and after the chase with these constraints is finished, we (possibly) chase with α_2, which was not necessary here. It can be shown that this strategy always leads to finite chase sequences, regardless of the underlying instance.

The observations made in this example can be generalized to the next theorem.

Theorem 89. Let Σ be a fixed set of constraints. If for every strongly-connected component Σ' of $G(\Sigma)$ it holds that $\Sigma' \in \mathrm{CT}_{\forall\exists}$ and a terminating chase sequence can be statically constructed, then $\Sigma \in \mathrm{CT}_{\forall\exists}$ and a terminating chase sequence can be statically constructed.

Proof. Let the chase graph $G(\Sigma) = (\Sigma, E)$ be given. We write $\alpha \sim \beta$ if and only if α and β are contained in a common cycle in $G(\Sigma)$ or $\alpha = \beta$. Note that $\sim$ is an equivalence relation.
Let $\Sigma/\sim = \{W_1, ..., W_n\}$ and $E' := \{\ (W_i, W_j) \mid i, j \in [n], i \neq j,$ there is some $\alpha_i \in W_i, \beta_j \in W_j$ such that $\alpha_i \prec \beta_j\ \}$. Let $W_1', ..., W_n'$ be a topological sorting of $(\Sigma/\sim, E')$. Note that $W_1', ..., W_n'$ are the strongly connected components of the chase graph and constraint sets that are not involved in any cycle in the chase graph, therefore the chase terminates independently of the database instance and independently of the chase order for these constraint sets. Let I_0 be an arbitrary database instance. Let I_i be obtained from I_{i-1} by chasing with W_i' according to the chase strategy from the prerequisites for every $i \in [n]$. It holds that $I_2 \models W_1'$. Otherwise, there is some $\alpha \in W_1'$, $\beta \in W_2'$ and a database instance I such that $I \models \alpha$, but $I \xrightarrow{\beta,\bar{b}} J \not\models \alpha$. But this implies $\beta \prec \alpha$ which means W_2' must come before W_1' in the topological sorting of $(\Sigma/\sim, E')$. Using induction on n it can be seen that $I_n \models \Sigma$. Observe that $W_1', ..., W_n'$ is a partition of Σ. □

This allows us to safely apply the chase procedure in situations when the termination cannot be guaranteed for every chase sequence. We avoid the overhead of branching in the breadth-first chase and therefore reduce the complexity of generating a chase result.

7.2 The ∀∃-T-Hierarchy

We continue our study of $\text{CT}_{\forall\exists}$ by combining our ideas related to the ∀∀-T-hierarchy in Section 6.6 with stratification in Section 7.1, thus creating much larger decidable fragments of $\text{CT}_{\forall\exists}$ than stratification. We also propose an efficient algorithm that decides membership for a given level in this hierarchy.

We take a simple idea to do that in changing the weak acyclicity condition in the definition of stratification to some larger fragment of $\text{CT}_{\forall\forall}$ like safety or some level of the ∀∀-T-hierarchy. This creates indeed a fragment of $\text{CT}_{\forall\exists}$ because the proof of Theorem 87 does not depend on the notion of weak acyclicity but solely on the observation that weak acyclicity is a fragment of $\text{CT}_{\forall\forall}$. A possible definition of such a class is as follows.

Definition 90. Let $k \geq 2$. A set of constraints Σ is in ∀∃-T$[k]$ iff every strongly connected component of $G(\Sigma)$ is in ∀∀-T$[k]$.

As a first property we want to mention that this hierarchy is strict.

Example 91. (cf. to Example 73) We have seen in Example 73 that the constraint set $\Sigma_k := \{\alpha_k\}$, where

$$\alpha_k := \mathtt{S}(x_k), \mathtt{R}_k(x_1, ..., x_k) \to \exists y\ \mathtt{R}_k(y, x_1, ..., x_{k-1})$$

is in ∀∀-T$[k+1]$\∀∀-T$[k]$. As we also have that $\alpha_{k+1} \prec \alpha_{k+1}$, we can conclude that $\Sigma \in$ ∀∃-T$[k+1]$\∀∃-T$[k]$.

As already explained, the following corollary is easily obtained.

Corollary 92. (to Theorem 89) Let $k \geq 2$ and $\Sigma \in$ ∀∃-T$[k]$ be a fixed set of constraints. Then, there exists a polynomial $Q \in \mathbb{N}[X]$ such that for every database instance I the length of every chase sequence is bounded by $Q(|dom(I)|)$. Thus, $\Sigma \in \text{CT}_{\forall\exists}$.

The definition's use can be illustrated by the example now to be mentioned.

Example 93. We consider the constraint set from Example 31 again and combine it with the constraint set from Example 51 and an additional full TGD. We repeat the whole constraint here for convenience. Let Σ be the following set of TGDs:

$$\begin{aligned}
\alpha_1 &:= \mathtt{R}(x_1) \rightarrow \mathtt{S}(x_1, x_1),\\
\alpha_2 &:= \mathtt{S}(x_1, x_2) \rightarrow \exists z\ \mathtt{T}(x_2, z),\\
\alpha_3 &:= \mathtt{S}(x_1, x_2) \rightarrow \mathtt{T}(x_1, x_2),\ \mathtt{T}(x_2, x_1),\\
\alpha_4 &:= \mathtt{T}(x_1, x_2),\ \mathtt{T}(x_1, x_3),\ \mathtt{T}(x_3, x_1) \rightarrow \mathtt{R}(x_2),\\
\alpha_5 &:= \mathtt{T}(x_1, x_2) \rightarrow \mathtt{E}(x_1, x_2),\\
\alpha_6 &:= \mathtt{U}(x_1),\ \mathtt{E}(x_1, x_2) \rightarrow \mathtt{E}(x_2, x_1) \text{ and}\\
\alpha_7 &:= \mathtt{U}(x_1),\ \mathtt{E}(x_1, x_2) \rightarrow \exists y\ \mathtt{E}(x_2, y),\ \mathtt{E}(y, x_1).
\end{aligned}$$

It is clear that $\Sigma \in \mathrm{CT}_{\forall\exists}$ because we can first apply a terminating chase sequence to satisfy the constraints $\{\alpha_1, ..., \alpha_4\}$ and then apply the rest of the constraints afterwards arbitrarily. After finitely many steps this chase sequence will terminate.

The important point to see is that no termination condition for the chase introduced so far recognizes that there is always a terminating chase sequence for Σ. On the one hand, Σ is not in any level of the ∀∀-T-hierarchy because for $\{\alpha_1, ..., \alpha_4\}$ there may be non-terminating chase sequences (cf. Example 31). On the other hand, Σ can also not be stratified because the constraint set $\{\alpha_6, \alpha_7\}$ is not weakly acyclic (cf. Example 51).
We will now show that $\Sigma \in \forall\exists\text{-T}[2]$. The chase graph is depicted in Figure 7.3. We see that there are two strongly connected components, namely $\{\alpha_1, \alpha_3, \alpha_4\}$ and $\{\alpha_6, \alpha_7\}$. As both are in ∀∀-T[2], we can conclude that $\Sigma \in \forall\exists\text{-T}[2]$.

Like in the case of stratification, we are not only able to prove that there must be a terminating chase sequence but can also determine such a sequence statically from the shape of the chase graph. We chase sequentially with

1. $\{\alpha_1, \alpha_3, \alpha_4\}$ until these constraints are satisfied,
2. then take α_2 and apply it until it is satisfied,
3. afterwards, we do every possible chase step with α_5 and
4. finally, we chase with the second strongly connected component $\{\alpha_6, \alpha_7\}$.

At the end, all constraints are satisfied and therefore the chase sequence is finite.

This example shows us that a further decomposition of the strongly connected components of the chase graph via the ∀∀-T-hierarchy leads to improved termination conditions.

We present an observation made during the example. As a corollary of Theorem 89 we can state that for every database instance a terminating chase sequence can

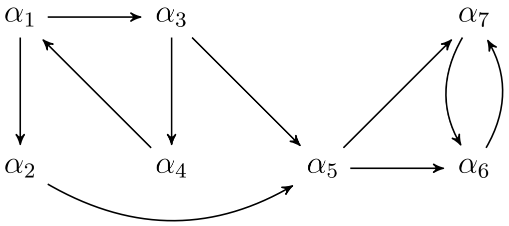


Figure 7.3: Chase graph for Example 93.

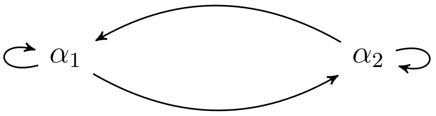


Figure 7.4: Chase graph for Example 95.

always be statically constructed from the underlying chase graph.

Corollary 94. (to Theorem 89) Let $k \geq 2$ and Σ be a fixed set of constraints in ∀∃-T$[k]$ constraints. For every database instance I a terminating chase sequence for I and Σ can be statically constructed.

As a closing remark we want to mention that the ∀∃-T-hierarchy is a proper subset of $\mathrm{CT}_{\forall\exists}$. We leave weaker decidable fragments of $\mathrm{CT}_{\forall\exists}$ as an open problem for future research.

Example 95. Consider the set of constraints $\Sigma := \{\alpha_1, \alpha_2\}$, where

$$\begin{array}{lcl}\alpha_1 & := & \mathtt{R}(x_1, x_2, x_3),\ \mathtt{S}(x_2) \rightarrow \exists y\ \mathtt{R}(x_2, y, x_1),\ \mathtt{S}(y) \text{ and} \\ \alpha_2 & := & \mathtt{R}(x_1, x_2, x_3),\ \mathtt{S}(x_2) \rightarrow \mathtt{R}(x_2, x_3, x_1),\ \mathtt{S}(x_3).\end{array}$$

Given an arbitrary finite database instance, we always apply the constraint α_2 during the chase. This obviously leads only to finite chase sequences because α_2 is a full TGD. At the end, α_1 will also be satisfied. Observe that $\{\alpha_2\} \models \alpha_1$. The chase graph for Σ is depicted in Figure 7.4. It shows the strongly connected component $\{\alpha_1, \alpha_2\}$, which cannot be in any level of the T-hierarchy because the constraint α_1 has a non-terminating chase sequence for the instance $\{\mathtt{R}(a, b, c), \mathtt{S}(b)\}$. Thus, Σ cannot be in any level of the ∀∃-T-hierarchy.

```
$\forall\exists$-check($\Sigma$: Set of TGDs and EGDs, $k$: not equal to 1) {
    if ($\Sigma$ is safe) then
        return true;
    endif
    compute the strongly connected components (as
    sets of constraints) $C_1$, ..., $C_n$ of the chase
    graph of $\Sigma$;
    for i=1 to n do
        if (not check($C_i$,$k$)) then
            return false;
        endif
    endfor
    return true; }
```

Figure 7.5: Algorithm ∀∃-check to decide membership in ∀∃-T[·].

7.3 A More Efficient Membership Test

This section aims to develop an efficient algorithm to test membership in ∀∃-T[k]. The idea on which we base such an improvement in efficiency is the same as in Section 6.7. We have seen before that the computation of chase graphs and k-restriction systems is costly because we need Np time to compute the relations $\prec$ and $\prec_{k,P}$. For this reason, we try to avoid the computation of chase graphs and k-restriction systems if possible. Like in Section 6.7 we rely on the idea that the stronger condition safety can be checked in polynomial time (cf. Section 6.3). Before computing the chase graph or k-restriction system, we always check for safety and, whenever safety holds, we conclude that the chase for the respective constraint set terminates and omit the computation of the chase graph or k-restriction system. We illustrate this idea by a simple example.

Example 96. Consider the constraint from $\Sigma := \{\beta\}$ Examples 37 and 40, where

$$\beta := \mathtt{R}(x_1, x_2, x_3),\ \mathtt{S}(x_2) \rightarrow \exists y\ \mathtt{R}(x_2, y, x_1).$$

We assume that we want to test if this constraint falls into some (fixed) level k of the ∀∃-T-hierarchy. Computing a chase graph is superfluous because we already know that this constraint is safe and its membership in ∀∃-T[k] trivially follows from the satisfaction of the safety condition.

In general, of course, the situation is not that simple as the next example shows.

Example 97. Consider for instance the constraint set $\Sigma' = \{\alpha_1, ..., \alpha_4\}$ from Examples 82 and 86 again, where

$$\begin{array}{lcl}
\alpha_1 & := & \mathtt{R}(x_1) \rightarrow \mathtt{S}(x_1, x_1), \\
\alpha_2 & := & \mathtt{S}(x_1, x_2) \rightarrow \exists z\ \mathtt{T}(x_2, z), \\
\alpha_3 & := & \mathtt{S}(x_1, x_2) \rightarrow \mathtt{T}(x_1, x_2), \mathtt{T}(x_2, x_1) \text{ and} \\
\alpha_4 & := & \mathtt{T}(x_1, x_2), \mathtt{T}(x_1, x_3), \mathtt{T}(x_3, x_1) \rightarrow \mathtt{R}(x_2).
\end{array}$$

Say that we want to test whether $\Sigma \in \forall\exists\text{-T}[k]$. Its dependency graph is depicted in Figure 7.2. It shows that the only strongly connected component in the chase graph is $C := \{\alpha_1, \alpha_3, \alpha_4\}$. The definition of the ∀∃-T-hierarchy implies that C must be further decomposed via k-restriction systems but this is not necessary here because C consists only of full TGDs and is therefore safe. Thus, we can avoid computing a k-restriction system for C.

We implement the scheme described above in algorithm ∀∃-*check*, provided in Figure 6.7.

Proposition 98. Algorithm ∀∃-check terminates and correctly decides membership in the ∀∃-T-hierarchy, i.e. there exists $k' \in [k]$ such that ∀∃-check(Σ, k') returns true if and only if $\Sigma \in \forall\exists\text{-T}[k]$.

Proof Sketch. The algorithm terminates because all recursive calls are made on constraint sets with a size smaller than the input constraint set. What the algorithm does is trying to avoid the computation of chase graphs and k-restriction systems by testing for safety. The correctness follows from Lemma 77 and Theorem 89 because the only property we need to show is that for all strongly connected components of the chase graph the chase terminates in the sense of $\text{CT}_{\forall\exists}$, which is ensured by the additional safety checks. □

Chapter 8

Data-dependent Chase Termination

> **Riccardo:** "Is there no alternative to all this theory?"
> **Alice:** "Of course. Run the algorithm and monitor what happens."
> **Sergio:** "If nothing bad happens, then everything is fine."

8.1 Motivation

So far, we have discussed conditions that guarantee chase termination for every database instance. In this section we study the problem of data-dependent termination, i.e. given the constraint set Σ and a *fixed* instance I, does the chase with Σ terminate on I? By the best of our knowledge, this problem has not been studied before, so we start our discussion with a motivating scenario. Let us consider the travel agency database in Figure 8.1, where predicate `hasAirport` contains cities that have an airport and `fly` (`rail`) stores flight (rail) connections between cities, including their distance *dist*. In addition to the schema, constraints $\{\alpha_1, \alpha_2, \alpha_3\}$ have been specified, e.g. α_3 might have been added to assert that for each city reachable via plane, the schedule is integrated in the local database. Now consider the conjunctive query q_1 below.

$$q_1\text{: } \texttt{rf}(x_2) \leftarrow \texttt{rail}(c_1, x_1, y_1),\ \texttt{fly}(x_1, x_2, y_2)$$

The query selects all cities that can be reached from c_1 through rail-and-fly. Assume that in the style of semantic query optimization we want to optimize q_1 under constraints Σ using the chase. We then interpret the body of q_1 as database instance $I := \{\texttt{rail}(c_1, x_1, y_1), \texttt{fly}(x_1, x_2, y_2)\}$, where c_1 is a constant and the x_i, y_i labeled nulls. We observe that α_3 does not hold on I, since there is a flight to city x_2 but no outgoing flight from x_2. Hence, the chase adds a new tuple $t_1 := \texttt{fly}(x_2, x_3, y_3)$ to I, where x_3, y_3 are fresh labeled null values. In the resulting instance $I' := I \cup \{t_1\}$, α_3 is again violated (this time for x_3) and in subsequent

Sample Schema: $\texttt{hasAirport}(c\_id)$
$\texttt{fly}(c\_id1, c\_id2, dist)$
$\texttt{rail}(c\_id1, c\_id2, dist)$

Constraint Set: $\Sigma = \{\alpha_1, \alpha_2, \alpha_3\}$, where

α_1 : If there is a flight connection between two cities, both of them have an airport:
$\texttt{fly}(c_1, c_2, d) \rightarrow \texttt{hasAirport}(c_1), \texttt{hasAirport}(c_2)$

α_2 : Rail-connections are symmetrical:
$\texttt{rail}(c_1, c_2, d) \rightarrow \texttt{rail}(c_2, c_1, d)$

α_3 : Each city that is reachable via plane has at least one outgoing flight scheduled:
$\texttt{fly}(c_1, c_2, d) \rightarrow \exists c_3, d'\ \texttt{fly}(c_2, c_3, d')$

Figure 8.1: Sample database schema and constraints.

steps the chase adds $\texttt{fly}(x_3, x_4, y_4)$, $\texttt{fly}(x_4, x_5, y_5)$, $\texttt{fly}(x_5, x_6, y_6)$, ... Obviously, it will never terminate in this setting.

Reasonable applications should not risk non-termination, so for the constraint set in Figure 8.1 termination is in question for *all queries*, although there might be queries for which the chase terminates.[1] Tackling this problem, we propose to investigate data-dependent chase termination, i.e. to study sufficient termination guarantees for a *fixed instance* when no general termination guarantees apply. We illustrate the benefits of having such guarantees for query q_2 below, which selects all cities x_2 that can be reached from c_1 via rail-and-fly and the same transport route leads back from x_2 to c_1 (c_1 is a constant, x_i, y_i are variables).

$$\begin{aligned} q_2\text{:}\quad \texttt{rffr}(x_2) \leftarrow\ & \texttt{rail}(c_1, x_1, y_1), \texttt{fly}(x_1, x_2, y_2), \texttt{fly}(x_2, x_1, y_2), \\ & \texttt{rail}(x_1, c_1, y_1) \end{aligned}$$

Query q_2 violates only α_1. It is easy to verify that the chase terminates for this query and transforms q_2 into q_2':

$$\begin{aligned} q_2'\text{:}\quad \texttt{rffr}(x_2) \leftarrow\ & \texttt{rail}(c_1, x_1, y_1), \texttt{fly}(x_1, x_2, y_2), \texttt{fly}(x_2, x_1, y_2), \\ & \texttt{rail}(x_1, c_1, y_1), \texttt{hasAirport}(x_1), \texttt{hasAirport}(x_2) \end{aligned}$$

---

[1] Note that query optimization can be done with a bounded portion of a chase result but in general we do not find minimal rewritings of the input query in the style of [Deutsch et al., 2006]. Therefore, it is desirable to guarantee chase termination.

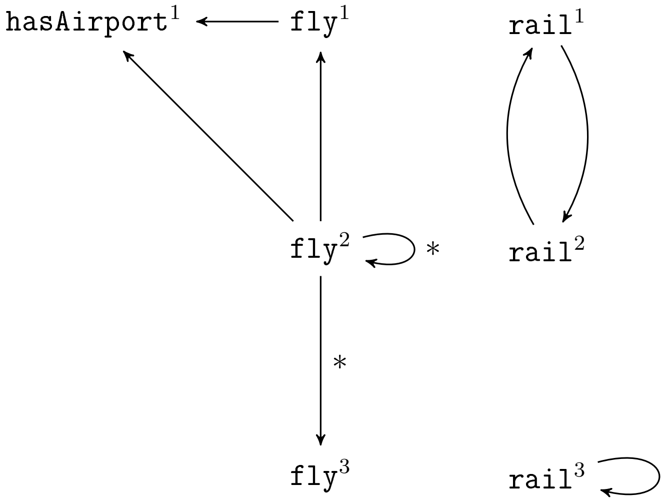


Figure 8.2: Dependency graph for Σ from Figure 8.1.

The resulting query q_2' satisfies all constraints and is a so-called *universal plan* [Deutsch et al., 2006]: intuitively, it incorporates all possible ways to answer the query. As discussed in [Deutsch et al., 2006], the universal plan forms the basis for finding smaller equivalent queries (under the respective constraints) by choosing any subquery of q_2' and testing if it can be chased to a homomorphical copy of q_2'. Using this technique, we can easily show that the following two queries are equivalent to q_2.

q_2'': $\texttt{rffr}(x_2) \leftarrow \texttt{rail}(c_1, x_1, y_1), \texttt{fly}(x_1, x_2, y_2), \texttt{fly}(x_2, x_1, y_2)$
q_2''': $\texttt{rffr}(x_2) \leftarrow \texttt{hasAirport}(x_1), \texttt{rail}(c_1, x_1, y_1), \texttt{fly}(x_1, x_2, y_2),$
$\texttt{fly}(x_2, x_1, y_2)$

Thus, instead of q_2 we could evaluate q_2'' or q_2''', which might well be more performant: in both q_2'' and q_2''' the join with $\texttt{rail}(x_1, c_1, y_1)$ has been eliminated; moreover, if `hasAirport` is duplicate-free, the additional join of `rail` with `hasAirport` in q_2''' may serve as a filter that decreases the size of intermediate results and speeds up query evaluation. This strategy is called *join introduction* in SQO (cf. [King, 1986]). Ultimately, the chase for q_2 makes it possible to detect q_2'' and q_2''', so it would be desirable to have data-dependent termination guarantees that allow us to chase q_2 (and q_2'', q_2'''). We will present such conditions in the remainder of this chapter.

8.2 On $\mathrm{CT}_{I,\forall}$ and $\mathrm{CT}_{I,\exists}$

We want to point out that decidable fragments of $\mathrm{CT}_{I,\forall}$ and $\mathrm{CT}_{I,\exists}$ cannot always guarantee that the chase can be safely applied in query optimization. To be more precise, even if the chase terminates on an instance, there may be subinstances for which there is no terminating chase sequence. For example, this is important for the Chase & Backchase algorithm (cf. Section 4.2) as the subsequent example demonstrates.

Example 99. Consider the set of constraints

$$\Sigma := \{\mathtt{E}(x,y) \rightarrow \exists z\ \mathtt{E}(x,z), \mathtt{E}(z,y)\}$$

and the two queries

$$q_1() \leftarrow \mathtt{E}(x,y), \mathtt{E}(y,x), \mathtt{E}(x,x), \mathtt{E}(y,y) \text{ and}$$
$$q_2() \leftarrow \mathtt{E}(x,y), \mathtt{E}(y,x).$$

Observe that $body(q_1) \models \Sigma$ and that q_2 is a subquery of q_1. We assume that the Chase & Backchase produced the universal plan q_1 and that the subquery q_2 is selected in the backchase phase to be tested for equivalence to q_1. So, the first step is to chase $body(q_2)$ with Σ. Unfortunately, no chase sequence with $body(q_2)$ and Σ terminates.

This implies that even if a decidable fragment of $\mathrm{CT}_{I,\forall}$ or $\mathrm{CT}_{I,\exists}$ can guarantee the termination of the chase for a query, there may be cases where there is a subquery for which no chase sequence terminates at all, so the data-dependent approach must necessarily fail. Thus, in general, only data-independent techniques can always ensure the applicability of Chase & Backchase but in some cases techniques can be used if the data-independent techniques fail as shown at the beginning of this chapter.

Although, the Chase & Backchase cannot always be applied as a blackbox in the data-dependent scenario we can slightly modify it such that an optimization is possible. It is important to note that we cannot guarantee optimality of the results, i.e. minimality of the output queries. The idea is to chase the query as usual in the Chase & Backchase, but to apply the backchase phase only to those subqueries for which data-dependent chase termination guarantees can be made. In this way, we can surely optimize the query, however, the result may not be minimal.

8.3 Results for CT$_{I,\forall}$

Our first approach to data-dependent chase termination is a static one. It relies on the observation that the chase will always terminate on instance I if the subset of constraints that might fire when chasing I with Σ is contained in some level of the $\forall\forall$-T-hierarchy.

Definition 100. We call a constraint $\alpha \in \Sigma$ (I, Σ)-*irrelevant* if and only if there is no chase sequence such that α can eventually fire, i.e. no chase sequence of the form $I \xrightarrow{\alpha_1,\overline{a_1}} \cdots \xrightarrow{\alpha,\overline{a}} \ldots$.

We formalize our observation in Lemma 101.

Lemma 101. Let $k \geq 2$ and $\Sigma' \subseteq \Sigma$ s.t. $\Sigma \setminus \Sigma'$ is a set of (I, Σ)-irrelevant constraints. If $\Sigma' \in T[k]$, then the chase with Σ terminates for instance I.

Proof Sketch. It holds that Σ' contains all constraints that may fire during the execution of the chase starting with I and Σ. $I^{\Sigma'}$ is finite and $I^{\Sigma'} = I^{\Sigma}$. $\square$

Hence, the crucial point is to effectively compute the set of (I, Σ)-irrelevant constraints. Unfortunately, it turns out that checking (I, Σ)-irrelevance is an undecidable problem in general:

Theorem 102. Let Σ be a set of constraints, $\alpha \in \Sigma$ a constraint, and I an instance. It is undecidable if α is (I, Σ)-irrelevant.

Proof Sketch. It is well-known that the following problem is undecidable: given a Turing machine M and and a state transition t from the description of M, does M reach t (given the empty string as input)? From (M, t), we will compute a set of TGDs and EGDs Σ_M and a TGD $\alpha_t \in \Sigma_M$ such that the following equivalence holds: M reaches t (given the empty string as input) $\Leftrightarrow$ there is a chase sequence in the computation of the chase with Σ_M applied to the empty instance such that α_t will eventually fire.
Our reduction uses the construction in the proof of Theorem 1 in [Deutsch et al., 2008]. To be self-contained, we review it here again. We use the signature consisting of the relation symbols:

- $\mathtt{T}(x, a, y)$ tape "horizontal" edge from x to y with symbol a,
- $\mathtt{H}(x, s, y)$ head "horizontal" edge from x to y with state s,
- $\mathtt{L}(x, y)$ left "vertical" edge,

- $\mathtt{R}(x, y)$ right “vertical” edge,
- $\mathtt{A}_\delta(x)$, $\mathtt{B}_\delta(x)$ for every state transition δ,
- one constant for every tape symbol,
- one constant for every head state,
- the special constant B marking the beginning of the tape, and
- $\Box$ to denote an empty tape cell.

The set of constraints Σ_M is as follows.

1. The initial configuration:

$$\exists w, x, y, z\ \mathtt{T}(w, B, x),\ \mathtt{T}(x, \Box, y),\ \mathtt{H}(x, s_0, y),\ \mathtt{T}(y, E, z),$$

   where $\Box$ is the blank symbol and s_0 is the initial state (both are constants).

2. For every state transition δ which moves the head to the right, replacing symbol a with a' and going from state s to state s':

$$\begin{aligned}\mathtt{T}(x, a, y),\ \mathtt{H}(x, s, y),\ \mathtt{T}(y, b, z) \rightarrow\ & \exists x', y', z'\ \mathtt{L}(x, x'),\ \mathtt{R}(y, y'),\ \mathtt{R}(z, z'),\\ & \mathtt{T}(x', a', y'),\ \mathtt{T}(y', b, z'),\ \mathtt{H}(y', s', z'),\ \mathtt{A}_\delta(w').\end{aligned}$$

   Here a, s, a', b, and s' are constants.

3. For every state transition δ which moves the head to the right past the end of the tape replacing symbol a with a’ and going from state s to state s’:

$$\begin{aligned}\mathtt{T}(x, a, y),\ \mathtt{H}(x, s, y),\ \mathtt{T}(y, E, z) \rightarrow\ & \exists w', x', y', z'\ \mathtt{L}(x, x'),\ \mathtt{R}(y, y'),\ \mathtt{R}(z, z'),\\ & \mathtt{T}(x', a', y'),\ \mathtt{T}(y', \Box, z'),\ \mathtt{H}(y', s', z'),\\ & \mathtt{T}(y', E, w'),\ \mathtt{A}_\delta(w').\end{aligned}$$

   Here a, s, a', b, and s' are constants.

4. Similarly for state transitions which move the head to the left.
5. Similarly for state transitions which do not move the head.
6. For every state transition δ:

$$\mathtt{A}_\delta(x) \rightarrow \mathtt{B}_\delta(x)$$

7. Left copy:

$$\mathtt{T}(x,a,y),\, \mathtt{L}(y,y') \rightarrow \exists x'\ \mathtt{L}(x,x'),\, \mathtt{T}(x',a,y').$$

   Here a is a constant.

8. Right copy:

$$\mathtt{T}(x,a,y),\, \mathtt{R}(x,x') \rightarrow \exists y'\ \mathtt{T}(x',a,y'),\, \mathtt{R}(y,y').$$

   Here a is a constant.

The state transition t is transformed to α_t in the same way like in bullet three above. It is crucial to the proof that every state transition δ in M is represented as a single TGD $\mathtt{A}_\delta(x) \rightarrow \mathtt{B}_\delta(x)$. The constraint for the initial configuration fires exactly once. The computation of the chase with this set of constraint can be understood as a grid and each row in the grid represents a configuration of the Turing machine. It can be shown that (M,t) is a yes-instance if and only if (Σ_M, α_t) is a yes-instance. This concludes the proof. ☐

This result prevents us from computing the minimal set of constraints that may fire when chasing I. Still, we can give sufficient conditions that guarantee (I,Σ)-irrelevance for a constraint. For this purpose, we use the chase graph.
We fix some notation. Let I be a fixed finite database instance,

$$\alpha_I := \exists \overline{x} \bigwedge_{R(\overline{x}') \in I} R(\overline{x}')$$

and Σ a set of constraints which contains no constraints with an empty body. We compute the c-chase graph $G_c(\Sigma \cup \{\alpha_I\})$ and denote by $\mathrm{WCC}_\Sigma(\alpha_I)$ the set of constraints that are reachable from α_I in the c-chase graph.

Proposition 103. If $\alpha \notin \mathrm{WCC}_\Sigma(\alpha_I)$, then α is (I,Σ)-irrelevant.

Proof Sketch. Assume that α is not (I,Σ)-irrelevant. Then, there is a chase sequence $I \xrightarrow{\alpha_1,\overline{a_1}} I_1 \xrightarrow{\alpha_2,\overline{a_2}} \cdots \xrightarrow{\alpha_r,\overline{a_r}} I_r \xrightarrow{\beta,\overline{a}} \ldots$. If $\alpha_I \prec_c \beta$, we are finished. Otherwise, there must be some $n_r \in [r]$ such that $\alpha_{n_r} \prec_c \beta$ (otherwise β could not fire). If $\alpha_I \prec_c \alpha_{n_r}$, we are finished. Otherwise, there must be some $n_{r-1} \in [n_r - 1]$ such that $\alpha_{n_{r-1}} \prec_c \alpha_{n_r}$ (otherwise α_{n_r} could not fire). After some finite amount of iterations of this process we have that $\alpha_I \prec_c \alpha_{n_1} \prec_c \ldots \prec_c \alpha_{n_r} \prec_c \beta$. Therefore, the c-chase graph contains a directed path from α_I to β. □

Please note that the prerequisite that every constraint must have a non-empty body is not a restriction. If this is not the case, let every constraint with an empty body fire. Eliminate these constraint from the constraint set and apply the proposition with the obtained instance. Proposition 103 together with Lemma 101 gives us a sufficient data-dependent condition for chase termination, as illustrated in the following example.

Example 104. Consider constraint set Σ from Figure 8.1 and q_2 from the beginning of this section. Please note that Σ is neither in $\mathrm{CT}_{\forall\forall}$ nor in $\mathrm{CT}_{\forall\exists}$ because of α_3. Therefore, data-independent techniques do not apply in this scenario. We set

$$\begin{aligned}\alpha_I := \exists c_1, x_1, x_2, y_1, y_2\ & \texttt{rail}(c_1, x_1, y_1), \texttt{fly}(x_1, x_2, y_2),\\ & \texttt{fly}(x_2, x_1, y_2), \texttt{rail}(x_1, c_1, y_1)\end{aligned}$$

and compute the c-chase graph

$$G(\Sigma \cup \{\alpha_I\}) := (\Sigma \cup \{\alpha_I\}, \{(\alpha_I, \alpha_1), (\alpha_3, \alpha_3)\}).$$

By Proposition 103, α_2 and α_3 are (I, Σ)-irrelevant. It holds that $\Sigma \setminus \{\alpha_2, \alpha_3\} = \{\alpha_1\}$ is inductively restricted, so we know from Lemma 101 that the chase of q_2 with Σ terminates. A similar argumentation holds for q_2'' and q_2''' from the beginning of Section 8.

8.4 Results for $\mathrm{CT}_{I,\exists}$

We continue our journey through chase termination with sufficient termination conditions for a given database instance and at least one terminating chase sequence.
We use the same notation as in Proposition 103.

Corollary 105. Let $k \geq 2$. If $\mathrm{WCC}_\Sigma(\alpha_I) \in \forall\exists\text{-T}[k]$, then there is a terminating chase sequence for Σ and I.

Proof Sketch. Assume that all chase sequences for Σ and I do not terminate. As $\Sigma\setminus\mathrm{WCC}_\Sigma(\alpha_I)$ contains no (Σ, I)-irrelevant constraints, we know that the chase with (Σ, I) has a terminating chase sequence iff the chase with $(\mathrm{WCC}_\Sigma(\alpha_I), I)$ has a terminating chase sequence. As we have $\mathrm{WCC}_\Sigma(\alpha_I) \in \forall\exists\text{-T}[k]$, we can conclude

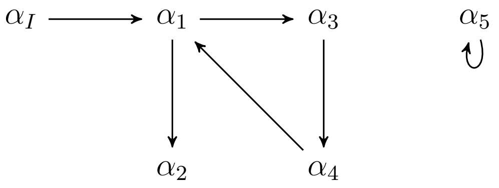


Figure 8.3: C-chase graph for Example 106.

that there is a terminating chase sequence for $(\mathrm{WCC}_\Sigma(\alpha_I), I)$. ☐

We close this section with an example for a scenario in which neither data-independent techniques nor techniques related to $\mathrm{CT}_{I,\forall}$ apply.

Example 106. Let the database instance $\{\mathtt{R}(a)\}$ be given. We consider the constraint set from Example 31 again. We repeat it here as it is from the beginning of Chapter 6 and add a new constraint α_5 to it. Thus, we have the set of TGDs $\Sigma = \{\alpha_1, ..., \alpha_4\}$, where

$$\begin{array}{lcl}
\alpha_1 & := & \mathtt{R}(x_1) \rightarrow \mathtt{S}(x_1, x_1), \\
\alpha_2 & := & \mathtt{S}(x_1, x_2) \rightarrow \exists z\ \mathtt{T}(x_2, z), \\
\alpha_3 & := & \mathtt{S}(x_1, x_2) \rightarrow \mathtt{T}(x_1, x_2),\ \mathtt{T}(x_2, x_1), \\
\alpha_4 & := & \mathtt{T}(x_1, x_2),\ \mathtt{T}(x_1, x_3),\ \mathtt{T}(x_3, x_1) \rightarrow \mathtt{R}(x_2), \text{ and} \\
\alpha_5 & := & \mathtt{E}(x_1, x_2) \rightarrow \exists z\ \mathtt{E}(x_2, z).
\end{array}$$

Please note that Σ is not in

- $\mathrm{CT}_{\forall\forall}$ because $\{\alpha_1, ..., \alpha_4\}$ is not,
- $\mathrm{CT}_{\forall\exists}$ because α_5 is not, and
- $\mathrm{CT}_{I,\forall}$ because we have a non-terminating chase sequence (see Example 86).

Therefore, all data-independent or data-dependent techniques developed before this section do not apply in this scenario.
We set $\alpha_I := \mathtt{R}(a)$ and compute the c-chase graph $G(\Sigma \cup \{\alpha_I\})$ which is depicted in Figure 8.3. We see that $\mathrm{WCC}_\Sigma(\alpha_I) = \{\alpha_1, ..., \alpha_4\}$ which is stratified by Example 86 and therefore in $\forall\exists$-T[2]. Hence, by Corollary 105 there is a terminating chase sequence.

8.5 Stop the Chase: Monitoring

If the previous data-dependent termination conditions do not apply, we propose to monitor the chase run and abort if tuples that may potentially lead to non-termination are created. We introduce a data structure called *monitor graph* that allows us to track the chase run.

Definition 107. A *monitor graph* is a tuple (V, E), where $V \subseteq \Delta_{null} \times 2^{\mathrm{pos}(\Sigma)}$ and $E \subseteq V \times \Sigma \times 2^{\mathrm{pos}(\Sigma)} \times V$.

A node in a monitor graph is a tuple (n, Π), where n is a database value and Π the positions in which n was first created (e.g. as null value with the help of some TGD). An edge $(n_1, \Pi_1, \varphi_i, \Pi, n_2, \Pi_2)$ between (n_1, Π_1), (n_2, Π_2) is labeled with the constraint φ_i that created n_2 and the set of positions Π from the body of φ_i in which n_1 occurred when n_2 was created. The monitor graph is successively constructed while running the chase according to the following definition.

Definition 108. The monitor graph $G_{\mathcal{S}}$ with respect to $\mathcal{S} = I_0 \xrightarrow{\varphi_0, \bar{a}_0} \ldots \xrightarrow{\varphi_{r-1}, \bar{a}_{r-1}} I_r$ is a monitor graph that is inductively defined as follows.

- $G_0 = (\emptyset, \emptyset)$ is the empty chase segment graph.
- If $i < r$ and φ_i is an EGD then $G_{i+1} := G_i$.
- If $i < r$ and φ_i is a TGD then G_{i+1} is obtained from $G_i = (V_i, E_i)$ as follows.

  If the chase step $I_i \xrightarrow{\varphi_i, \bar{a}_i} I_{i+1}$ does not introduce any new null values, then $G_{i+1} := G_i$.

  Otherwise, V_{i+1} is set as the union of V_i and all pairs (n, π), where n is a newly introduced null value and π the set of positions in which n occurs. $E_{i+1} := E_i \cup \{(n_1, \Pi_1, \varphi_i, \Pi, n_2, \Pi_2) \mid (n_1, \Pi_1) \in V_i, (n_2, \Pi_2) \in V_{i+1} \backslash V_i$ and Π is the set of positions in $\mathit{body}(\varphi_i)(\bar{a}_i)$ where n_1 occurs$\}$.

Our next task is to define a necessary criterion for non-termination on top of the monitor graph. To this end, we introduce the notion of *k-cyclicity.*

Definition 109. Let $G = (V, E)$ be a monitor graph and $k \in \mathbb{N}$. G is called k-cyclic if and only if there are pairwise distinct $e_1, ..., e_k \in E$ such that

- there is a path in G that contains $e_1, ..., e_k$ and e_i occurs before e_{i+1} in that path and

- for all $i \in [k-1]$: $p_{2,3,4,6}(e_i) = p_{2,3,4,6}(e_{i+1})$.

We call a chase sequence *k-cyclic* if its monitor graph is k-cyclic. A chase sequence may potentially be infinite if some finite prefix is k-cyclic, for any $k \geq 1$:

Lemma 110. For all $k \in \mathbb{N}$ and every infinite chase sequence $\mathcal{S}$ when chasing I_0 with Σ, there is some finite prefix of $\mathcal{S}$ that is k-cyclic.

Proof. Assume that

- we have an infinite chase sequence $\mathcal{S} = (I_i)_{i \in \mathbb{N}}$ and
- there is some $k \in \mathbb{N}$ such that every finite prefix of $\mathcal{S}$ is not k-cyclic.

Let $(\mathcal{S}_i)_{i \in \mathbb{N}}$ be the sequence of finite prefixes of $\mathcal{S}$ (such that $\mathcal{S}_i$ is a chase sequence of length i) and let $(G_{\mathcal{S}_i})_{i \in \mathbb{N}}$ the respective sequence of monitor graphs. A path in a monitor graph is a finite sequence of edges $e_1, ..., e_l$ (and not of nodes) such that $p_{5,6}(e_i) = p_{1,2}(e_{i+1})$ for $i \in [l-1]$.
We define the notion of *depth* of a node in a monitor graph. Let v be a node in $G_{\mathcal{S}_i}$ and $pred(v)$ the set of predecessors of v. In case v has no predecessors, the depth of v, $depth_{G_{\mathcal{S}_i}}(v)$, is defined as zero. In case v has predecessors, then $depth_{G_{\mathcal{S}_i}}(v) := 1 + max\{\ depth_{G_{\mathcal{S}_i}}(w) \mid w \in pred(v)\ \}$.

The following claim follows immediately from the definition of the monitor graph.

Proposition 111. Let v be a node in $G_{\mathcal{S}_i}$ and $j > i$.

- $G_{\mathcal{S}_i}$ is an acyclic labeled tree.
- Every null value that appears in I_i appears in some first position of a node in $G_{\mathcal{S}_i}$.
- There is a homomorphism h_{ij} from $G_{\mathcal{S}_i}$ to $G_{\mathcal{S}_j}$ such that $depth_{G_{\mathcal{S}_i}}(v) \leq depth_{G_{\mathcal{S}_j}}(h_{ij}(v))$.
- If $I_i \overset{\varphi_i, \overline{a}_i}{\rightarrow} I_{i+1}$, $b \in \overline{a}_i$ is a null value and c a null value that was newly created in this step, then the depth of any node in $G_{\mathcal{S}_{i+1}}$ in which b appears is strictly smaller than the depth of any node in $G_{\mathcal{S}_{i+1}}$ in which c appears. (Proof by induction on i) □

The next proposition is the most important step in the proof of this lemma and follows directly from bullet four in Proposition 111.

Proposition 112. Let $i \in \mathbb{N}$. For every $d \in \mathbb{N} \cup \{0\}$ there is a number $k_d \in \mathbb{N}$ such that for every $i \in \mathbb{N}$ it holds that $|\{\ v \mid \mathit{depth}_{G_{\mathcal{S}_i}}(v) \leq d\ \}| \leq k_d$. *Note that* k_d *is independent from* i. (Proof by induction on d)

We observe another fact.

Proposition 113. There is some $p_k \in \mathbb{N}$ such that if some $G_{\mathcal{S}_i}$ has a path of length p_k, then $\mathcal{S}_i$ is k-cyclic.

This is because we have only a bounded number of relational symbols and constraints available. The remaining step in the proof is to show that if we choose i large enough, then $G_{\mathcal{S}_i}$ contains a path of length p_k. Assume that this claim does not hold. By Proposition 112, the number of nodes of a certain depth is bounded (independent of i). So, if for any i there would be no path of length p_k in $G_{\mathcal{S}_i}$, then the number of nodes in $G_{\mathcal{S}_i}$ would be bounded (independent of i). This implies that the chase has introduced only a bounded number of fresh null values, which contradicts the assumption of an infinite chase sequence. ☐

To avoid non-termination, an application can fix a cycle-depth k and stop the chase when this limit is exceeded. For every terminating chase sequence there is a k s.t. the sequence is not k-cyclic, so if k is chosen large enough, the chase will succeed. We argue that k-cyclicity is a *natural* condition that considers only situations which may cause non-termination, so our approach is preferable to blindly chasing the instance and stopping after a fixed amount of time or number of chase steps. As justified by the following proposition, the choice of k follows a pay-as-you-go principle: for larger k-values the chase will succeed in more cases as the next proposition demonstrates.

Proposition 114. For each $k \in \mathbb{N}$ there is some Σ_k and I_k such that

- both Σ_k and the subset of constraints in Σ_k that are not (I_k, Σ_k)-irrelevant are not inductively restricted, and
- every chase sequence for I_k with Σ_k is $(k-1)$-, but not k-cyclic.

Proof. We set

$I_k := \{\mathtt{S}(c_1), ..., \mathtt{S}(c_k), \mathtt{R}_k(c_1, ..., c_k)\}$ and
$\Sigma_k := \{\varphi\}$,

where $\varphi := \mathtt{S}(x_k), \mathtt{R}_k(x_1, ..., x_k) \rightarrow \exists y\ \mathtt{R}_k(y, x_1, ..., x_{k-1})$.

First observe that Σ_k contains no (I, Σ_k)-irrelevant constraints, so the subset of the constraints in Σ_k that is not (I, Σ)-irrelevant equals to Σ_k. It is easy to verify that Σ_k is not inductively restricted, although the chase with Σ_k always terminates, independently of the underlying data instance, so the condition in bullet one holds. We now chase of I_k with Σ_k. There is only one possible chase sequence $(J_i)_{0 \leq i \leq k}$, defined as

$J_0 := I_k$, and
for $i \leq k$: $J_i := J_{i-1} \cup \{\mathtt{R}(n_i, ..., n_1, c_1, ..., c_{k-i})\}$,

where $n_1, ..., n_k$ are fresh null values. It holds that $J_k \models \Sigma_k$.
The monitor graph with respect to $(J_i)_{0 \leq i \leq k}$ is (V, E), where

$$V := \{(n_i, \mathtt{R}_k^1) | i \in [k]\}, \text{ and}$$
$$E := \{(n_i, \mathtt{R}_k^1, \varphi, \{\mathtt{R}_k^{j-i}\}, n_j, \mathtt{R}_k^1) | 1 \leq i < j \leq k\}.$$

We observe that the sequence is $(k-1)$-cyclic because

$$(n_1, \mathtt{R}_k^1, \varphi, \{\mathtt{R}_k^1\}, n_2, \mathtt{R}_k^1), ..., (n_{k-1}, \mathtt{R}_k^1, \varphi, \{\mathtt{R}_k^1\}, n_k, \mathtt{R}_k^1)$$

constitute a path in the chase graph that satisfies the conditions of the definition of $(k-1)$-cyclicity. The chase sequence is not k-cyclic because there is no path of length at least k in the monitor graph. This proves the second bullet of the proposition. □

Chapter 9

Applications

> **Riccardo:** "What about negation and disjunction?"
> **Sergio:** "Then, we need theory again."
> **Alice:** "Let's have a look at some extensions."

As we have seen in Chapter 4 the chase has various applications. The goal of this chapter is to demonstrate that the chase has even more applications than previously known in the literature. In Section 9.1 we develop a unified approach to typed-based and semantic query optimization techniques showing that both optimization tasks can be addressed using a single, logic-based framework, which seamlessly incorporates typing knowledge into the semantic optimization process. Section 9.2 is devoted to minimization of RDF graphs in the presence of rules allowing a smaller representation of data. Both sections have in common that they use the chase as an important tool in the respective optimization/minimization process and therefore depend on its termination properties. This puts all results on termination in this thesis directly to use in these scenarios. The presentation of both sections is taken from the respective papers [Meier et al., 2010] and [Meier, 2008].

9.1 Semantic Query Optimization in the Presence of Types

Typing is a central component of many practical database systems, including (but not limited to) relational databases, object-oriented database models [Kifer et al., 1995; Papakonstantinou et al., 1995], typed datalog [Zook et al., 2009], and semi-structured data [Milo and Suciu, 1999]. In response, to date a rich theory of type-based optimization has been developed [Frühwirth et al., 1991; Litwin and Risch, 1992; Levy and Suciu, 1997; Gallagher and Puebla, 2002; Henriksson and Maluszynski, 2004; Bruynooghe et al., 2005]. These optimization approaches often use type inference algorithms and have a background in the world of programming languages (cf. [de Moor et al., 2008; Schäfer and de Moor, 2010]).

From a logical point of view, types restrict the state space of the database and therefore can be understood as constraints that each valid database instance must satisfy. In this regard, type-based query optimization is quite similar to semantic query optimization (SQO), where general constraints that are known to hold on the database instance (also called data dependencies) are used to find equivalent query rewritings, with the goal to obtain more efficient evaluation plans [Johnson and Klug, 1982; Chakravarthy et al., 1990; Popa and Tannen, 1999; Deutsch et al., 2006]. Beyond query optimization, constraint-based rewriting has been successfully applied in many other database areas, such as query rewriting using views [Halevy, 2001], data exchange [Fagin et al., 2005], peer data exchange [Fuxman et al., 2005], data integration [Lenzerini, 2002], and probabilistic databases [Olteanu et al., 2009].

Due to their close connection, it is natural to integrate both typing knowledge and integrity constraints, such as tuple generating dependencies (TGDs) and equality generating dependencies (EGDs) [Fagin, 1982; Beeri and Vardi, 1984], into a single logical framework [Buneman et al., 2003; Fan and Libkin, 2002]. In such a framework, one can fall back on established techniques, such as the classical chase algorithm [Maier et al., 1979; Johnson and Klug, 1982; Beeri and Vardi, 1984], to deploy a uniform optimization process. To this aim, several authors have studied encoding the typing knowledge, mainly with the following two formalisms: Datalog (and extensions) [Chan, 1992; Dong and Su, 1996] and Description Logics (DL) [Calvanese et al., 2007, 2008]. However, the lack of value creation (which can be captured by TGDs) in Datalog prevents it from being a suitable candidate for modeling integrity constraints. Although some DLs can capture TGDs, they are unable to express EGDs. There have also been attempts to integrate EGDs in DLs [Toman and Weddell, 2005] but very strong syntactic restrictions are required to obtain decidability of reasoning. Recently, Calì et al. [Calì et al., 2009] have proposed *Datalog*$^{\pm}$, to extend plain Datalog with guarded TGDs and stratified negation. Yet, *Datalog*$^{\pm}$ can not express disjunctive rules. In order to express typing knowledge, integrity constraints containing negation and disjunction are required. To the best of our knowledge, a framework targeted at query optimization has not been investigated before.
As a motivating example, let us consider query optimization in the context of an employee database, where the following typed relations are given (types are prefixed with @):

`Person`(id : *@employee*, gender : *@gender*)
`UpperFloor`(id : *@ceo*, room : *@int*)

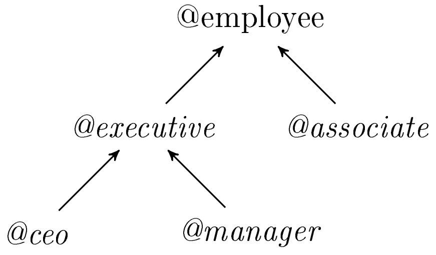


Figure 9.1: Example type hierarchy.

$\texttt{MiddleFloor}(\text{id}: \textit{@manager}, \text{room}: \textit{@int})$
$\texttt{LowerFloor}(\text{id}: \textit{@associate}, \text{room}: \textit{@int})$

Further assume that, in the style of an object-oriented database, the type hierarchy from Figure 9.1 is given. For short, *@executive* and *@associate* are subtypes of *@employee*, and *@executive* splits up into *@ceo* and *@manager*. Further assume that

- *@associate*, *@executive* partition @employee, and
- *@ceo*, *@manager* partition *@executive*.

We represent types as unary relations over the domain of the database, identified by a leading "@" symbol, e.g. write *@employee(x)* to denote that x is of type *@employee*. Suppose that, in coexistence with the restrictions imposed by the type system itself, the two constraints

$\alpha_1 := \texttt{Person}(x, y), \textit{@ceo}(x) \rightarrow \exists z\ \texttt{UpperFloor}(x, z),$
$\alpha_2 := \texttt{Person}(x, y), \textit{@manager}(x) \rightarrow \exists z\ \texttt{MiddleFloor}(x, z)$

are given, enforcing that all CEOs are sitting in the upper floor and all managers are sitting in the middle floor. Having described the setting, we now turn towards query optimization. Consider the following two conjunctive queries

$q_1: \ ans(x) \leftarrow \texttt{Person}(x, y), \texttt{UpperFloor}(x, z),$
$q_2: \ ans(x) \leftarrow \texttt{Person}(x, y), \texttt{MiddleFloor}(x, z)$

and assume that we are interested in computing the union $Q_{1 \vee 2} := q_1 \vee q_2$, i.e. all persons sitting in the middle and upper floor. It is easy to see that, when given *only* the data dependencies α_1, α_2 or *only* the type information as input, $Q_{1 \vee 2}$ is minimal with respect to the number of atoms and unions in the query. Yet, when combining both typing and constraint knowledge, we can derive that $Q_{1 \vee 2}$ is equivalent (on each database instance satisfying the constraints and type restrictions) to the simpler query

$$q'_{1\vee 2}: \quad ans(x) \leftarrow \texttt{Person}(x,y), @executive(x).$$

To see why, consider q_1 and first observe that $@ceo(x)$ must hold, due to the type restriction in the first position of relation `UpperFloor`; hence, we can add the literal $@ceo(x)$ to the body of q_1. But then α_1 implies that there is an entry in relation `UpperFloor` that contains x in its first position, so q_1 is equivalent to

$$q'_1 : ans(x) \leftarrow \texttt{Person}(x,y), @ceo(x).$$

With similar argumentation, we obtain that q_2 is equivalent to

$$q'_2 : ans(x) \leftarrow \texttt{Person}(x,y), @manager(x).$$

Given these two rewritings and the type constraint that *@executive* is exactly the union of *@ceo* and *@manager*, we conclude that $Q_{1\vee 2}$ is equivalent to $q'_{1\vee 2}$ above. Another possible rewriting for $Q_{1\vee 2}$ is the conjunctive query with (safe) negation

$$q''_{1\vee 2} : ans(x) \leftarrow \texttt{Person}(x,y), \neg @associate(x),$$

because *@executive* and *@associate* partition *@employee.*
The previous example does not only show that queries may exhibit non-trivial rewritings in the presence of constraints and types, but also demonstrates that a framework that exploits data dependencies and type information at the same time may give us better optimization results than a sequential, isolated application of these information.

We implement our combined optimization approach in a logic-based framework, where we encode both the data dependencies and the type restrictions in first-order logic. To give an example, for our type hierarchy from before we use constraints like

$$\beta_1 := @executive(x) \rightarrow @employee(x)$$

to fix the subtype relations, and may use the constraint

$$\beta_2 := @executive(x) \wedge @associate(x) \rightarrow \neg @associate(x)$$

to enforce that *@associate* and *@executive* are disjoint. Following the de-facto standard approach, we then use the classical chase algorithm for the optimization process. In particular, we use a variation of the Chase & Backchase algorithm (C&B) [Popa, 2000; Deutsch et al., 2006], an extension of the chase developed to enumerate minimal queries in the presence of constraints.

While straightforward by idea, our approach brings along many new technical challenges, mainly due to the fact that the encoding of non-trivial type systems involves constraints containing disjunction and negation (cf. constraint β_2 above). Previous work on semantic query optimization, though, has mainly focused on tuple-generating and equality-generating dependencies, which contain neither negation nor disjunction. Here, we consider TGDs with disjunction and negation (denoted as $\text{TGD}^{\vee,\neg}$) and EGDs containing disjunction ($\text{EGD}^{\vee}$). While the chase algorithm can easily be extended to these constraint classes (cf. [Deutsch et al., 2007]), to date only few is known about its properties in that setting. Such properties are the central topic in this section.

Structure. The remainder of this section is structured as follows. We start with the preliminaries in the following subsection. In Subsection 9.1.2 we introduce our first-order logic based framework to semantic query optimization in the presence of types, before presenting central results in Subsection 9.1.3. Next, we investigate the complexity of related decision problems in Subsection 9.1.4. We then turn towards an investigation of chase termination in the presence of disjunction and negation in Subsection 9.1.5.

9.1.1 Additional Preliminaries

An this section we are going to generalize many definitions that we introduced earlier in Chapters 2 and 3 such that they can be used in our new scenario. Among these generalized notions are homomorphisms, conjunctive queries, constraints and the chase itself.

Databases. Additionally to our usual database schema $\mathcal{R}$, we have a set $\mathcal{T}$ of unary relational symbols that represent types for our schema. Database instances may include type symbols. In the rest of the paper, we assume the database schema, the type symbols and the set of constants and labeled nulls to be fixed.

Homomorphisms. As usual, a homomorphism from a set of literals A_1 to a set of literals A_2 is a mapping $\mu : \Delta \cup V \rightarrow \Delta \cup \Delta_{null}$ such that the following conditions hold:

- if $c \in \Delta$, then $\mu(c) = c$,
- if $R(c_1, ..., c_n) \in A_1$, then $R(\mu(c_1), ..., \mu(c_n)) \in A_2$, and
- if $\neg R(c_1, ..., c_n) \in A_1$, then $\neg R(\mu(c_1), ..., \mu(c_n)) \in A_2$.

We write $A_1 \to A_2$ to express that there is a homomorphism from A_1 to A_2.

Conjunctive queries (with union and negation). A conjunctive query with negation ($\textsc{Cq}^\neg$) is an expression of the form

$$ans(\overline{x}) \leftarrow \varphi(\overline{x}, \overline{z}),$$

where φ is a conjunction of relational $\mathcal{R}$-literals, $\overline{x}$, $\overline{z}$ are sequences of variables and constants, and it holds that every variable in $\overline{x}$ also occurs in φ. We restrict our discussion to safe queries, i.e. every $\textsc{Cq}^\neg$ has the property that every variable that occurs in the query also occurs in some positive $\mathcal{R}$-atom. This is an easy syntactic restriction that ensures domain-independence. Whenever we speak of a query in this paper, we tacitly assume that it fulfills this safety condition. If a $\textsc{Cq}^\neg$ contains no negation we call it a conjunctive query $\textsc{Cq}$. If $q \in \textsc{Cq}^\neg$, then $db(q)$ is the database that consists of one tuple for each positive atom in q, where each variable x has been replaced by constant c_x.

A union of conjunctive queries with negation ($\textsc{Ucq}^\neg$) is an expression of the form

$$\bigvee_{i \in [n]} q_i,$$

where all q_i are conjunctive queries with safe negation and all head predicates have the same arity. If a $\textsc{Ucq}^\neg$ contains no negation we call it a union of conjunctive queries $\textsc{Ucq}$. For $Q \in \textsc{Ucq}^\neg$, we set $db(Q) := \{db(q) \mid q \in Q\}$.
The semantics of evaluating a conjunctive query with negation q on a database instance I is defined as

$$q(I) := \{\overline{a} \in \Delta^{|\overline{x}|} | I \models \exists \overline{z} \varphi(\overline{a}, \overline{z})\}.$$

Extending the previous definition, the semantics of a union of conjunctive queries with negation $Q := \bigvee_{i \in [n]} q_i$ is defined as

$$Q(I) := \bigcup_{i \in [n]} q_i(I).$$

If $Q, Q' \in \textsc{Ucq}^\neg$ we say that Q is contained in Q' ($Q \sqsubseteq Q'$) iff for all databases I it holds that $Q(I) \subseteq Q'(I)$.
Q and Q' are equivalent, $Q \equiv Q'$, iff $Q \sqsubseteq Q'$ and $Q' \sqsubseteq Q$.
Given a set of first-order sentences Σ, we say that Q is contained in Q' under Σ iff for all databases I s.t. $I \models \Sigma$ it holds that $Q(I) \subseteq Q'(I)$.
We write $Q \equiv_\Sigma Q'$ iff $Q \sqsubseteq_\Sigma Q'$ and $Q' \sqsubseteq_\Sigma Q$.
By convention, we denote $\textsc{Cq}^\neg$ by lowercase and $\textsc{Ucq}^\neg$ by uppercase letters. We write $q \in Q$ iff q is a disjunct of Q. Abusing notation, we write $q_1 \to q_2$ iff there is a homomorphism from the set of atoms in q_1 (including the head atom) to the

set of atoms in q_2 (also including the head atom). By $Q|_\tau$ we denote the query Q from which all non-τ-literals were dropped.

Constraints. Let $\overline{x}, \overline{y}$ be sequences of variables. We consider two types of database constraints: tuple-generating dependencies with union and negation ($\textsc{Tgd}^{\vee,\neg}$) and equality-generating dependencies with union ($\textsc{Egd}^{\vee}$).

A $\textsc{Tgd}^{\vee,\neg}$ φ is a first-order sentence

$$\forall\overline{x}(\phi(\overline{x}) \rightarrow \bigvee_{i\in[n]} \exists\overline{y}_i\psi_i(\overline{x},\overline{y}_i))$$

such that

- $\phi, \psi_1, ..., \psi_n$ are conjunctions of literals, possibly with constants,
- $\psi_1, .., \psi_n$ are not empty,
- ϕ is possibly empty,
- $\phi, \psi_1, ..., \psi_n$ do not contain equality atoms, and
- for all $i \in [n]$ all variables from $\overline{x}$ that occur in ψ_i must also occur in ϕ.

By φ^i we denote the $\textsc{Tgd}$ $\forall\overline{x}(\phi(\overline{x}) \rightarrow \exists\overline{y}_i\psi_i(\overline{x},\overline{y}_i))$ and set $\widehat{\varphi} := \{\varphi_1, ..., \varphi_n\}$.
We obtain the classes $\textsc{Tgd}^{\neg}$, $\textsc{Tgd}^{\vee}$, and $\textsc{Tgd}$ by disallowing $\vee$, $\neg$, and both $\vee$ and $\neg$.

An $\textsc{Egd}^{\vee}$ φ is a first-order logic sentence of the form

$$\forall\overline{x}(\phi(\overline{x}) \rightarrow \bigvee_{i\in[n]} x_{i,1} = x_{i,2}),$$

where all $x_{i,1}, x_{i,2}$ either occur in ϕ or are constants and ϕ is a non-empty conjunction of equality-free $\mathcal{R}$-atoms, possibly with constants. We obtain the subclass $\textsc{Egd}$ from $\textsc{Egd}^{\vee}$ by disallowing $\vee$ in the conclusion.
By φ^i we denote the $\textsc{Egd}$ $\forall\overline{x}(\phi(\overline{x}) \rightarrow x_{i,1} = x_{i,2})$ and set $\widehat{\varphi} := \{\varphi_1, ..., \varphi_n\}$.
When using the word “constraints” in the following, we always mean the union of the classes $\textsc{Tgd}^{\vee,\neg}$ and $\textsc{Egd}^{\vee}$ if not explicitly stated otherwise. As a notational convenience, we will often omit the $\forall$-quantifier and the respective list of universally quantified variables.
We use the term *body*(α) for a constraint α as the set of atoms in its premise; analogously *head*(α) is the set of sets of all atoms in some disjunct of the constraint's conclusion. By $\Sigma|_\tau$ we denote the set Σ from which all non-τ-literals in the constraint bodies were dropped. If α is a constraint and $\overline{a}$ is a sequence of labeled nulls and constants, then $\alpha(\overline{a})$ is the constraint α without universal quantifiers

but with parameters $\overline{a}$. We shall abuse this notation and say that a labeled null occurs in $\alpha(\overline{a})$, meaning that a labeled null is the parameter for some universally quantified variable in α.

Chase steps. Let $q \in \textsc{Cq}^{\neg}$and $\alpha \in \textsc{Tgd}^{\vee,\neg}$ of the form $\phi_1(\overline{x}) \rightarrow \bigvee_{i\in[n]} \exists\overline{y}_i\psi_i(\overline{x},\overline{y}_i)$. We say that α is applicable to q if there is a homomorphism μ from $body(\alpha)$ to q and for every $A \in head(\alpha)$ it holds that μ cannot be extended to a homomorphism $\mu' \supseteq \mu$ from A to q. In such a case the chase step

$$q \overset{\alpha,\mu(\overline{x})}{\longrightarrow} q_u$$

is defined as follows. For every $i \in [n]$ we define a homomorphism ν_i as follows:

- ν_i agrees with μ on all universally quantified variables in α,
- for every existentially quantified variable y in ψ_i we choose a "fresh" labeled null $n_{y,i} \in \Delta_{null}\backslash dom(db(q))$ and define $\nu(y) := n_{y,i}$.

We set q_u to be the union of safe conjunctive queries with negation

$$\bigvee_{i\in[n]} db(q) \wedge \nu_i(\psi_i)$$

from which all unsatisfiable disjuncts are removed. In case that this results in the empty query the result of the chase step is False.
A chase step with an Egd is just a standard chase step in case the query is satisfiable, otherwise the result of the chase step is the empty query False.
Finally, we lift chase steps to apply to unions of conjunctive queries. For a union of safe conjunctive queries with negation $q := \bigvee_{i\in[l]} q_i$ we write

$$q \overset{\alpha,\mu(\overline{x})}{\longrightarrow} q'$$

iff there is $i \in [l]$ such that $q_i \overset{\alpha,\mu(\overline{x})}{\longrightarrow} q_u$ is defined and $q' := q_u \vee \bigvee_{j\in[l]\backslash\{i\}} q_j$.

Chase sequences. A chase sequence is defined analogously to standard chase sequences using the chase steps defined before, i.e. it is an exhaustive application of chase steps until no more chase step is applicable. Note that different orders of application of applicable constraints may lead to a different chase result. However, as proved in [Deutsch et al., 2007], two different chase orders lead to homomorphically equivalent results, if these exist. Therefore, we write Q^{Σ} for the result of the chase on $Q \in \textsc{Ucq}^{\neg}$ under constraints Σ. In case that the constraint set involves disjunction, a chase sequence can be represented as a chase tree, as in [Fagin et al., 2005]. The initial tree contains an artificial root node which has every disjunct in the query to be chased as a child node. When a chase step with some leaf l of the tree involves a branching due to disjunctions in the constraint that is applied to l, then we add every new disjunct as a child of l. If a chase step yields the empty query, then we add FALSE as the only child of l. If the chase terminates, this tree is finite. The disjunction of the tree's leaves represents the resulting query.

Chase termination. As termination is a big issue in the theory of the chase, we face even greater challenges in the presence of negation and disjunction. We extend the definitions of $\mathrm{CT}_{\forall\forall}$ and $\mathrm{CT}_{\forall\exists}$ to our broader notion of chase termination.
So far there have been only few works that consider $\mathrm{TGD}^{\vee,\neg}$ and $\mathrm{EGD}^{\vee}$. In [Deutsch et al., 2007] the authors provided an adaption of weak acyclicity to this new setting, which we will describe next.

Definition 115. (see [Deutsch et al., 2007]) Let Σ be as set of $\mathrm{TGD}^{\vee,\neg}$ and EGDs. The chase flow graph $G = (V, E)$ is a a directed graph whose edge labels may be empty or $*$. It is constructed as follows. For every relation $\mathtt{R}$ mentioned in Σ, V contains the nodes $\mathtt{R}^1, \ldots, \mathtt{R}^{ar(\mathtt{R})}$. For every $\mathrm{TGD}^{\vee,\neg}$ $\sigma \in \Sigma$ and every pair of relations $\mathtt{R}$, $\mathtt{S}$ if $\mathtt{R}(\overline{x})$ appears in *body*(σ) and $\mathtt{S}(\overline{y})$ appears in *head*(σ), then

- if $x_i = y_j$, then add an edge from $\mathtt{R}^i$ to $\mathtt{S}^j$, and
- if y_j is existentially quantified, then add an $*$-labeled edge from $\mathtt{R}^i$ to $\mathtt{S}^j$.

We say that Σ has *stratified witness* iff its chase flow graph contains no cycle through a $*$-labeled edge.

This definition is a straightforward adaption of weak acyclicity by simply ignoring negation and disjunction. It was shown in [Deutsch et al., 2007] that stratified witness guarantees the termination for every union of safe conjunctive queries in the sense of $\mathrm{CT}_{\forall\forall}$. To the best of our knowledge, this is the only condition for chase termination with respect to $\mathrm{TGD}^{\vee,\neg}$ in the literature.

In [Fagin et al., 2005] the authors considered a mix of TGDs and $\mathrm{EGD}^{\vee}$. They have shown that whenever we have have a set of TGDs and $\mathrm{EGD}^{\vee}$ and the set of TGDs is weakly acyclic, then the chase with this constraint set terminates in the sense of $\mathrm{CT}_{\forall\forall}$.

Canonical databases and completeness. We define $ans(q)$ to be the tuple in the head of q where again each variable x has been replaced by constant c_x.
We say that q is satisfiable if there is a database instance I such that $q(I) \neq \emptyset$. Notice that q is satisfiable iff it contains no atom that appears positively and negatively in q.

We say that $q \in \text{CQ}^{\neg}$ is complete [Deutsch et al., 2007] iff it is satisfiable and for all $q' \in \text{CQ}^{\neg}$ it holds that $ans(q) \in q'(db(q))$ implies $q' \to q$.
We say that $Q \in \text{UCQ}^{\neg}$ is satisfiable iff there is a database I such that $Q(I) \neq \emptyset$. $Q \in \text{UCQ}^{\neg}$ is complete iff all disjuncts are complete.
It was shown in [Deutsch et al., 2007] that for every $Q \in \text{UCQ}^{\neg}$ we can compute $Q' \in \text{UCQ}^{\neg}$ that is complete and $Q \equiv Q'$. Therefore, we denote Q' by $comp(Q)$. Let $\texttt{ADom} \notin \mathcal{R} \cup \mathcal{T}$. It was shown in [Deutsch et al., 2007] that $comp(Q) = Q^{\Sigma_{\neg}}|_{\mathcal{R} \cup \mathcal{T}}$, where we define $\Sigma_{\neg}$ as the set which contains for every $\texttt{R} \in \mathcal{R} \cup \mathcal{T}$ with $ar(\texttt{R}) = k$ the constraints

$$\texttt{ADom}(x_1),...,\texttt{ADom}(x_k) \to \texttt{R}(x_1, ..., x_k) \vee \neg\, \texttt{R}(x_1, ..., x_k)$$
$$\texttt{R}(x_1, ..., x_k) \to \texttt{ADom}(x_1), \ldots, \texttt{ADom}(x_k).$$

For $q \in \text{CQ}^{\neg}$, we use $size(q)$ as an abbreviation for the number of literals in q. We extend $size(Q)$ to $Q \in \text{UCQ}^{\neg}$ by $size(Q) := \Sigma_{q \in Q} size(q)$.

9.1.2 Constraints and Types

Our approach to combined semantic and type-based optimization relies on a rigorous first-order logic formalization. Given our special vocabulary $\mathcal{T}$ consisting of unary relation symbols, we define for $a \in \Delta$ its associated type interpretation $\text{TYPE}(a)$ which is a set of literals that exactly contains for every $T \in \mathcal{T}$ either $T(a)$ or $\neg T(a)$.
Abusing notation we identify T with the set $\{a \in \Delta \mid T(a) \in \text{TYPE}(a)\}$ and analogously write $\neg T$ instead of $\{a \in \Delta \mid \neg T(a) \in \text{TYPE}(a)\}$. A type hierarchy H is a set of full constraints over the schema $\mathcal{T}$. A type system over $\mathcal{T}$ is a tuple (H, TYPE). We are interested in type systems in which the type hierarchy adheres to the type interpretation according to the following definition:

Definition 116. We say that a type hierarchy H reflects TYPE if H is logically equivalent to the set of constraints obtained as follows: for all $1 \leq k, l \leq |\mathcal{T}|$ and for all $A_1, ..., A_k$, $B_1, ..., B_l \in \{T, \neg T \mid T \in \mathcal{T}\}$,

- if $\emptyset \neq A_1 \cap ... \cap A_k \subseteq B_1 \cup ... \cup B_l$, we have a full constraint of the form $A_1(x), ..., A_k(x) \to B_1(x) \vee ... \vee B_l(x)$,
- if $\emptyset \neq A_1 \cap ... \cap A_k$ is finite, we have an $\text{EGD}^{\vee}$ of the form $A_1(x), ..., A_k(x) \to \bigvee_{a \in A_1 \cap ... \cap A_k} x = a$, and
- if $\emptyset = A_1 \cap ... \cap A_k$, we have a TGD $A_1(x), ..., A_k(x) \to \neg A_1(x)$.

Informally speaking, a type system reflects a type interpretation if

- all subsumption relationships between types can be derived from the type hierarchy,
- whenever a type or the intersection of several types is finite but non-empty, there is a constraint that fixes the domain of the type, and
- whenever two types are disjoint, we can derive this information using the constraints in the type hierarchy.

The following example illustrates the previous definition.

Example 117. We formalize the type system of our motivating example from the Introduction. First, we define the vocabulary $\mathcal{T}_1 := \{\textit{@employee}, \textit{@manager}, \textit{@ceo}, \dots\}$, where we interpret the elements of $\mathcal{T}_1$ as unary relation symbols. We then fix a type interpretation $\textsc{Type}_1$ that reflects the type relationships that were informally discussed in the Introduction. Consider for instance the constant $a_1 :=$ *'CEO1'* standing for a CEO. Its interpretation is defined as

$$\begin{aligned}\textsc{Type}_1(a_1) := \; & \{\textit{@ceo}(a_1), \textit{@executive}(a_1), \textit{@employee}(a_1),\\ & \neg\textit{@manager}(a_1), \neg\textit{@associate}(a_1),\\ & \neg\textit{@gender}(a_1), \neg\textit{@int}(a_1)\},\end{aligned}$$

stating that a_1 is of type *@ceo*, *@executive*, and *@employee*, but not of type *@manager*, *@associate*, and so on. The type hierarchy H_1, which we construct according to the type interpretation induced by Figure 9.1 and Definition 116, is defined as

$$\begin{aligned}H_1 := \; \{ & \beta_1 := \textit{@ceo}(x) \rightarrow \textit{@executive}(x),\\ & \beta_2 := \textit{@manager}(x) \rightarrow \textit{@executive}(x),\\ & \beta_3 := \textit{@executive}(x) \rightarrow \textit{@employee}(x),\\ & \beta_4 := \textit{@associate}(x) \rightarrow \textit{@employee}(x),\\ & \beta_5 := \textit{@gender}(x) \rightarrow x = \textit{'male'} \vee x = \textit{'female'},\\ & \beta_6 := \textit{@manager}(x), \textit{@ceo}(x) \rightarrow \neg\textit{@manager}(x),\\ & \beta_7 := \textit{@executive}(x), \textit{@associate}(x) \rightarrow \neg\textit{@executive}(x),\\ & \beta_8 := \textit{@employee}(x), \textit{@gender}(x) \rightarrow \neg\textit{@employee}(x),\\ & \beta_9 := \textit{@employee}(x), \textit{@int}(x) \rightarrow \neg\textit{@employee}(x), \text{ and}\\ & \beta_{10} := \textit{@gender}(x), \textit{@int}(x) \rightarrow \neg\textit{@gender}(x)\}\end{aligned}$$

and satisfies Definition 116: β_1 – β_4 model all type subsumption relationships (cf. bullet one of the definition). Next, β_5 fixes the domain of type *@gender* (cf. bullet two), which we assume to be the only finite type in our scenario. Finally, constraints β_6 – β_{10} express disjointness between incompatible types (cf. bullet three), which we mentioned in Section 9.1 when we presented our type hierarchy first. Observe that H_1 contains other relationships, for instance the constraint $@\mathit{ceo}(x) \rightarrow @\mathit{employee}(x)$ can be derived from β_1 and β_3; this is in line with Definition 116, which enforces only logical equivalence to a "complete" list of constraints.

Having established a framework to model type hierarchies in first-order logic, we now extend our framework to general data dependencies used in semantic query optimization.

Definition 118. We call a tuple $(\Sigma, H, \textsc{Type})$ a typed relational schema iff

- Σ is a set of integrity constraints over $\mathcal{R} \cup \mathcal{T}$,
- Σ contains no negative $\mathcal{R}$-literals,
- for all $\alpha \in \Sigma$ it holds that every variable in α appears in some $\mathcal{R}$-atom, and
- $(H, \textsc{Type})$ is a type system over $\mathcal{T}$ such that H reflects $\textsc{Type}$.

Note that Σ is defined over $\mathcal{R} \cup \mathcal{T}$, so it may contain both data dependencies in the common sense and dependencies involving types. In particular, we can use Σ to encode type information given by the schema:

Example 119. To capture our example scenario from the Introduction, we define $\Sigma_1 := \{\alpha_1, \alpha_2, \gamma_1, \gamma_2, \gamma_3, \gamma_3\}$, where α_1 and α_2 are the constraints from the Introduction asserting that CEOs are sitting in the upper floor and managers are sitting in the middle floor and γ_1 – γ_4 are defined as

$$\begin{aligned}
&\gamma_1 := \texttt{Person}(x, y) \rightarrow @\mathit{employee}(x), @\mathit{gender}(y),\\
&\gamma_2 := \texttt{UpperFloor}(x, y) \rightarrow @\mathit{ceo}(x), @\mathit{int}(y),\\
&\gamma_3 := \texttt{MiddleFloor}(x, y) \rightarrow @\mathit{manager}(x), @\mathit{int}(y),\\
&\gamma_4 := \texttt{LowerFloor}(x, y) \rightarrow @\mathit{associate}(x), @\mathit{int}(y).
\end{aligned}$$

Using $\textsc{Type}_1$ and H_1 introduced in Example 117, the tuple $\mathcal{S}_1 := (\Sigma_1, H_1, \textsc{Type}_1)$ is a typed relational schema.

Given a typed relational schema $\mathcal{S} := (C_1, C_2, C_3)$, we will use the conventions that $\Sigma(\mathcal{S}) := C_1$, $H(\mathcal{S}) := C_2$, and $\textsc{Type}(\mathcal{S}) := C_3$. The following definition of satisfying database instances is straightforward.

Definition 120. An $\mathcal{R} \cup \mathcal{T}$-database instance I satisfies a typed relational schema $\mathcal{S}$, $I \models \mathcal{S}$, iff $I|_{\mathcal{R}} \neq \emptyset$, for every constant $a \in \mathit{dom}(I) \cap \Delta$ we have that $I \models \textsc{Type}(\mathcal{S})(a)$ and $I \models \Sigma(\mathcal{S}) \cup H(\mathcal{S})$.

We are now in the position to define the notions of query containment, equivalence, and minimality in typed schemas.

Definition 121. Let $\mathcal{S}$ be a typed relational schema. For $Q, Q' \in \textsc{Ucq}^{\neg}$, we write $Q \sqsubseteq_{\mathcal{S}} Q'$ iff for all $I \models \mathcal{S}$, we have that $Q(I) \subseteq Q'(I)$. Q, Q' are equivalent under $\mathcal{S}$, $Q \equiv_{\mathcal{S}} Q'$, iff $Q \sqsubseteq_{\mathcal{S}} Q'$ and $Q' \sqsubseteq_{\mathcal{S}} Q$.

We are not aware of a generally accepted notion of minimality of $\textsc{Ucq}^{\neg}$. Therefore, we abstract from a concrete cost measure and use a generic cost function instead.

Definition 122. Let $\mathcal{L} \in \{\textsc{Ucq}^{\neg}, \textsc{Ucq}, \textsc{Cq}^{\neg}, \textsc{Cq}\}$ be a query language.

- A cost function for $\mathcal{L}$ is a polynomial-time computable $c : \mathcal{L} \to \mathbb{N}$ such that $\mathit{size}(Q) \leq c(Q)$ and for every subquery $\mathit{sub} \subseteq Q$ we have that $c(\mathit{sub}) \leq c(Q)$.

  Extending the previous definition, the semantics of a union of conjunctive queries with negation $Q := \bigvee_{i \in [n]} q_i$ is defined as $Q(I) := \bigcup_{i \in [n]} q_i(I)$.
- Given a typed relational schema $\mathcal{S}$ and a query $Q \in \mathcal{L}$, we say that Q is $(\mathcal{L}, c, \mathcal{S})$-minimal iff there is no $Q' \in \mathcal{L}$ such that $Q \equiv_{\mathcal{S}} Q'$ and $c(Q') \leq c(Q)$.
- We say that Q' is an $(\mathcal{L}, c, \mathcal{S})$-rewriting of Q iff $Q' \equiv_{\mathcal{S}} Q$ and $c(Q') < c(Q)$.
- An $(\mathcal{L}', c, \mathcal{S})$-minimal rewriting of Q is a query $Q' \in \mathcal{L}'$ such that Q' is $(\mathcal{L}', c, \mathcal{S})$-minimal and $Q \equiv_{\mathcal{S}} Q'$.

It is easily shown that the generic cost function imposes an upper bound on the size of minimal rewritings:

Proposition 123. Let $Q \in \textsc{Ucq}$. The set of $Q' \in \textsc{Ucq}^{\neg}$ with $c(Q') \leq c(Q)$ is finite and its size can be bounded by

$$\mathit{ar}(\mathcal{R} \cup \mathcal{T}) \cdot c(Q)^3 \cdot (3 \cdot |\mathcal{R} \cup \mathcal{T}|)^{c(Q)} \cdot \mathit{ar}(\mathcal{R} \cup \mathcal{T}) \cdot (c(Q) \cdot \mathit{ar}(\mathcal{R} \cup \mathcal{T}) + |\mathit{dom}(Q)| \\ + |\mathit{dom}(\Sigma(\mathcal{S}) \cup H(\mathcal{S}))|).$$

Thus, the set of $(\textsc{Ucq}^{\neg}, c, \mathcal{S})$-minimal rewritings of Q is also bounded by this number.

Proof. Let $Q' \in \text{UCQ}^{\neg}$ such that $c(Q') < c(Q)$. It holds that $size(Q') < c(Q)$. We see that Q' has at most $c(Q)$ many disjuncts and every disjunct has at most $c(Q)$ many literals in the body. There are at most $(3 \cdot |\mathcal{R} \cup \mathcal{T}|)^{c(Q)}$ possibilities to choose for relational symbols or negated relational symbols. The number of distinct terms in one position is bounded by $(c(Q) \cdot ar(\mathcal{R} \cup \mathcal{T}) + |dom(Q)| + |dom(\Sigma(\mathcal{S}) \cup H(\mathcal{S}))|)$. Therefore, the number of terms in one literal is bounded by $ar(\mathcal{R} \cup \mathcal{T}) \cdot (c(Q) \cdot ar(\mathcal{R} \cup \mathcal{T}) + |dom(Q)| + |dom(\Sigma(\mathcal{S}) \cup H(\mathcal{S}))|)$. The arity of the head is bounded by $ar(\mathcal{R} \cup \mathcal{T}) \cdot c(Q)$ because every constant in it must be already either in Q or $\mathcal{S}$. □

We conclude this section with an example that illustrates the cost function, rewritings, and minimality.

Example 124. Let $Q := q_1 \vee \cdots \vee q_n$ be a $\text{UCQ}^{\neg}$. Let $pos(q_i)$ denote the number of positive literals and $neg(q_i)$ the number of negative literals in the body of q_i. We exemplarily consider the cost function

$$c_1(Q) := \textstyle\sum_{1 \leq i \leq n} (pos(q_i) + 2 * neg(q_i)).$$

Given query $Q_{1\vee 2}$ from the Introduction and the typed relational schema $\mathcal{S}_1$ from Example 119, we have that both $q'_{1\vee 2}$ and $q''_{1\vee 2}$ from the Introduction are $(\text{UCQ}^{\neg}, c_1, \mathcal{S}_1)$-rewritings of $Q_{1\vee 2}$: it is easily verified that

$$Q_{1\vee 2} \equiv_{\mathcal{S}_1} q'_{1\vee 2} \equiv_{\mathcal{S}_1} q''_{1\vee 2}$$

and we have

$$c_1(Q_{1\vee 2}) = 4 > c_1(q''_{1\vee 2}) = 3 > c_1(q'_{1\vee 2}) = 2.$$

By enumerating all candidate $(\text{UCQ}^{\neg}, c_1, \mathcal{S}_1)$-rewritings of $Q_{1\vee 2}$ it can be shown that $q'_{1\vee 2}$ is $(\text{UCQ}^{\neg}, c_1, \mathcal{S}_1)$-minimal.

9.1.3 Semantic Query Optimization in Typed Relational Schemas

Having presented our framework, the goal of this section is to develop rewriting techniques and to identify fragments for which rewritings (and hence, by Proposition 123, also minimal rewritings) can be computed. Note that, in the general case, query containment for conjunctive queries under TGDs and EGDs is already undecidable[1], so it follows immediately that query containment in our fragment (where we consider extended classes of TGDs and EGDs, as well as CQs with

---

[1] This follows e.g. from Corollary 9 and Theorem 15 in [Calì et al., 2008].

union and negation) is generally undecidable. Therefore, the study of decidable fragments is of high practical interest.

In our effort to provide a mechanism for containment testing, we start with a slight variant of the standard chase algorithm which, after each chase step, adds type information for constants that were introduced during the chase.

Definition 125. Let $Q \in \textsc{Ucq}^{\neg}$. The sequence $(Q_i)_{i \in \mathbb{N}}$ is inductively defined as follows. $Q_1 := Q$. For $i \geq 2$ we set $Q_i := \bigvee_{q \in Q_{i-1}^{\Sigma(\mathcal{S}) \cup H(\mathcal{S})}} q'$, where q' is defined as q to which $\textsc{Type}(a)$ has been added to the body of the query for all $a \in \Delta \cap dom(db(q))$. If there is some $i \in \mathbb{N}$ such that $Q_i = Q_{i+1}$, then $Q^{\mathcal{S}} := Q_i$.

Example 126. Consider $\mathcal{S}_1 := (\Sigma_1, H_1, \textsc{Type}_1)$ from Example 119, query $q : ans() \leftarrow \texttt{Person}(\textit{`CEO1'}, y)$, and assume that $@ceo(\textit{`CEO1'}) \in \textsc{Type}_1(\textit{`CEO1'})$. When chasing q according to Definition 125, we obtain the query $q^{\mathcal{S}}$ defined as

$$
\begin{aligned}
ans() \leftarrow\ & \texttt{Person}(\textit{`CEO1'}, y), \texttt{UpperFloor}(\textit{`CEO1'}, z),\\
& @ceo(\textit{`CEO1'}), \neg @ceo(y), \neg @ceo(z),\\
& @manager(\textit{`CEO1'}), \neg @manager(y), \neg @manager(z), @integer(z)\\
& \ldots
\end{aligned}
$$

where the rest of the query contains some more type information. The interesting thing here is that, by adding the type restriction $@ceo(\textit{`CEO1'})$ according to our modified version of the chase, we obtain precise type information for the constant *'CEO1'* and therefore in subsequent chase steps are able to derive the literal $\textit{UpperFloor}(\textit{`CEO1'}, z)$ (which is implied by the constraint α_1 from the Introduction).

It can be shown that our modified chase terminates whenever the standard chase algorithm terminates:

Proposition 127. Let $\mathcal{S}$ be fixed. If $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ and the chase with $\mathcal{S}$ terminates in polynomial-time data complexity, then $Q^{\mathcal{S}}$ is defined and the mapping $Q \mapsto Q^{\mathcal{S}}$ can be computed in polynomial time.

Proof. Let c be the number of constants in $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$. Note that c is a constant because $\mathcal{S}$ is fixed. Therefore, $Q_{c+1} = Q_{c+2}$ and the overall computation takes only polynomial time in the input query. $\square$

In the following, we show that our modified chase with $\Sigma(\mathcal{S})$ and $H(\mathcal{S})$ indeed gives universal models and therefore always queries that are equivalent under $\mathcal{S}$.

Definition 128. We call a finite set of database instances $\{I_1, ..., I_n\}$ universal for a typed relational schema $\mathcal{S}$ and a set of database instance $\mathcal{J}$ iff

- for all $i \in [n]$: $I_i \models \mathcal{S}$, and
- for every database instance I it holds that if $I \models \mathcal{S}$ and there is some $J \in \mathcal{J}$ such that $J \to I$, then there is $i \in [n]$ such that $I_i \to I$.

Lemma 129. Let $\mathcal{S}$ be a typed relational schema, $Q \in \text{UCQ}^{\neg}$. If $Q^{\mathcal{S}}$ exists, then $db(Q^{\mathcal{S}})$ is universal for $(\mathcal{S}, db(Q))$.

Proof. Let I be a database instance such that $I \models \mathcal{S}$ and there is some $J \in db(Q)$ such that $J \to I$. We prove our claim by induction on the number of iterations i needed to compute $Q^{\mathcal{S}}$ via Definition 125. We know from [*Deutschet al.*, 2008] that $db(Q^{\Sigma(\mathcal{S}) \cup H(\mathcal{S})})$ is a universal model for $db(Q)$ and $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$. Hence, there is some $I' \in db(Q^{\Sigma(\mathcal{S}) \cup H(\mathcal{S})}$ such that $I' \to I$. We can extend this homomorphism to $(I' \cup \bigcup_{a \in \Delta \cap dom(I')} \text{TYPE}(a)) \to I$ because $I \models \mathcal{S}$. $\square$

Lemma 130. Let $Q \in \text{UCQ}^{\neg}$. It holds that $Q \equiv_{\mathcal{S}} Q^{\mathcal{S}}$.

Proof. It is standard to see that $Q^{\mathcal{S}} \sqsubseteq_{\mathcal{S}} Q$. For the opposite direction, we take an instance I such that $I \models \mathcal{S}$. Clearly, $Q(I) \subseteq Q^{\Sigma(\mathcal{S}) \cup H(\mathcal{S})}(I) \subseteq Q_c(I)$, where Q_c is the query $Q^{\Sigma(\mathcal{S}) \cup H(\mathcal{S})}$ to which all $\text{TYPE}(a)$ has been added for all $a \in \Delta \cap dom(db(Q^{\Sigma(\mathcal{S}) \cup H(\mathcal{S})}))$. Using induction, we obtain that $Q(I) \subseteq Q^{\mathcal{S}}(I)$. $\square$

The central idea of our approach now is to use the modified chase from Definition 125 to find minimal rewritings. More precisely, given a query Q, typed relational schema $\mathcal{S}$, and cost function c, we first compute $Q^{\mathcal{S}}$, enumerate all candidate candidate rewritings Q_i (whose number is bounded by Proposition 123), and finally check if $Q \equiv_{\mathcal{S}} Q_i$ holds. Note that, compared to the C&B algorithm from Section 4.2, we lose the property that every minimal rewriting Q' of Q is a subquery of $Q^{\mathcal{S}}$, as witnessed by the following example.

Example 131. Consider the union of the two conjunctive queries

$p_1 : ans(x) \leftarrow \texttt{Person}(x, \textit{‘male’}), \texttt{militaryService}(x)$ and
$p_2 : ans(x) \leftarrow \texttt{Person}(x, \textit{‘female’})$,

extracting all men who have completed their compulsory military service and all women. Let

$$\delta := \texttt{Person}(x, \textit{`male'}) \rightarrow \texttt{militaryService}(x)$$

and consider $\mathcal{S}_1$, c_1 from Example 124. Given that position $\texttt{Person}^2$ is typed with *@gender* and $H(\mathcal{S}_1)$ contains the constraint

$$\beta_5 := \textit{@gender}(x) \rightarrow x = \textit{`male'} \vee x = \textit{`female'},$$

it follows that

$$ans(x) \leftarrow \texttt{Person}(x, y)$$

is a minimal rewriting of $p_1 \vee p_2$ with respect to $(\textsc{Ucq}^{\neg}, c_1, (\{\delta\}, H(\mathcal{S}_1), \textsc{Type}(\mathcal{S}_1)))$.

The only reason that may prevent us from computing (minimal) rewritings in typed schemas is the fact that the chase (for the original query or the rewritten candidate queries) does not necessarily terminate. In the remainder of this section, we therefore investigate decidable fragments.

Fragment I

We first carry over the results on containment testing for conjunctive queries in the presence of negation from [Deutsch et al., 2007] into the context of typed relational schemas. As a side contribution, we show that this approach works for full TGDs only.

Let $\mathcal{S} \cup \Sigma_{\neg}$ denote the typed relational schema in which $\Sigma_{\neg}$ has been added to $\Sigma(\mathcal{S})$. The next lemma transfers Theorem 9 from [Deutsch et al., 2007] into the context of typed relational schema:

Lemma 132. Let $Q, Q' \in \textsc{Ucq}^{\neg}$. If $Q^{\mathcal{S} \cup \Sigma_{\neg}}$ exists, then it holds that $Q \sqsubseteq_{\mathcal{S}} Q'$ iff for every $P \in Q^{\mathcal{S} \cup \Sigma_{\neg}}|_{\mathcal{R} \cup \mathcal{T}}$ there is $P' \in Q'$ such that $P' \rightarrow P$.

Proof. Without loss of generality assume that there is some instance I such that $I \models \mathcal{S}$ and $Q(I) \neq \emptyset$. Otherwise, the claim is trivial.

The statement of this corollary can be reduced to Theorem 9 in [Deutsch et al., 2007]. It states that for $W, W' \in \textsc{Ucq}^{\neg}$ we have that $W \sqsubseteq_{\Sigma} W'$ iff $W^{\Sigma \cup \Sigma_{\neg}} \sqsubseteq W'$. Our reduction is as follows. For every constant c in Q' and in $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and for every $R \in \mathcal{R} \cup \mathcal{T}$ we add the constraint

$$R(c, x_2, ..., x_{ar(\mathtt{R})}) \rightarrow \bigwedge\nolimits_{l \in \textsc{Type}(c)} l$$

$R(x_1, c, x_3, ..., x_{ar(\mathtt{R})}) \to \bigwedge_{l \in \textsc{Type}(c)} l$
$...$
$R(x_1, , ..., x_{ar(\mathtt{R})-1}, c) \to \bigwedge_{l \in \textsc{Type}(c)} l$

to $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and call the resulting constraint set Σ_{new}. We conclude that

- for all database instances I it holds that $I \models \mathcal{S} \cup \Sigma_\neg \iff I \models \Sigma_{new}$, and
- $Q'^{\mathcal{S}\cup\Sigma_\neg}$ is homomorphically equivalent to $Q'^{\Sigma_{new}}$.

The first bullet is obvious. For the second bullet note that $Q'^{\mathcal{S}}$ can be viewed as $Q'^{\Sigma_{new}}$ obtained via a certain chase order. Then, the statement can be obtained from Theorem 9 in [Deutsch et al., 2007]. □

From this lemma we obtain the following theorem, which – in combination with the chase – gives us a query minimization algorithm whenever the chase with $\mathcal{S} \cup \Sigma_\neg$ terminates:

Theorem 133. There is an algorithm that, given $Q \in \textsc{Ucq}^\neg$ such that $Q \mapsto Q^{\mathcal{S}\cup\Sigma_\neg}$ is computable, enumerates exactly all $(\textsc{Ucq}^\neg, c, \mathcal{S})$-minimal rewritings of Q up to isomorphism.

Proof. Let $Q \in \textsc{Ucq}^\neg$ such that $Q \mapsto Q^{\mathcal{S}\cup\Sigma_\neg}$ is computable. The algorithm is as follows:

1. Initialize $M := \emptyset$.
2. Compute $Q^{\mathcal{S}\cup\Sigma_\neg}$.
3. Enumerate all $Q' \in \textsc{Ucq}^\neg$ with $c(Q') \leq c(Q)$:
    a) Compute $Q'^{\mathcal{S}\cup\Sigma_\neg}$.
    b) Test whether $Q'^{\mathcal{S}\cup\Sigma_\neg} \equiv_{\mathcal{S}} Q^{\mathcal{S}\cup\Sigma_\neg}$ holds with the help of Lemma 132 and if so, add Q' to M.
4. Output $\{Q'' \in M \mid \text{for all } Q''' \in M \text{ it holds that } c(Q'') \leq c(Q''')\}$.

By Proposition 123 this algorithm terminates. It follows from Lemma 132 that it is sound and complete for finding $(\textsc{Ucq}^\neg, c, \mathcal{S})$-minimal rewritings of the input query. □

The problem with this result, though, is that the extension of the constraint set by $\Sigma_\neg$ often leads to non-terminating chase sequences, even if the chase with $\mathcal{S}$ terminates.

Example 134. If $\Sigma(\mathcal{S}) := \{\epsilon_1 := \mathtt{R}(x_1, x_2) \rightarrow \exists y\ \mathtt{E}(x_2, y)\}$ and $q() := \mathtt{R}(x_1, x_2)$, then $q^{\mathcal{S} \cup \Sigma_\neg}$ is not defined. To see why, observe that $\Sigma_\neg$ contains (amongst others) the constraints

$$\epsilon_2 := \mathtt{E}(x_1, x_2) \rightarrow \mathtt{ADom}(x_1),\ \mathtt{ADom}(x_2),$$
$$\epsilon_3 := \mathtt{ADom}(x_1),\ \mathtt{ADom}(x_2) \rightarrow \mathtt{R}(x_1, x_2) \vee \neg\ \mathtt{R}(x_1, x_2).$$

It is easily verified that there is no terminating chase sequence for the chase of q with $\{\epsilon_1, \epsilon_2, \epsilon_3\}$. The same still holds when chasing with the full set $\Sigma(\mathcal{S}) \cup \Sigma_\neg$.

In the light of the previous example, there is only little hope that the chase algorithm terminates when Σ contains TGDs with existentially quantified variables. With this observation in mind, the next corollary identifies the only situation where Theorem 133 is of practical benefit:

Corollary 135. Let $\Sigma(\mathcal{S})$ consist of full constraints only. There is an algorithm that, given $\mathcal{S}$ and $Q \in \mathrm{UCQ}^\neg$ as input, enumerates exactly all $(\mathrm{UCQ}^\neg, c, \mathcal{S})$-minimal rewritings of Q up to isomorphism.

Our finding that the above approach (and hence, also the approach proposed in [Deutsch et al., 2007]) is essentially restricted to full constraints motivates the search for fragments in which the minimization problem can be solved without adding $\Sigma_\neg$. We will present such a fragment in the next subsection.

Fragment II

We now define a fragment of $\mathrm{UCQ}^\neg$. We call $Q \in \mathrm{UCQ}^\neg$ semi-positive, denoted as $\mathrm{UCQ}^{\neg/2}$, iff all variables that occur in a negative $\mathcal{R}$-literal occur also in the head predicate. Semi-positive queries are interesting because containment testing using the chase can be done without adding $\Sigma_\neg$:

Theorem 136. Let $Q \in \mathrm{UCQ}^{\neg/2}$ without negative $\mathcal{T}$-literals and $Q' \in \mathrm{UCQ}^\neg$. W.l.o.g. every $q \in Q'$ contains all constants from Q, Q' and Σ. If Q'^Σ exists, then it holds that

$$\begin{aligned} Q' \sqsubseteq_\Sigma Q &\iff \text{for all } P' \in \mathit{comp}(Q')^\Sigma \text{ there exists } P \in Q \text{ such that } P \rightarrow P' \\ &\iff Q'^\Sigma \sqsubseteq Q. \end{aligned}$$

Proof.

- Proof of the first equivalence. For the forward direction assume that $Q' \sqsubseteq_\Sigma Q$ holds. Let $comp(Q')^\Sigma = \bigvee_{i\in[k]} P_i$ and $comp(P_i) = \bigvee_{j\in[l_i]} P_{i,j}$. Note that for every P_i, every $R \in \mathcal{R} \cup \mathcal{T}$ and every tuple $\overline{x}$ of constants in $comp(Q')^\Sigma$ and variables in $ans(P_i)$ we have that either $R(\overline{x})$ or $\neg R(\overline{x})$ is in the body of P_i. We have that $ans(P_i) \in P_i(P_i)$, which implies $ans(P_i) \in Q'^\Sigma(P_i)$. As $db(P_i) \models \Sigma$, it follows that $ans(P_i) \in Q'(P_i)$. Using the assumption, we obtain $ans(P_i) \in Q(P_i)$. Assume $Q = \bigvee_{j\in[l]} P_j$. For some $j \in [l]$ we have that $ans(P_i) \in Q_j(P_i)$. We will show that for all $j' \in [l_i]$ it holds that $ans(P_i) = ans(P_{i,j'}) \in Q_j(P_{i,j'})$. Suppose not. Then, there is $j'' \in [l_i]$ such that $ans(P_i) = ans(P_{i,j''}) \notin Q_j(P_{i,j''})$. Then, for every homomorphism h that maps Q_j^+ to $P_{i,j''}$ there is some atom A such that $\neg A \in Q_j^-$ and $h(A) \in P_{i,j''}$. Let μ be a homomorphism that witnesses $ans(P_i) \in Q_j(P_i)$. Let A be an atom such that $\neg A \in Q_j^-$. We can conclude that $\neg\mu(A) \in P_i^-$ because $Q \in \textsc{Ucq}^{\neg/2}$. Therefore, μ witnesses $ans(P_i) = ans(P_{i,j''}) \in Q_j(P_{i,j''})$, which is a contradiction. We have shown that for all $j' \in [l_i]$ it holds that $Q_j \to P_{i,j'}$, which implies our claim.

  For the backward direction assume that for all $P' \in comp(Q')^\Sigma$ there exists $P \in Q$ such that it holds that $P \to P'$. Hence, $comp(Q')^\Sigma \sqsubseteq Q$. Observe that $Q' \equiv comp(Q') \equiv_\Sigma comp(Q')^\Sigma \sqsubseteq Q$ implies the claim.

- The second equivalence is a corollary from the first one. ☐

As a corollary from the previous theorem we obtain the following result for typed relational schemas, clarifying that the constraints in $H(\mathcal{S})$ do not harm this nice property.

Corollary 137. Let $\Sigma(\mathcal{S}) \subseteq \textsc{Tgd}^{\vee,\neg} \cup \textsc{Egd}^{\vee}$, $Q \in \textsc{Ucq}^{\neg/2}$ without negative $\mathcal{T}$-literals and $Q' \in \textsc{Ucq}^{\neg}$. W.l.o.g. every $q \in Q'$ contains all constants from Q, Q' and Σ. If $Q'^{\mathcal{S}}$ exists, then it holds that

$$\begin{aligned} Q' \sqsubseteq_{\mathcal{S}} Q &\iff \text{for all } P' \in comp(Q')^{\mathcal{S}} \text{ there exists } P \in Q \text{ such that } P \to P' \\ &\iff Q'^{\mathcal{S}} \sqsubseteq Q. \end{aligned}$$

Proof. The statement of this corollary can be reduced to Theorem 136 as follows. For every constant c in Q' and in $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and for every $R \in \mathcal{R} \cup \mathcal{T}$ we add the constraint

$$R(c, x_2, ..., x_{ar(\mathtt{R})}) \to \bigwedge\nolimits_{l\in\textsc{Type}(c)} l$$
$$R(x_1, c, x_3, ..., x_{ar(\mathtt{R})}) \to \bigwedge\nolimits_{l\in\textsc{Type}(c)} l$$

...

$R(x_1, , ..., x_{ar(\mathtt{R})-1}, c) \to \bigwedge_{l \in \textsc{Type}(c)} l$

to $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and call the resulting constraint set Σ_{new}. We conclude that

- for all database instances I it holds that $I \models \mathcal{S} \iff I \models \Sigma_{new}$, and
- $Q'^{\mathcal{S}}$ is homomorphically equivalent to $Q'^{\Sigma_{new}}$.

The first bullet is obvious. For the second bullet note that $Q'^{\mathcal{S}}$ can be viewed as $Q'^{\Sigma_{new}}$ obtained via a certain chase order. Then, the corollary can be obtained from Theorem 136. □

In order to demonstrate why the condition that every $q \in Q'$ contains all constants from Q, Q' and $\Sigma(\mathcal{S})$ is not a restriction we have a look at the next example.

Example 138. Let $q' : ans() \leftarrow A(x, y)$ and $q : ans() \leftarrow A(x, a)$, where we want to test for $q' \sqsubseteq_{\mathcal{S}} q$. The constant a occurs in q but not in q', which contradicts our condition. We rewrite q' to $Q' : q_1 \vee q_2$, where $q_1 : ans() \leftarrow A(x, y), A(a, a)$ and $q_2 : ans() \leftarrow A(x, y), \neg A(a, a)$. Obviously, $q' \equiv Q'$. We can now proceed by testing for $Q' \sqsubseteq_{\mathcal{S}} q$.

Clearly, the above results are applicable in practice whenever $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$ is in $\mathrm{CT}_{\forall\forall}$ or $\mathrm{CT}_{\forall\exists}$. The following lemma gives an even weaker precondition, showing that constraints in $H(\mathcal{S})$ do never affect chase termination:

Lemma 139. For $Q \in \textsc{Ucq}^{\neg}$: $Q \mapsto Q^{\mathcal{S}}$ is computable if

- $\Sigma(\mathcal{S})|_{\mathcal{R}} \in \mathrm{CT}_{\forall\forall}$, or
- $\Sigma(\mathcal{S}) = \Sigma(\mathcal{S})|_{\mathcal{R}} \in \mathrm{CT}_{\forall\exists}$.

Proof.

- We have that $\Sigma(\mathcal{S})|_{\mathcal{R}} \subseteq \mathrm{CT}_{\forall\forall} \implies \Sigma(\mathcal{S})|_{\mathcal{R}} \cup H(\mathcal{S}) \subseteq \mathrm{CT}_{\forall\forall} \implies \Sigma(\mathcal{S}) \cup H(\mathcal{S}) \subseteq \mathrm{CT}_{\forall\forall}$ because every variable that occurs in some $\mathcal{T}$-literal also occurs in some $\mathcal{R}$-atom. Therefore, we can conclude that for every $\overline{Q} \in \textsc{Ucq}^{\neg}$ the mapping $Q \mapsto Q^{\mathcal{S}}$ is computable.
- Let $\Sigma(\mathcal{S}) = \Sigma(\mathcal{S})|_{\mathcal{R}} \subseteq \mathrm{CT}_{\forall\exists}$. We can conclude that $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ because we can first chase with $\Sigma(\mathcal{S})$ according to a strategy that guarantees termination and afterward apply the constraints in $H(\mathcal{S})$. Therefore, we can conclude that for every $\overline{Q} \in \textsc{Ucq}^{\neg}$ the mapping $Q \mapsto Q^{\mathcal{S}}$ is computable. □

As our central result, we obtain an algorithm for $\mathrm{UCQ}^{\neg/2}$ minimization whenever we can guarantee the termination of the underlying chase according to the previous lemma:

Theorem 140. There is an algorithm that, given a query $Q \in \mathrm{UCQ}^{\neg/2}$ such that $Q \mapsto Q^{\mathcal{S}}$ is computable, enumerates exactly all $(\mathrm{UCQ}^{\neg/2}, c, \mathcal{S})$-minimal rewritings of Q under Σ up to isomorphism.

Proof. Without loss of generality, we assume that $H(\mathcal{S})$ contains for every $T \in \mathcal{T}$ some $\overline{T} \in \mathcal{T}$ such that $\overline{T}$ reflects the complement of T, i.e. we have constraints

- $\neg T(x) \rightarrow \overline{T}(x)$,
- $\neg \overline{T}(x) \rightarrow T(x)$,
- $T(x) \rightarrow \neg \overline{T}(x)$,
- $\overline{T}(x) \rightarrow \neg T(x)$, and
- $T(x), \overline{T}(x) \rightarrow \neg T(x)$.

We define a mapping $E : \mathrm{UCQ}^{\neg} \rightarrow \mathrm{UCQ}^{\neg}$ that substitutes all negative $\mathcal{T}$-literals $\neg T(...)$ by $\overline{T}(...)$. This operation preserves equivalence under $\mathcal{S}$.
Let $Q \in \mathrm{UCQ}^{\neg/2}$ such that $Q \mapsto Q^{\mathcal{S}}$ is computable. The algorithm is as follows:

1. Initialize $M := \emptyset$.
2. Compute $comp(Q)^{\mathcal{S}}$.
3. Enumerate all $Q' \in \mathrm{UCQ}^{\neg/2}$ with $c(Q') \leq c(Q)$:
    a) Rewrite Q' to Q'' such that $Q' \equiv Q''$ and every $q \in Q''$ contains all constants from Q, Q'' and $\Sigma(\mathcal{S}) \cup H(\mathcal{S})$
    b) Compute $comp(Q'')^{\mathcal{S}}$.
    c) Test whether $E(Q'') \equiv_{\mathcal{S}} E(Q)$ holds with the help of Corollary 137 and if so, add Q' to M.
4. Output $\{Q'' \in M \mid \text{for all } Q''' \in M \text{ it holds that } c(Q'') \leq c(Q''')\}$.

By Proposition 123 this algorithm terminates. It follows from Corollary 137 that it is sound and complete for finding $(\mathrm{UCQ}^{\neg/2}, c, \mathcal{S})$-minimal rewritings of the input query. $\square$

9.1.4 Complexity

Satisfiability

As a first problem in our complexity study, we investigate the satisfiability of typed relational schemas. We define satisfiability and the associated decision problem as follows.

Definition 141. A typed relational schema $\mathcal{S}$ is satisfiable iff there is a finite $\mathcal{R} \cup \mathcal{T}$-database instance I s.t. $I \models \mathcal{S}$.

SATISFIABILITY:
Input : A typed relational schema $\mathcal{S}$.
Question: Is there a finite instance I: $I \models \mathcal{S}$?
Answer: Yes or no.

It turns out that, whenever there is a terminating chase sequence for the constraints induced by the data dependencies and the type hierarchy, we can use the chase algorithm to test whether a satisfying model exists or not:

Theorem 142. Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$. There is an algorithm that, given a typed relational schema $\mathcal{S}$ as input, decides whether $\mathcal{S}$ is satisfiable.

Proof. Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$. We show that

$$\mathcal{S} \text{ is satisfiable iff } (\bigvee_{\mathtt{R} \in \mathcal{R}} q() \leftarrow R(x_1, ..., x_{ar(\mathtt{R})}))^{\mathcal{S}} \neq \textsc{False}.$$

The backward direction is standard. For the forward direction assume that $\mathcal{S}$ is satisfiable but $(\bigvee_{\mathtt{R} \in \mathcal{R}} q() \leftarrow R(x_1, ..., x_{ar(\mathtt{R})}))^{\mathcal{S}} = \textsc{False}$. Let I be a finite database instance such that $I \models \mathcal{S}$ and $I|_{\mathcal{R}} \neq \emptyset$. It follows from the proof of Lemma 129 that for every intermediate (during the application of the chase) union of queries Q there is $q \in Q$ such that $db(q) \to I$. We can conclude that $(\bigvee_{\mathtt{R} \in \mathcal{R}} q() \leftarrow R(x_1, ..., x_{ar(\mathtt{R})}))^{\mathcal{S}} \neq \textsc{False}$, which is a contradiction. ☐

A reduction from CNF-SAT gives us an NP lower bound:

Theorem 143. SATISFIABILITY is NP-hard. This still holds if the set of integrity constraints $\Sigma(\mathcal{S})$ contains no negation and no $\textsc{Egd}^{\vee}$.

Proof. We prove NP-hardness by reduction from CNF-SAT, the satisfiability problem for propositional formulas in conjunctive normal form.

Let

$\alpha := \bigwedge_{i \in [m]} B_i$

be given, where

$B_i = x_{i,1} \vee \ldots \vee x_{i,k_i} \vee \neg y_{i,1} \vee \ldots \vee \neg y_{i,l_i}$

and the set of propositional variables that occurs in α is $\{x_1, \ldots, x_n\}$.
We encode α in a typed relational schema as follows. Note that we denote strings by surrounding quotation marks.

$$\exists x_1, \ldots, x_n \, (\bigwedge_{i \in [m]}((\bigwedge_{j \in [k_i]} R(\text{`}i\text{'}, \text{`}j\text{'}, \text{`}+\text{'}, \text{`}x_{i,j}\text{'}, x_{i,j})) \wedge (\bigwedge_{j \in [l_i]} R(\text{`}i\text{'}, \text{`}j\text{'}, \text{`}\neg\text{'}, \text{`}y_{i,j}\text{'}, y_{i,j})))),$$

$$(\bigwedge_{j \in [k_i]} R(\text{`}i\text{'}, \text{`}j\text{'}, \text{`}+\text{'}, \text{`}x_{i,j}\text{'}, x_{i,j})) \wedge (\bigwedge_{j \in [l_i]} R(\text{`}i\text{'}, \text{`}j\text{'}, \text{`}\neg\text{'}, \text{`}y_{i,j}\text{'}, y_{i,j})) \\ \rightarrow \bigvee_{j \in [k_i]} L_1(x_{i,j}) \vee \bigvee_{j \in [l_i]} L_0(y_{i,j}), \text{ for all } i \in [m],$$

$R(x_1, x_2, x_3, x_4, x_5) \rightarrow L_0(x_5) \vee L_1(x_5)$,
$L_0(x) \rightarrow \neg L_1(x)$, and
$L_1(x) \rightarrow \neg L_0(x)$.

The last two constraints constitute the type hierarchy $H(\mathcal{S})$ and all other constraints constitute $\Sigma(\mathcal{S})$, which means that the constraints obey the syntactic restrictions in order to form valid typed relational schema. It is standard to verify that α is satisfiable iff $\mathcal{S}$ is satisfiable. ☐

The next theorem identifies a large class of typed relational schemas for which we obtain a Σ_2^P upper bound:

Theorem 144. Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ ensuring polynomial depth of the chase tree. Then SATISFIABILITY $\in \Sigma_2^P$.

Proof. We give a Σ_2^P-algorithm that tests satisfiability, i.e. an NP algorithm with coNP oracle. Initially, guess some predicate $R \in \mathcal{R}$ and consider the query

$q() \leftarrow R(x_1, \ldots, x_{ar(R)})$.

Then compute $q^{\mathcal{S}}$, using the following modification of the algorithm from Definition 125. In each chase step starting with query q_i as input, guess some $\alpha \in \Sigma(\mathcal{S}) \cup H(\mathcal{S})$ and constants $\overline{a}$ for the universally quantified variables in α such that $db(q_i) \models body(\alpha(\overline{a}))$ (testing the latter condition can be done in polynomial time). Then use the coNP oracle to verify that there are no constants $\overline{b}$ for the existentially quantified variables in the head of α such that $db(q_i) \models head(\alpha(\overline{a}, \overline{b}))$. If α is a TGD or EGD with disjunction in the head, i.e. is of the form

$\alpha := \phi \rightarrow \psi_1 \vee \cdots \vee \psi_n$,

guess $i \in [n]$ and perform a chase step with $\alpha' := \phi \rightarrow \psi_i$ instead of α. Otherwise, if the head of constraint α is disjunction-free, simply perform a chase step with α. By assumption, the above algorithm terminates in k steps, where k is polynomially bounded by the depth of the chase tree, so it is easy to see that the algorithm is in Σ_2^P. It is standard to show that $\mathcal{S}$ is satisfiable iff $q_{k+1} \neq$ FALSE (in case $q_i =$ FALSE for some $i < k+1$, we set $q_{k+1} :=$ FALSE). ☐

We obtain even better bounds when restricting the size of the constraint's heads:

Corollary 145. Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ ensuring polynomial depth of the chase tree and assume that the size of the head of every $\alpha \in \Sigma(\mathcal{S}) \cup H(\mathcal{S})$ is bounded by some constant k. Then SATISFIABILITY is in NP.

Proof. As an additional assumption compared to Theorem 144 we have that the size of every constraint's head is bounded. The algorithm that tests satisfiability works similar to the algorithm from the proof of Theorem 144; the only difference is that model checking now can be done in polynomial time (i.e., does not require a coNP oracle), because the size of the constraint's head is fixed. This gives us an NP algorithm. ☐

Observe that the previous (and some of the following) results rely on the polynomial depth of the chase tree. We will come back to this issue in Section 9.1.5, where we indicate chase termination conditions that guarantee this property.

Query Optimization

We now come to the central complexity results of this paper, namely the complexity of testing whether a query is a rewriting or a minimal rewriting of another query. We define these two decision problems as follows.

MINIMAL$(\mathcal{L}, c, \mathcal{S})$:
Input : $Q \in \mathcal{L}$.
Question: Is Q $(\mathcal{L}, c, \mathcal{S})$-minimal?
Answer: Yes or no.

REWRITE$(\mathcal{L}, c, \mathcal{S})$:
Input : $(Q, Q') \in \mathcal{L} \times \mathcal{L}$.
Question: Is Q' a $(\mathcal{L}, c, \mathcal{S})$-rewriting of Q?
Answer: Yes or no.

By definition, Q' is an $(\mathcal{L}, c, \mathcal{S})$-minimal rewriting of Q iff $Q' \in \textsc{Minimal}(\mathcal{L}, c, \mathcal{S})$ and $(Q, Q') \in \textsc{Rewrite}(\mathcal{L}, c, \mathcal{S})$. A reduction from the containment problem for conjunctive queries with inequalities gives us a lower bound for the REWRITE problem in the general case.

Theorem 146. $\textsc{Rewrite}(\textsc{Cq}, c, \mathcal{S})$ is Π_2^P-hard.

Proof. We prove Π_2^P-hardness by a reduction from the containment problem for conjunctive queries with inequalities. In [Kolaitis et al., 1998] it was shown that this problem is Π_2^P-complete even for boolean queries. Thus, we can restrict ourselves to boolean queries which we represent as the set of atoms in the query's body. Without loss of generality, we can assume that q_1 and q_2 have disjoint sets of variables when we want to test whether $q_1 \sqsubseteq q_2$ holds.
In the first step, we reduce the containment problem for such queries to the equivalence problem via the reduction $q_1 \sqsubseteq q_2 \Longleftrightarrow q_1 \equiv q_1 \cup q_2$.
In the second step we reduce the equivalence problem for conjunctive queries with inequalities to $\textsc{Rewrite}(\textsc{Cq}, c, \mathcal{S})$ for the fixed typed relational schema $\mathcal{S}$ with

$$\begin{aligned}\Sigma(\mathcal{S}) = \{\ & \mathtt{I}(x,x) \rightarrow \mathtt{T}(x),\\ & \mathtt{J}(x,y) \rightarrow x = y,\\ & \mathtt{I}(x,y),\ \mathtt{J}(x,y) \rightarrow \mathtt{T}(x),\\ & \mathtt{D}(x_1),\ \mathtt{D}(x_2) \rightarrow \mathtt{I}(x_1,x_2) \vee\ \mathtt{J}(x_1,x_2)\}\\ & \cup\{\mathtt{R}(x_1,...,x_{ar(\mathtt{R})}) \rightarrow \mathtt{D}(x_1),...,\mathtt{D}(x_{ar(\mathtt{R})}) \mid \mathtt{R} \in \mathcal{R}\}\end{aligned}$$

and

$$H(\mathcal{S}) = \{\mathtt{T}(x) \rightarrow \neg\ \mathtt{T}(x)\},$$

where `I`, `J`, `D`, and `T` are fresh relational symbols. Intuitively, J simulates the equality predicate and I its complement with respect to the active domain. Given two conjunctive queries with inequalities q_1, q_2 as input, we translate to q_1' and q_2' by substituting every inequality atom of the form $x_i \neq x_k$ by $I(x_i, x_k)$. We can conclude that $q_1 \equiv q_2 \Longleftrightarrow q_1' \equiv_{\mathcal{S}} q_2'$ holds, which finishes this proof. ☐

Whenever the chase tree is of polynomial depth, we can also guarantee membership in Π_2^P:

Theorem 147. Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ ensuring polynomial depth of the chase tree. Then, $\textsc{Rewrite}(\textsc{Ucq}^{\neg/2}, c, \mathcal{S})$ is Π_2^P-complete.

Proof. Given Theorem 146, it suffices to prove membership in Π_2^P. We give an algorithm which is appropriate for our needs. Given (Q, Q') as input our test essentially consists of an application of Theorem 136 and Corollary 137. W.l.o.g. every $q \in Q'$ contains all constants from Q, Q' and Σ. We test whether for all $P' \in \mathit{comp}(Q'^{\Sigma})$ there is $P \in Q$ such that for all $P'' \in \mathit{comp}(P')$ it holds that $P \rightarrow P''$ and whether for all $P \in \mathit{comp}(Q^{\Sigma})$ there is $P' \in Q'$ such that for all $P'' \in \mathit{comp}(P)$ it holds that $P' \rightarrow P''$. This is doable in Π_2^P-time. $\square$

Having discussed Rewrite, we conclude our complexity study with a similar result for the minimization problem:

Theorem 148. Let $\Sigma(\mathcal{S}) \cup H(\mathcal{S}) \in \mathrm{CT}_{\forall\exists}$ ensure polynomial depth of the chase tree. Then, $\textsc{Minimal}(\textsc{Ucq}^{\neg/2}, c, \mathcal{S}) \in \Pi_3^P$.

Proof. We show that the complement of $\textsc{Minimal}(\textsc{Ucq}, c, \mathcal{S})$ is in Σ_3^P. In order to see this we can use the following algorithm. Given Q as input, we guess Q' such that $c(Q') < c(Q)$ and test whether $(Q, Q') \in \textsc{Rewrite}(\textsc{Ucq}, c, \mathcal{S})$ (see Theorem 147) holds. Clearly, all this is doable in Σ_3^P-time. $\square$

To put our results into context, we point out that containment testing under negation- and disjunction-free Tgds and Egds is already Np-hard. Nevertheless, the experiments in [Popa et al., 2000] confirm that minimization works well in practice, last but not least because the constraint sets are usually small. What we have shown here is that semantic query optimization in the presence of types, $\textsc{Tgd}^{\neg,\vee}$, and $\textsc{Egd}^{\vee}$ falls into the lower levels of the polynomial hierarchy. Therefore, we may expect our methods to work well in practice, too.

9.1.5 Chase Termination: Eliminating Negation and Disjunction

We have seen in Section 9.1.3 that the applicability of our framework for semantic query optimization in the presence of types heavily depends on the termination of the underlying chase algorithm. However, little attention has been spent on chase termination in the presence of negation and disjunction in the constraints. Notable exceptions are [Fagin et al., 2005], which treats the case of Tgds and $\textsc{Egd}^{\vee}$s, and [Deutsch et al., 2007], which uses an adaption of the weak acyclicity condition from [Fagin et al., 2005] to allow for negation and disjunction.

We present a more general approach and eliminate negation and disjunction in the constraints such that, if a termination guarantee for the rewritten constraint set can be made, then a guarantee can be also made for the original constraint set.

This allows us to reduce the problem of chase termination to a standard setting involving only TGDs and EGDs.
This elimination process is defined via the two mappings introduced in the following definition.
Let $\mathcal{T}' = \{\mathtt{T}'_1, ..., \mathtt{T}'_m\}$ and $\mathcal{R}' := \{\mathtt{R}'_1, ..., \mathtt{R}'_n\}$ be two additional sets of relational symbols such that $(\mathcal{R}' \cup \mathcal{T}') \cap (\mathcal{R} \cup \mathcal{T}) = \emptyset = \mathcal{R}' \cap \mathcal{T}'$.

Definition 149. By L_0 we denote the set of first order sentences over vocabulary $\mathcal{R} \cup \mathcal{T}$.

- The mapping $T_\neg : 2^{L_0} \to 2^{L_0}$ is defined as follows. For every $\Phi \subseteq L_0$ we set $T_\neg(\Phi) := \widetilde{\Phi}$, where $\widetilde{\Phi}$ equals Φ except that for every $i \in [n]$ every occurrence of $\neg\mathtt{R}_i$ ($\neg\mathtt{T}_i$) is substituted by $\mathtt{R}'_i$ ($\mathtt{T}'_i$).
- The mapping $T_\vee : 2^{L_0} \to 2^{L_0}$ is defined as follows. For every $\Phi \subseteq L_0$ we set $T_\vee(\Phi)$ to be $\bigcup_{\varphi \in \Phi} \widehat{\varphi}$.

Note that both mappings are well-defined and computable in polynomial time. We show by an example how we intend to use these mappings.

Example 150. Let Σ be a set of TGD$^{\vee,\neg}$ and EGD$^{\vee}$. If Σ has stratified witness as defined in [Deutsch et al., 2007], then it is easy to see that $T_\vee(T_\neg(\Sigma))$ is weakly acyclic. The converse is not true. Consider e.g. the constraint set Σ consisting of

$\mathtt{R}(x_1, x_2) \to \exists y \neg\, \mathtt{S}(x_1, y), \mathtt{T}(x_1, y)$, and
$\mathtt{S}(x_1, x_2) \to \exists y \neg\, \mathtt{R}(x_1, y), \mathtt{T}(x_1, y)$.

The example shows that, if we can derive termination guarantees for $T_\vee(T_\neg(\Sigma))$, then we are more general than stratified witness, the only termination condition known so far that covers constraints containing negation and disjunction. The next theorem fills this gap and allows to derive such termination bounds from existing termination conditions for (negation- and disjunction-free) TGDs and EGDs.

Theorem 151. Let Σ be a set of TGD$^{\vee,\neg}$ and EGD$^{\vee}$.

- If $T_\neg(\Sigma) \in \mathrm{CT}_{\forall\forall}$, then $\Sigma \in \mathrm{CT}_{\forall\forall}$.
- If $\Sigma = T_\vee(\Sigma)$ and $T_\neg(\Sigma) \in \mathrm{CT}_{\forall\exists}$, then $\Sigma \in \mathrm{CT}_{\forall\exists}$.
- If $T_\vee(\Sigma) \in \mathrm{CT}_{\forall\forall}$, then $\Sigma \in \mathrm{CT}_{\forall\forall}$.
- If $T_\vee(T_\neg(\Sigma)) \in \mathrm{CT}_{\forall\forall}$, then $\Sigma \in \mathrm{CT}_{\forall\forall}$.

Proof.

- It is immediate from the definition of chase steps that the set of chase sequences with $T_\neg(\Sigma)$ coincide with the chase steps of Σ up to renaming of negative literals with $T_\neg$ and unsatisfiability.

- Let Σ contain no disjunctions. Observe again that the set of chase sequences with $T_\neg(\Sigma)$ coincide with the chase steps of Σ up to renaming of negative literals with $T_\neg$ and unsatisfiability.

- Suppose not. Then, there is an infinite chase sequence $Q_0 \xrightarrow{\alpha_1,\overline{a}_1} Q_1 \xrightarrow{\alpha_2,\overline{a}_2} Q_2 \ldots$, where $\{\alpha_i | i \in \mathbb{N}\} \subseteq \Sigma$. It follows that there is $q_0 \in Q_0 \cap \textsc{Cq}^\neg$ and for every $i \in \mathbb{N}$ there is $q_i \in \textsc{Cq}^\neg \cap Q_i$ and $\overline{\alpha}_i \in \widehat{\alpha}_i$ such that $q_0 \xrightarrow{\overline{\alpha}_1,\overline{a}_1} q_1 \xrightarrow{\overline{\alpha}_2,\overline{a}_2} q_2 \ldots$ is an infinite chase sequence, which implies $T_\vee(\Sigma) \notin \text{CT}_{\forall\forall}$.

- Follows from bullet one and three. $\square$

We emphasize that, with this powerful tool at hand, we subsume both the work on chase termination in the presence of $\textsc{Egd}^\vee$ from [Fagin et al., 2005] and for $\textsc{Tgd}^{\vee,\neg}$ from [Deutsch et al., 2007].

We now turn towards the question of data complexity (i.e. in the size of the input query) of chase termination.

Theorem 152. Let Σ be a set of $\textsc{Tgd}^{\vee,\neg}$ and $\textsc{Egd}^\vee$. If there is a function $f : \textsc{Ucq}^\neg \to \mathbb{N}$ such that for every $Q \in \textsc{Ucq}$ the depth of every chase tree for $T_\vee(T_\neg(\Sigma))$ and Q is bounded by $f(Q)$, then for every $Q' \in \textsc{Ucq}^\neg$ the depth of every chase tree for Σ and Q' is also bounded by $f(Q)$.

Proof. Suppose for a moment that there is $Q' \in \textsc{Ucq}^\neg$ and a chase tree for Σ whose depth is $d > f(Q')$. Then, this already holds for some $q' \in Q' \cap \textsc{Cq}^\neg$, which is why we assume $q' = Q'$. There must be a path in this chase tree whose length is d. Observe that we can build a chase sequence of length at least d for the query $T_\neg(q') \in \textsc{Cq}$ with constraints from $T_\vee(T_\neg(\Sigma))$ only, which is a contradiction. $\square$

In particular, it follows from this theorem that if we can data-independently derive polynomial data complexity for $T_\vee(T_\neg(\Sigma))$, which is the case for all termination conditions for the chase that have been developed so far, then the depth of every chase tree for Σ is also polynomial.

9.2 Minimization of RDF Graphs

As a second application scenario we turn towards the Semantic Web [Berners-Lee et al., 2001] and rule-based minimization of RDF graphs under constraints. This section can be read independently from the previous chapter on semantic query optimization in the presence of types.

The Semantic Web facilitates semantic interoperability and exchange of data between applications. The Resource Description Framework [World Wide Web Consortium, 2003a] was proposed by the World Wide Web consortium as a standard language for data in the Semantic Web. As RDF has only very simple language constructs, RDF data often becomes large. There has been a line of research [World Wide Web Consortium, 2003c; Gutierrez et al., 2004, 2003; Iannone et al., 2005; Esposito et al., 2005] to minimize RDF graphs without losing any information, i.e. retaining homomorphic equivalence. This allows applications to exchange reduced data, thus minimizing storage cost, transfer and query evaluation time. In [World Wide Web Consortium, 2003c; Gutierrez et al., 2004, 2003] the notion of lean graphs was introduced as a minimal representation of an RDF graph. Basically, a lean graph eliminates triples which contain blank nodes that specify redundant information. In [Iannone et al., 2005; Esposito et al., 2005] different algorithms are introduced that approximately compute a lean version of a given RDF graph. The notion of lean is orthogonal to derivability by application-specific rules. If such rules exist, a lean graph may still contain triples which are redundant in the sense that they need not be stored explicitly because they could be derived by the rules as well.
We propose a user-specific redundancy elimination technique based on rules. Before describing it, we give an example for our scenario. Consider Figure 9.2. It shows the original RDF graph that models some train connections between cities. If the transitive edges are not relevant for an application, we may want to eliminate them as shown in the second graph in the figure[2]. Yet, if an application often asks for connections from 'Fr' to any other city, then we want to keep all outgoing edges from 'Fr' in order to avoid unnecessary recomputations. This is depicted in the third graph. What we have done is the following. First, we eliminate triples according to the rule

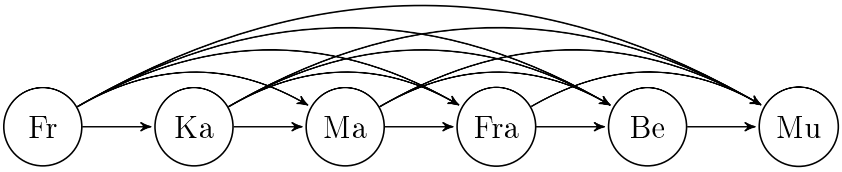
"delete all transitive edges"

and second we satisfied the constraint

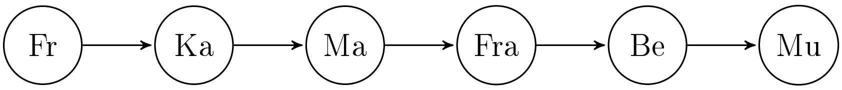
"keep all outgoing edges from 'Fr'".

---

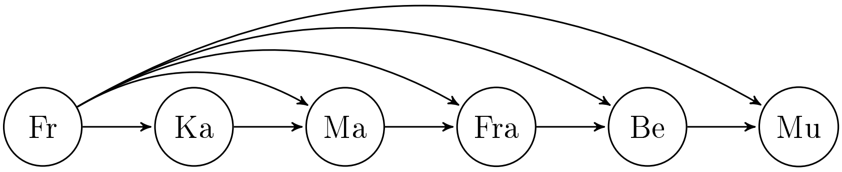
[2] An application of transitively reduced graphs was given for the context of transitive reduction [Aho et al., 1972] in [Vincent and Cecile, 2005].



The original graph.



A reduced graph.



A reduced graph that satisfies a constraint.

Figure 9.2: An RDF graph modeling some train connections.

The constraint expresses that we are mainly interested in connections from 'Fr' to the other cities. If we would be looking for connections from 'Ka' to 'Mu' we would have to perform some additional computations on the reduced graph.

Usually, rules are interpreted generatively, i.e. if we have a rule of the form

$$\mathtt{P}(X,Y) \leftarrow \mathtt{R}(Y,X)$$

and we find $\mathtt{R}(a,b)$ in our data, we add $\mathtt{P}(b,a)$ to it. In our work, we use rules in the following sense: whenever $\mathtt{P}(b,a)$ and $\mathtt{R}(a,b)$ are in our data, we delete $\mathtt{P}(b,a)$. If later needed, we can recompute the tuple $\mathtt{P}(b,a)$ again with the help of our rule, i.e. we minimize a given RDF graph such that all deleted triples can be reconstructed. We want to stress that our work is well-suited for scenarios in which it is known in advance what structures in the RDF graph are redundant and therefore building appropriate rules is possible. For example, this applies when the graph to be minimized has a user-defined rule semantics in the sense of [ter Horst, 2005], where it could be desirable to minimize a given graph along its user-defined rule semantics.
Additionally, we take data consistency into account. In [Lausen et al., 2008] it was proposed to extend RDF and RDFS [World Wide Web Consortium, 2003b] by constraints. We adapt the notion of tuple-generating dependencies to the RDF scenario and ensure that if such a kind of constraint is satisfied before the minimization step, then it is also satisfied afterwards. We will exploit these constraints for answering conjunctive queries on the reduced graph. For certain queries it is possible to use only the reduced graph to compute the answer to a query posed over the original graph. Of course, this is not always possible and for another case we show that we can guarantee a non-empty answer for a query on the reduced graph if the same query yields a non-empty answer on the original graph.

9.2.1 Additional Preliminaries

Throughout the paper, we will use several results from database theory that we adapt to the case of RDF. We introduce them in this section.

General mathematical notation. For sets M and N, $M \subset N$ denotes that M is a proper subset of N. For a mapping f, we denote by $f \upharpoonright_M$ the restriction of f to M, where M is a subset of f's domain.

Syntax of RDF. The Semantic Web necessitates data in a machine-readable format. RDF databases are sets of triples (s,p,o). Such a triple states a directed relationship p between s and o. More formally, an RDF vocabulary is a triple

(U, B, L), where U, B, L are infinite sets. U is usually referred to as the set of URI references, B is the set of blank nodes and L the set of literals. An RDF graph G (with respect to (U, B, L)) is a finite subset of $(U \cup B) \times U \times (U \cup B \cup L)$. RDF graphs have a graphical representation. A triple $(s, p, o) \in G$ can be seen as two nodes connected by a labeled arc $s \xrightarrow{p} o$. In a triple (s, p, o), s is called the subject, p the predicate and o the object of the triple. The subset of U that occurs in an RDF graph G is denoted by U_G. B_G and L_G are defined similarly. A map (with respect to (U, B, L)) is a function $\nu : (U \cup B \cup L) \to (U \cup B \cup L)$ such that for all $x \in (U \cup L) : \nu(x) = x$. For matters of convenience we introduce the following notion. A maplet[3] (with respect to (U, B, L) and G) is a function $\mu : (U \cup B_G \cup L) \to (U \cup B \cup L)$ such that for all $x \in (U \cup L) : \mu(x) = x$. If the image of such a μ is contained in a graph G', we will denote this by $\mu : G \to G'$. Two RDF graphs G_1, G_2 are called homomorphically equivalent if there are maps μ_1, μ_2 such that $\mu_1(G_1) \subseteq G_2$ and $\mu_2(G_2) \subseteq G_1$.

Constraints in RDF. Logical constraints are a useful tool to help modeling an application domain. They restrict the legal state space of a database and guarantee that only meaningful data can be inserted into a database. In [Lausen et al., 2008] it was proposed to add constraints to RDF in order to ensure data consistency. We are going to introduce a large class of constraints for RDF graphs, namely we will adapt the notion of tuple-generating dependencies.
Following previous approaches [Schmidt et al., 2010], we focus on TGDs. When talking about constraints in the following we always mean TGDs. We refer the interested reader to [Lausen et al., 2008] for motivating examples and a study of constraints for RDF.

We represent each constraint $\alpha \in \Sigma$ by a first-order logic formula over a ternary relation $\mathtt{T}(s, p, o)$ that stores all triples contained in RDF database G and use $\mathtt{T}$ as the corresponding relation symbol. For instance, the constraint

$$\forall x_1, x_2 (\mathtt{T}(x_1, p_1, x_2) \to \exists y_1 \; \mathtt{T}(x_1, p_2, y_1))$$

states that each RDF resource with property p_1 also has property p_2.

Definition 153. Given $b \in \mathbb{N}$, φ is called b-bounded if $|body(\varphi)| \leq b$ and $|head(\varphi)| \leq b$. A set Σ of TGDs is called b-bounded if every element of Σ is b-bounded.

---

[3] The notion of maplet is introduced because in many situations we are not interested in the image of blank nodes that do not occur in an RDF graph. It can always be extended to a map.

Note that we did not define the semantics of a constraint in terms of interpretations of RDF graphs like in [World Wide Web Consortium, 2003c], but only gave an algebraic version of satisfaction. The following proposition states that checking constraints is tractable. The proof follows from classical results, see [Flum and Grohe, 2006; Johnson and Klug, 1982].

Proposition 154. Let $b \in \mathbb{N}$ fixed. For any RDF graph G and b-bounded set of TGDs Σ, it can be tested in polynomial time whether $G \models \Sigma$ holds.

Datalog. Rules allow to derive new knowledge from given knowledge and especially add recursion to a database query language. We define syntax and semantics of Datalog.

Definition 155. A (Datalog) rule is of the form $t \leftarrow G$, where t is an RDF triple and G is an RDF graph such that $B_{\{t\}} \subseteq B_G$. Given $b \in \mathbb{N}$, $t \leftarrow G$ is called b-bounded if $|G| \leq b$. A set of rules $\mathcal{R}$ is called b-bounded if every element of $\mathcal{R}$ is b-bounded. The set $head(\mathcal{R})$ is defined as $\{p|\ \mathtt{T}(s,p,o) \leftarrow G \in \mathcal{R}\}$.

Definition 156. Let G be an RDF graph. The semantics of a set of rules $\mathcal{R} = \{t_1 \leftarrow G_1, ..., t_n \leftarrow G_n\}$ is defined via the help of the $T_\mathcal{R}$-operator, where $T_\mathcal{R}(G) := G \cup \{\ \mu(t_i) \mid i \in [n]$ and there is a maplet $\mu : B_{G_i}$ with $\mu(G_i) \subseteq G\ \}$. The semantics of $\mathcal{R}$ applied to G is $\mathcal{R}(G) := \bigcup_{i=1}^{\infty} T_\mathcal{R}^i(G)$.

The $T_\mathcal{R}$-operator is monotonic, therefore $\mathcal{R}(G) \supseteq G$. This means that we can generate new data from old one, but never lose original data. Note that $\mathcal{R}(G)$ may not be an RDF graph again because it can happen that a literal occurs in the subject position of a triple. The following proposition states that evaluating rules takes polynomial data complexity and is well-known in database theory, see [Abiteboul et al., 1995].

Proposition 157. Let $b \in \mathbb{N}$ fixed and $\mathcal{R}$ any b-bounded set of rules. Then, for any RDF graph G there exists $n \in \mathbb{N}$ such that $\mathcal{R}(G) := T_\mathcal{R}^n(G)$. Furthermore, the mapping $(G, \mathcal{R}) \mapsto \mathcal{R}(G)$ can be computed in polynomial time.

Proof. Clearly, $|\mathcal{R}(G)| \leq |\mathcal{R}| \cdot 27 \cdot (|G| + |\mathcal{R}|)^3$ and $T_\mathcal{R}$ is monotonic. Therefore, there exists $n_0 \leq |\mathcal{R}| \cdot 27 \cdot (|G| + |\mathcal{R}|)^3$, such that $T_\mathcal{R}^{n_0}(G) = T_\mathcal{R}^{n_0+1}(G)$. It follows that $\mathcal{R}(G) = \bigcup_{i=1}^{n_0} T_\mathcal{R}^i(G) = T_\mathcal{R}^{n_0}(G)$. Let $\widetilde{G} \subseteq \mathcal{R}(G)$. It remains to show that for any $i \in [|\mathcal{R}| \cdot 27 \cdot (|G| + |\mathcal{R}|)^3]$ it holds that $(\mathcal{R}, \widetilde{G}) \mapsto T_\mathcal{R}(\widetilde{G})$ can be computed in polynomial time. Given $r := t' \leftarrow G' \in \mathcal{R}$ there are at most $|G|^{2b}$ maplets $\mu : G' \to G$ because any triple in G' can contain at most two blank nodes. For every such μ check whether $\mu(G') \subseteq G$ holds. For fixed r this can be done in polynomial time. As there is only a polynomial number of rules, computing $T_{\mathcal{R}}(\widetilde{G})$ can be done in polynomial time. □

9.2.2 Formal Description and Examples

This section formally introduces the minimization problem. We consider two versions of it: the construction problem and the corresponding decision problem.
As already mentioned earlier, we do not use Datalog rules to generate new data. Instead, we delete it from a given RDF graph. Intuitively, we define the inverse semantics of a rule of the form $t \leftarrow G$ in such a way that, we delete t, if G is still in the graph afterward. This means that we are interested in subsets G' of a given RDF graph G such that $\mathcal{R}(G') \supseteq G$ (or equivalently $\mathcal{R}(G') = \mathcal{R}(G)$ because Datalog rules are monotonic). Unfortunately, such a G' is, in general, not unique. This motivates the following definition.

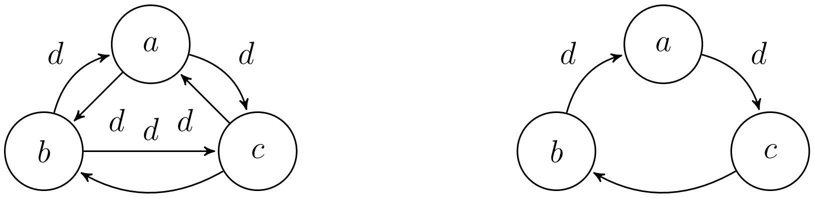
Definition 158. Let $\mathcal{R}$ be a set of Datalog rules. The inverse semantics of G with respect to $\mathcal{R}$ is given by $\mathcal{R}^{-}(G) := \{\, G' \subseteq G \mid \mathcal{R}(G') \supseteq G$ and there is no $G'' \subseteq G'$ with $\mathcal{R}(G'') \supseteq G \,\}$. If $G' \in \mathcal{R}^{-}(G)$, we call G' a reduction of G along $\mathcal{R}$.

Example 159. Figure 9.3 shows an RDF graph G and three reductions along $\mathcal{R} = \{\mathtt{T}(X, d, Z) \leftarrow \mathtt{T}(X, d, Y),\ \mathtt{T}(Y, d, Z)\}$.

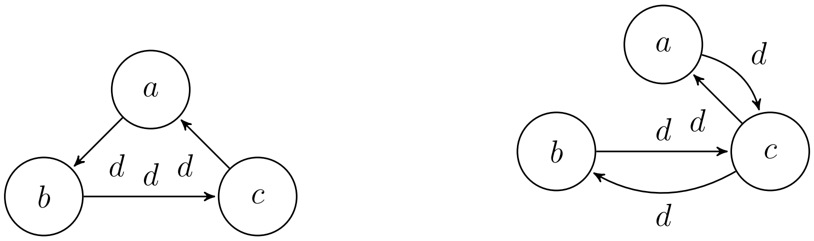
The construction problem Mini-rdf is defined as follows.

Mini-rdf	
Input :	An RDF graph G, a set of Datalog rules $\mathcal{R}$, a set of constraints Σ with $G \models \Sigma$.
Task :	Find $G' \subset G$ such that (i) $G' \models \Sigma$, (ii) $\mathcal{R}(G') \supseteq G$ and (iii) G' is minimal (w.r.t. to its cardinality) with (i) and (ii).
Answer:	G', if such a G' exists. No, otherwise.

Note that in the definition of Mini-rdf G' must be a proper subset of G. If G' can only be chosen equal to G, then the answer to Mini-rdf will be NO. To denote that G' is a solution to Mini-rdf for the input $G, \mathcal{R}, \Sigma$ we write $G' \in$ Mini-rdf$(G, \mathcal{R}, \Sigma)$. The corresponding decision problem Mini-rdf$_{dec}$ is the following.



Original graph G (1)



(2) (3)

Figure 9.3: An RDF graph and three possible reductions along transitivity.

Mini-rdf$_{dec}$
Input : $(G, \mathcal{R}, \Sigma)$ such that $G \models \Sigma$.
Question: Is there $G' \in$ Mini-rdf$(G, \mathcal{R}, \Sigma)$?
Answer: Yes or no.

A simple example for solutions to Mini-rdf is the following. It does not take any constraints into account.

Example 160. Consider again Example 159. Let additionally be $\Sigma = \emptyset$. Then, Mini-rdf$(G, \mathcal{R}, \Sigma)$ has the two solutions (1) and (2) depicted in Figure 9.3. This example shows that solutions to Mini-rdf are in general not homomorphically equivalent.

The next example makes use of a constraint. It demonstrates the interaction between the rules' inverse semantics and constraints.

Example 161. Consider again Figure 9.2 from the introduction and assume that all edges are implicitly labeled by `reach`. The transitivity rule $\mathcal{R} = \{\mathtt{T}(X, reach, Z) \leftarrow \mathtt{T}(X, reach, Y), \mathtt{T}(Y, reach, Z)\}$ and the constraint $\Sigma = \{\mathtt{T}(Fr, reach, X), \mathtt{T}(X, reach, Y) \rightarrow \mathtt{T}(Fr, reach, Y)\}$. The figure shows the original graph G, the only element of $\mathcal{R}^-(G)$ and the only solution to $\textsc{Mini-rdf}(G, \mathcal{R}, \Sigma)$.

The following theorem gives us a method which enables us to compute solutions to Mini-rdf using the well-known technique of the chase.

Theorem 162. Let $\Sigma \in \mathrm{CT}_{\forall\exists}$. If $G' \in \textsc{Mini-rdf}(G, \mathcal{R}, \Sigma)$, then there exists $\widetilde{G} \in \mathcal{R}^-(G)$ and $\pi \in Hom(\widetilde{G}^\Sigma, G)$ such that $|G'| = |\pi(\widetilde{G}^\Sigma)|$.

Proof. Assume that for all $\widetilde{G} \in \mathcal{R}^-(G)$ and for all $\pi \in Hom(\widetilde{G}^\Sigma, G)$ it holds that $|G'| \neq |\pi(\widetilde{G}^\Sigma)|$.

Case 1: Assume that there is $\widetilde{G} \in \mathcal{R}^-(G)$ and $\pi \in Hom(\widetilde{G}^\Sigma, G)$ such that $|G'| > |\pi(\widetilde{G}^\Sigma)|$. Clearly, for every choice of $\widetilde{G}$ and π we have that $\pi(\widetilde{G}^\Sigma) \models \Sigma$ and $\mathcal{R}(\pi(\widetilde{G}^\Sigma)) \supseteq G$. Thus, $G' \notin \textsc{Mini-rdf}(G, \mathcal{R}, \Sigma)$.

Case 2: Assume that for all $\widetilde{G} \in \mathcal{R}^-(G)$ and for all $\pi \in Hom(\widetilde{G}^\Sigma, G)$ it holds that $|G'| < |\pi(\widetilde{G}^\Sigma)|$. Define $A := \{ G'' \subseteq G' \mid G'' \in \mathcal{R}^-(G) \}$. By definition $A \neq \emptyset$. Choose $\widetilde{G} \in A$ arbitrarily. As $\widetilde{G} \subseteq G'$ $\iota : \widetilde{G} \to G'$ is a maplet, where $\iota(x) = x$ for all x in the domain of ι. $G' \models \Sigma$. So, there exists a maplet $\pi : \widetilde{G}^\Sigma \to G'$ with $\pi \supseteq \iota$ (see [Johnson and Klug, 1982]), thus $\pi \in Hom(\widetilde{G}^\Sigma, G)$ and $|\pi(\widetilde{G}^\Sigma)| \leq |G'|$. By assumption $|G'| < |\pi(\widetilde{G}^\Sigma)|$, which yields a contradiction. ☐

Given this theorem we can solve Mini-rdf in the following three steps if the set of constraints is in $\mathrm{CT}_{\forall\exists}$. Of course these steps depend on each other and, in general, cannot be solved separately:

1. Guess an adequate $\widetilde{G} \in \mathcal{R}^-(G)$.
2. Compute $\widetilde{G}^\Sigma$.
3. Find $\pi \in Hom(\widetilde{G}^\Sigma, G)$ such that $\pi(\widetilde{G}^\Sigma) \in \textsc{Mini-rdf}(G, \mathcal{R}, \Sigma)$.

A natural question that arises is whether for step one it suffices to take only $\widetilde{G} \in \mathcal{R}^-(G)$ of minimal cardinality into account. The next example shows that this conjecture is wrong, in general.

Example 163. Consider the graph G from Figure 9.3 together with the transitivity rule $\mathcal{R} = \{\mathtt{T}(X, reach, Z) \leftarrow \mathtt{T}(X, reach, Y), \mathtt{T}(Y, reach, Z)\}$ and the constraint $\Sigma = \{\mathtt{T}(X, d, Y) \leftarrow \mathtt{T}(Y, d, X)\}$. For readability we omitted all arc labels, but we assume that any arc is implicitly labeled by the URI *reach*. Then, graph (3) in the figure is a solution to Mini-rdf$(G, \mathcal{R}, \Sigma)$ and graphs (1) and (2) cannot be extended to a solution. Note that the graphs (1) and (2) are the only elements in $\mathcal{R}^{-}(G)$ of minimal cardinality.

9.2.3 Complexity Results

This section establishes complexity results for Mini-rdf$_{dec}$ and Mini-rdf. While it follows from the definition that the problem is decidable in general, we give an exact complexity bound for a restricted b-bounded version of Mini-rdf$_{dec}$. Namely, we show that it is Np-complete.

Proposition 164. Let $b \in \mathbb{N}$ fixed. Mini-rdf$_{dec}$ restricted to instances of b-bounded sets of TGDs and b-bounded sets of rules is solvable by an Np-algorithm.

Proof. Let $(G, \mathcal{R}, \Sigma)$ be an input for Mini-rdf$_{dec}$. Non-deterministically guess $G' \subset G$ and check whether $\mathcal{R}(G') \supseteq G$ and $G' \models \Sigma$. Notice that by Propositions 154 and 157 these steps take polynomial time. □

Theorem 165. Mini-rdf$_{dec}$ restricted to instances of 2-bounded sets of full dependencies and 1-bounded sets of rules is Np-hard. This still holds if neither the rules, the constraints nor the input graph contain any blank nodes.

Proof. It is well-known that $CNF - SAT$, the satisfiability problem for boolean formulas in conjunctive normal form, is Np-complete under polynomial-time many-one reductions, see [Arora and Barak, 2009]. We show that $CNF - SAT$ can be reduced to Mini-rdf$_{dec}$. An instance α of $CNF - SAT$ is of the form

$$(x_{1,1} \vee ... \vee x_{1,k_1}) \wedge ... \wedge (x_{l,1} \vee ... \vee x_{l,k_l}),$$

where $x_{i,j}$ are literals. Without loss of generality, we assume that $(x \vee \neg x)$ is a conjunct in α for every variable x that occurs in α. $\overline{x_{i,j}} = x_{i,j}$, if $x_{i,j}$ is a positive literal and $\overline{x_{i,j}} = p$, if $x_{i,j} = \neg p$ for a positive literal p. A possible reduction is given by $\alpha \mapsto (G, \mathcal{R}, \Sigma)$, where

$$\begin{array}{rcll} G & = & & \{\ T(x_{i,j}, i, x_{i,j}) \mid i \in [l], j \in [k_i]\ \} \cup \{T(d,d,d)\} \\ \mathcal{R} & = & & \{\ T(x_{i,j}, i, x_{i,j}) \leftarrow T(x_{i,j'}, i, x_{i,j'}) \mid i \in [l], j, j' \in [k_i], j \neq j'\ \} \\ & & \cup & \{\ T(d,d,d) \leftarrow t \mid t \in G \backslash \{T(d,d,d)\}\ \} \\ \Sigma & = & & \{\ T(x_{i,j}, i, x_{i,j}) \rightarrow T(x_{i',j'}, i', x_{i',j'}) \mid x_{i',j'} = x_{i,j}\ \} \\ & & \cup & \{\ T(x_{i,j}, i, x_{i,j}), T(x_{i',j'}, i', x_{i',j'}) \rightarrow t \mid \overline{x_{i',j'}} = x_{i,j}, t \in G\ \}. \end{array}$$

Note that $\alpha \mapsto (G, \mathcal{R}, \Sigma)$ is computable in polynomial time and that neither G, $\mathcal{R}$ nor Σ contain any blank nodes. It remains to show that α is satisfiable if and only if MINI-RDF$(G, \mathcal{R}, \Sigma)$ admits a solution.

(a) Assume that b is a satisfying assignment for α. Define $G' := \{ \ T(x_{i,j}, i, x_{i,j}) \mid i \in [l], j \in [k_i], b(x_{i,j}) = 1 \ \}$. We will show that

1. $\mathcal{R}(G') \supseteq G$, and

2. $G' \models \Sigma$ and $G' \subset G$.

Obviously, $\emptyset \neq G' \subseteq G$ holds. (1): Clearly, $T(d, d, d) \in \mathcal{R}(G')$ because $\emptyset \neq G'$ and for any $s \in G' \backslash \{T(d, d, d)\}$ there is $T(d, d, d) \leftarrow s \in \mathcal{R}$. Let $t \in G \backslash \{T(d, d, d)\}$ arbitrarily. Then, there exist $i \in [l], j \in [k_i]$ such that $t = T(x_{i,j}, i, x_{i,j})$. As b satisfies α, b also satisfies the i-th conjunct of α. So, there is $j' \in [k_i]$ such that $b(x_{i,j'}) = 1$. But then $T(x_{i,j}, i, x_{i,j}) \leftarrow T(x_{i,j'}, i, x_{i,j'}) \in \mathcal{R}$ which implies $t \in \mathcal{R}(G')$. (2): A constraint of the form $T((x_{i,j}, i, x_{i,j}) \rightarrow T(x_{i',j'}, i', x_{i',j'}))$ where $x_{i',j'} = x_{i,j}$ is satisfied because $b(x_{i',j'}) = b(x_{i,j})$. Constraints of the form $T((x_{i,j}, i, x_{i,j}), (x_{i',j'}, i', x_{i',j'}) \rightarrow t)$ $(\overline{x_{i',j'}} = x_{i,j}, t \in G)$ are satisfied too because otherwise $b(x_{i,j}) = b(x_{i',j'}) = 1$ although $\overline{x_{i',j'}} = x_{i,j}$. So, overall we have found a proper subset of G that satisfies (1) and (2). This implies MINI-RDF$(G, \mathcal{R}, \Sigma) \neq \emptyset$.

(b) Assume conversely that $G' \in$ MINI-RDF$(G, \mathcal{R}, \Sigma)$. Note that $T(d, d, d) \notin G'$. We define a satisfying assignment b for α. For $i \in [l], j \in [k_i]$, if $T(x_{i,j}, i, x_{i,j}) \in G'$, then $b(x_{i,j}) := 1$. Note that b is well-defined and that it cannot occur that $b(x_{i,j}) = b(\overline{x_{i,j}}) = 1$ because $G' \models \Sigma$. For all literals x that occur in α but neither x nor $\overline{x}$ was assigned a truth value by the previous step, set $b(x) \in \{0, 1\}$ arbitrarily. Assume b does not satisfy the i-th conjunct in α. By assumption $\mathcal{R}(G') \supseteq G$. So, by construction, there must be $j \in [k_i]$ such that $T(x_{i,j}, i, x_{i,j}) \in G'$. This means $b(x_{i,j}) = 1$. Thus, b satisfies the i-th conjunct in α, which is a contradiction. □

This result demonstrates even in such a restricted case the interaction between the rules' inverse semantics and the constraints is so complex that this leads to NP-hardness.

Corollary 166. Let $b \geq 2$ fixed. MINI-RDF$_{dec}$ restricted to instances of b-bounded sets of TGDs and b-bounded sets of rules is NP-complete under polynomial-time many-one reductions.

9.2.4 A Tractable Fragment

The complexity results for MINI-RDF$_{dec}$ and therefore also for MINI-RDF from the last section are negative. We will now look for tractable subsets. The proof of Theorem 165 seems to imply that recursion in the Datalog rules along with cycles in the input RDF graph are a source of high complexity. This is why we will restrict ourselves to a case where recursion is limited and the constraints are a set of full dependencies. We will give a syntactic restriction in terms of acyclicity of a graph.

Definition 167. Let $\mathcal{R}$ be a set of rules and G an RDF graph. The data dependency graph with respect to $(\mathcal{R}, G)$ is defined as $dep(\mathcal{R}, G) = (\mathcal{R}(G), E_{\mathcal{R}})$, where $E_{\mathcal{R}} := \{(v, w) \mid \text{there are } t \leftarrow \widetilde{G} \in \mathcal{R}, \mu : \widetilde{G} \to \mathcal{R}(G) \text{ such that } \mu(t) = w, v \in \mu(\widetilde{G})\}$. $(\mathcal{R}, G)$ is called acyclic if $dep(\mathcal{R}, G)$ is acyclic, i.e. for every node in the graph, there is no directed path to itself.

The intuition why the acyclicity of $dep(\mathcal{R}, G)$ yields a tractable fragment of MINI-RDF is the following. We want to solve MINI-RDF according to the steps in Theorem 162. The main problem in the reduction along the rules is that we do not know in what order triples should be deleted. When $dep(\mathcal{R}, G)$ is acyclic, we can impose an order on $\mathcal{R}(G)$ such that the deletion of an element that has a very high rank in this order will not affect the deletion of an element with a low rank. Deleting triples according to that order yields the unique reduction of G along $\mathcal{R}$.

Proposition 168.

1. If $(\mathcal{R}, G)$ is acyclic, then $|\mathcal{R}^-(G)| = 1$.
2. If Σ is a set of full dependencies, then $|Hom(\widetilde{G}^{\Sigma}, G)| = 1$ for any $\widetilde{G} \subseteq G$.

Proof. 1.) Define $G' := \{\ w \in G \mid \text{for all } v \in \mathcal{R}(G) : (v, w) \notin E_{\mathcal{R}}\ \}$. We will show that $\mathcal{R}^-(G) = \{G'\}$.
Assume that $G' \notin \mathcal{R}^-(G)$. Then, there is $a \in G'$ such that $\mathcal{R}(G' \backslash \{a\}) \supseteq G$. Thus, there must be a rule $t \leftarrow \widetilde{G} \in \mathcal{R}$ and a maplet $\mu : G' \backslash \{a\} \to \mathcal{R}(G)$ such that $a = \mu(t)$ and $\mu(\widetilde{G}) \subseteq G' \backslash \{a\}$. From the construction of G' it follows that $a \notin G'$, which is a contradiction.
Conversely, assume there is some $G'' \in \mathcal{R}^-(G)$. We will show that $G'' = G'$. Assume for a short moment that $G'' \neq G'$. Case 1: there is some $a \in G'' \backslash G'$. Without loss of generality, we can assume that a is minimal with respect to the order of an arbitrary topological sorting of $\mathcal{R}(G)$ according to $E_{\mathcal{R}}$. In case that a has no predecessors in $E_{\mathcal{R}}$, then $a \in G'$ by construction. Otherwise, a must have a

predecessor. It follows that $a \notin \mathcal{R}^-(G''\backslash\{a\})$. So, there must be $b \in \mathcal{R}(G)\backslash\mathcal{R}(G'')$ such that $(b,a) \in E_{\mathcal{R}}$. This is a contradiction to the minimality of a.
Case 2: there is some $a \in G'\backslash G''$. Then, it must hold that $\mathcal{R}(G') \neq \mathcal{R}(G'')$. We show that $a \notin \mathcal{R}(G'')$. Assume that $a \in \mathcal{R}(G'')$. Then there must be a rule $t \leftarrow \widetilde{G} \in \mathcal{R}$ and a maplet $\mu : \widetilde{G} \to \mathcal{R}(G)$ such that $\mu(t) = a$. As $\widetilde{G} \neq \emptyset$, a must have a predecessor in $E_{\mathcal{R}}$. But as $a \in G'$, then by construction of G' a cannot have any predecessors in $E_{\mathcal{R}}$, which is a contradiction.

2.) $|Hom(\widetilde{G}^{\Sigma}, G)| = 1$ for any $\widetilde{G} \subseteq G$ holds because Σ is a set of full dependencies, the chase does not introduce any variables and therefore the identity is the only element in $Hom(chase_{\Sigma}(\widetilde{G}), G)$. □

As a consequence, we obtain the following corollary, which states that under the conditions of the proposition computing solutions to Mini-rdf is tractable if we consider data complexity[4].

Corollary 169. Let $\mathcal{R}$ be a fixed set of rules and Σ be a fixed set of constraints. For every RDF graph G such that $(\mathcal{R}, G)$ is acyclic, a solution to Mini-rdf$(G, \mathcal{R}, \Sigma)$ can be computed in polynomial time (with respect to $|G|$).

Proof. Assume that G is an RDF graph such that $(\mathcal{R}, G)$ is acyclic. We give an algorithm that computes a solution to Mini-rdf$(G, \mathcal{R}, \Sigma)$. Let G' be as in the proof of Proposition 168 and note that the mapping $G \mapsto G'$ can be computed in polynomial time. Compute G'^{Σ}. This can be in time polynomial in $|G'|$. If $chase_{\Sigma}(G') \subset G$, then return $chase_{\Sigma}(G')$, otherwise return NO. By Theorem 162 and Proposition 168 this algorithm works correctly. We already argued that it takes time polynomial in $|G|$. □

As a possible example consider again $\mathcal{R}, G$ from Example 161. It can be easily seen that $(\mathcal{R}, G)$ is acyclic there.
Next, we will show that a special case of $(\mathcal{R}, G)$ being acyclic is that $\mathcal{R}$ is acyclic, i.e. non-recursive. We first repeat the definition of an acyclic set of rules.

Definition 170. Let $\mathcal{R}$ be a set of rules.

1. The dependency graph $dep(\mathcal{R}) = (V_{\mathcal{R}}, E_{\mathcal{R}})$ is defined as
   $V_{\mathcal{R}} := \{ v \mid v$ is a predicate symbol that occurs in $\mathcal{R} \}$ and
   $E_{\mathcal{R}} := \{ (v,w) \in V_{\mathcal{R}} \times V_{\mathcal{R}} \mid v$ occurs in the body of a rule with head $w \}$.
2. $\mathcal{R}$ is called acyclic if and only if $dep(\mathcal{R})$ is acyclic.

---

[4] We assume that the data, i.e. the RDF graph, is much larger than the rules and the constraints.

The following remark shows that the tractable fragment given in Theorem 169 generalizes the case when the set of rules is acyclic.

Remark 171. If $\mathcal{R}$ is an acyclic set of rules, then for any RDF graph G it holds that $(\mathcal{R}, G)$ is acyclic.

Thios remark can be proven by observing that a cycle in $\mathit{dep}(\mathcal{R}, G)$ would force a cycle in $\mathit{dep}(\mathcal{R})$.

9.2.5 Query Answering

So far, we were able to reduce the size of an RDF graph via rules. Now we want to consider the problem of answering conjunctive queries posed on the original graph using the reduced graph only. Consider a query q. As the reduced graph G' is always a subset of the input graph G, $q(G') \subseteq q(G)$. But there are some cases where $q(G) = q(G')$ or at least $q(G) \neq \emptyset \Leftrightarrow q(G') \neq \emptyset$ holds. We will briefly introduce the basic definitions and then use the Chase and Backchase technique [Deutsch et al., 2006] in order to answer queries.

From Theorem 21 we obtain as a first result the possibility to answer a query on the full graph using the query and reduced graph only. We use the Chase & Backchase in order to determine if it is possible to rewrite a given query such that its body does not contain any predicates that could have been minimized by the rules. Of course, this is not always possible and stronger results are left as future work.

Corollary 172. Let $\mathcal{R}$ be a set of rules, $\Sigma \in \mathrm{CT}_{\forall\forall}$, $G' \in \textsc{Mini-rdf}(G, \mathcal{R}, \Sigma)$ and $q : h \leftarrow \mathit{body}(q)$ a conjunctive query. The query q_0 is defined as $\emptyset \leftarrow \mathit{body}(q)$. If there is a conjunctive query q' such that no predicate in $\mathit{body}(q')$ appears in $\mathit{head}(\mathcal{R})$ and

1. $q \equiv_\Sigma q'$, then $q(G) = q(G')$.
2. $q_0 \equiv_\Sigma q'$, then $q(G) \neq \emptyset \Leftrightarrow q(G') \neq \emptyset$.

Proof.

1. Similar to point two.
2. If $q(G) \neq \emptyset$, then $\{\emptyset\} = q_0(G) = q'(G)$ because $G \models \Sigma$. Furthermore, $q'(G) = q'(G')$ because $\mathit{body}(q)$ contains no predicates that could have been minimized. As $G' \models \Sigma$, $q'(G') = q_0(G')$. Therefore, $q(G') \neq \emptyset$. $\square$

This corollary gives us a case where exact query answering is possible and a second case in which we can expect non-empty answers on the reduced graph for the case that we had a non-empty answer in the original graph.

Chapter 10

Related Work

> **Riccardo:** “Can you summarize again what has already been done?”
> **Alice:** “No problem!”

We have sketched in Chapter 4 that the chase has many applications. It is not surprising that there are wide research fields related to the chase, e.g. those that use the chase as a subprocedure, but also those that try to extend the chase in various ways. Although we already mentioned related work throughout the thesis, this chapter’s aim is to summarize it concisely.

10.1 Constraints in Databases

Integrity constraints in database systems (also known as data dependencies) have been considered from the early stages in database literature [Codd, 1970, 1971, 1974; Hammer and McLeod, 1975; Chen, 1976; Codd, 1979] up to current publications in the field [Abiteboul et al., 1995; Deutsch et al., 2007; Deutsch and Nash, 2008a; Deutsch et al., 2006; Aho et al., 1979; Beeri and Vardi, 1984; Calì et al., 2008, 2009; Cheng et al., 1999; Cosmadakis and Kanellakis, 1986; Deutsch et al., 2008; Fagin et al., 2005; Fuxman et al., 2005; Gottlob and Nash, 2008; Johnson and Klug, 1982; Maier et al., 1979; Marnette, 2009]. Relational constraints allow the database schema designer to restrict the contents of a database to “meaningful” data only. They ensure that the contents of the database obey certain real-life properties. They are important for two major reasons. First, they can be used in database schema design to create schemas which lack certain insert, update and deletion anomalies [Codd, 1971, 1974; Chen, 1976]. Second, they can be used in query optimization [King, 1986; Beeri and Vardi, 1984; Chakravarthy et al., 1990] to rewrite queries into a logically equivalent form which has lower execution costs. Usually, integrity constraints are represented as logical expressions without free variables. For a more thorough introduction to relational constraint types and their first-order representation, we recommend you to read the article [Deutsch and Nash, 2008a]. In this thesis, we have mostly considered a special fragment of constraints, namely equality- and tuple-generating dependencies. These are able to express virtually all database constraints from the literature [Abiteboul et al., 1995; Deutsch and Nash, 2008a], e.g. functional dependencies, key dependencies, join dependencies, multi-valued dependencies, inclusion dependencies and foreign key dependencies. Furthermore, we have seen the class of embedded dependencies. It is well-known that we can express every embedded dependency as a finite set of equality- and tuple-generating dependencies. This is why all of our results for equality- and tuple-generating dependencies directly carry over to embedded dependencies.

10.2 Previous Results on Chase Termination

We were rather surprised to find out that there has not been so much work on chase termination although so many applications depend on it. The textbook [Abiteboul et al., 1995] covers the notions of *full* constraints as well as *acyclic* constraints and shows that they guarantee chase termination in the sense of $\mathrm{CT}_{\forall\forall}$. Both classes of constraints are subsumed by the famous weak acyclicity condition in [Fagin et al., 2005] (cf. Section 3.6.2) which is a subset of $\mathrm{CT}_{\forall\forall}$, too. The authors of [Deutsch et al., 2008] aimed to improve weak acyclicity to stratification (cf. Section 3.6.3) but in this thesis we could show that it only guarantees termination only in the sense of $\mathrm{CT}_{\forall\exists}$ and not $\mathrm{CT}_{\forall\forall}$ (cf. Section 7.1). In [Marnette, 2009], weak acyclicity was generalized to *super-weak acyclicity*, which is a fragment of $\mathrm{CT}_{\forall\forall}$ using skolemization and unification techniques (cf. Section 3.6.4). With the help of our modular techniques, we can easily define decidable fragments of $\mathrm{CT}_{\forall\forall}$ that are supersets of super-weak acyclicity by allowing in the definition of our termination conditions that every strongly connected component of a chase graph or k-restriction system may be either safe or super-weakly acyclic.

10.3 Further Chase-Like Algorithms

The chase is extended in [Deutsch et al., 2007] to work with a fragment of first-order constraints[1], which includes negation and disjunction. As a possible application, [Deutsch et al., 2007] considers the problem of rewriting queries using views with access patterns under integrity constraints which unifies the works in [Deutsch et al., 2006], [Halevy, 2001], [Deutsch et al., 2006] and other works on query answering under limited access patterns like [Rajaraman et al., 1995] and [Florescu et al., 1999]. They also show that classical results on query containment

---

[1] Basically, this fragment is logically equivalent to the class of $\forall\exists$-sentences.

from [Chandra and Merlin, 1977] (see Theorem 19) carry over with minor modifications only when the integrity constraints contain negation and disjunction.

In [Deutsch et al., 2008] the *core chase* is introduced, a variant of the chase that is sound and complete for finding universal models in the style of Theorem 3. If the chase terminates, so does the core chase, but, in general, the converse is not true. So far, there has been no work on sufficient termination conditions for the core chase that do not also guarantee termination of the chase. The authors of [Deutsch et al., 2008] extend the core chase to treat negation and disjunction and show how classical results for the chase can be extended for the core chase. As the core chase involves computing cores of database instances (and core computation is known to be intractable [Chandra and Merlin, 1977]), it is unclear how to efficiently implement it.

We already introduced the oblivious chase in the preliminaries. It is known that if the oblivious chase terminates and does not fail, then it also produces universal models in the sense of Theorem 3. It seems reasonable to assume that an oblivious chase step can be implemented more efficiently than a standard chase step because we do not need to check the satisfaction of the constraints' heads. At first glance, the oblivious chase seems preferable to the standard chase. Whenever the oblivious chase terminates data-independently, so does the standard chase. But the oblivious chase does not always terminate when the standard chase does. An example would be the single TGD

$\mathtt{R}(x, y) \rightarrow \exists z\ \mathtt{R}(x, z), \mathtt{R}(x, y)$[2],

ensuring termination of the standard chase but not termination of the oblivious chase. As for many applications it is desirable or absolutely necessary to have termination, it seems almost inevitable to use the standard chase instead of the oblivious chase.

In [Marnette, 2009] a variation of the oblivious chase is considered which is called *oblivious skolem chase*. The idea is that the constraint set is first skolemized and then this new constraint set is evaluated as a logic program on the underlying data. Like the (oblivious) chase algorithm, the oblivious skolem chase outputs universal models.

10.4 Classical Applications

The chase procedure has been successfully applied in a variety of database applications [Maier et al., 1979; Johnson and Klug, 1982; Beeri and Vardi, 1984; Halevy, 2001; Deutsch et al., 2007; Lenzerini, 2002; Fagin et al., 2005; Fuxman et al., 2005; Deutsch et al., 2006; Olteanu et al., 2009]. Originally, it was proposed to tackle the implication problem [Zhang and Ozsoyoglu, 1993; Maher and Srivastava, 1996;

Coulondre, 2003] for data dependencies [Maier et al., 1979; Beeri and Vardi, 1984] and to optimize conjunctive queries under data dependencies [Aho et al., 1979; Johnson and Klug, 1982]. In [Johnson and Klug, 1982] it was shown that the chase can be used to reduce query containment under constraints to the classical query containment problem without constraints (cf. Theorem 20). Therefore, the search for more efficient query plans is supported. The chase has also become a central tool in semantic query optimization [Popa and Tannen, 1999; Deutsch et al., 2006; Popa et al., 2000; Schmidt et al., 2008; Amer-yahia et al., 2002; Calì and Martinenghi, 2008]. It was shown in [Popa et al., 2000] that the Chase & Backchase procedure (cf. Section 4.2) using the standard chase can be efficiently implemented and can cause significant time reductions in query answering. We also want to mention that there are many subareas in query optimization which do not use the chase. In [Afrati et al., 2003] containment of conjunctive queries with arithmetic comparisons is discussed. Containment of CQ with safe negation and disjunction is treated in [Lausen and Wei, 2003; Wei and Lausen, 2002b,a]. Query containment was also considered for F-Logic [Kifer et al., 1995] queries [Calì and Kifer, 2006]. Efficient algorithms for query containment is a topic in [Chekuri and Rajaraman, 1997; Wei and Lausen, 2008]. The complexity of query answering is investigated in [Vardi, 1982; Papadimitriou and Yannakakis, 1997]. Efficient query evaluation techniques are discussed in [Yannakakis, 1981; Aho et al., 1979; Biskup et al., 1995; Gottlob et al., 2001; Flum et al., 2002].

Beyond query optimization, the chase has been applied in data exchange [Fagin et al., 2005] (cf. Section 4.3), which can be understood as the execution of the chase with source-to-target plus target constraints on the underlying data instance. It was shown that no matter what chase sequence is taken, answering conjunctive queries on the result always delivers the same results. As a unique representative of the set of solutions, in [Fagin et al., 2005] it was argued that *cores* are the most natural ones because they are small and intuitive to understand. In [Gottlob and Nash, 2008] it was shown that they can be computed in polynomial time data complexity in case the constraint set is weakly acyclic.

Beyond data exchange, it has been applied in many other contexts such as peer data exchange [Fuxman et al., 2005], data integration [Lenzerini, 2002; Calì, 2004], query answering using views [Halevy, 2001; Deutsch et al., 2007], and probabilistic databases [Olteanu et al., 2009].

10.5 Semantic Query Optimization in the Presence of Types

Typing is a central component of many practical database systems, including (but not limited to) relational databases, object-oriented database models [Kifer et al., 1995; Papakonstantinou et al., 1995], typed datalog [Zook et al., 2009], and semi-structured data [Milo and Suciu, 1999]. In response, to date a rich theory of type-based optimization has been developed [Frühwirth et al., 1991; Litwin and Risch, 1992; Levy and Suciu, 1997; Gallagher and Puebla, 2002; Henriksson and Maluszynski, 2004; Bruynooghe et al., 2005]. These optimization approaches often use type inference algorithms and have a background in the world of programming languages (cf. [de Moor et al., 2008; Schäfer and de Moor, 2010]).
To this aim, several authors have studied encoding the typing knowledge, mainly with the following two formalisms: Datalog (and extensions) [Chan, 1992; Dong and Su, 1996] and Description Logics (DL) [Calvanese et al., 2007, 2008]. However, the lack of value creation (which can be captured by TGDs) in Datalog prevents it from being a suitable candidate for modeling typing knowledge. On the other hand, although DL (and its fragments) can capture TGDs, it is unable to express functional dependencies like integrity constraints. Recently, Calì et al. [Calì et al., 2009] have proposed *Datalog*$^\pm$, to extend plain Datalog with guarded TGDs and stratified negation. Yet, *Datalog*$^\pm$ can not express disjunctive rules.
Our results extend the work in [Schmidt et al., 2010] on semantic query optimization for RDF and SPARQL. The type hierarchy that we encounter in the definition of RDF can be easily expressed using our framework because we only have to assign a type for the subject, predicate and objects positions and state the disjointness of URIs U, blank nodes B, and literals L. More precisely, we can model these relationships with the help of the constraints

$T(x_1, x_2, x_3) \rightarrow U(x_1) \vee B(x_1)$,
$T(x_1, x_2, x_3) \rightarrow U(x_2)$,
$T(x_1, x_2, x_3) \rightarrow U(x_3) \vee B(x_3) \vee L(x_3)$,
$U(x) \rightarrow \neg B(x) \wedge \neg L(x)$,
$B(x) \rightarrow \neg U(x) \wedge \neg L(x)$, and
$L(x) \rightarrow \neg B(x) \wedge \neg U(x)$.

10.6 Rule-Based Minimization

The RDF standard [World Wide Web Consortium, 2003a] is a fragment of F-Logic [Kifer et al., 1995]. Our work on rule-based minimization extends the works on transitive reduction, see [Aho et al., 1972], in graph theory. Yet, in our framework it is not only possible to eliminate transivities in graphs, but to define general rules for the reduction. Constraints on the reduced graph are also not considered in [Aho et al., 1972].

The problem of optimizing the amount of data that must be exchanged if a graph is updated and the update must be transferred to other hosts was already studied in [Zeginis et al., 2007] using the notion of deltas. We are not aware of any work on RDF that provides user-specific minimization techniques the way we do.

The work in [Iannone et al., 2005; Esposito et al., 2005] eliminates triples in the sense of lean graphs and homomorphic equivalence. Although the results that are produced are homomorphically equivalent to the original graph, they are not necessarily lean. Our work does not subsume [Iannone et al., 2005; Esposito et al., 2005] nor the other way round. We focus on different aspects of redundancy elimination. In our work, it must be explicitly specified via rules what a redundancy structurally looks like. For example, our method may also be suitable when the graph to be minimized has a user-defined rule semantics in the sense of [ter Horst, 2005], where it could be desirable to minimize a given graph along its rule semantics. The notion of lean is orthogonal to derivability by application-specific rules. If such rules exist, a lean graph may still contain triples which are redundant in the sense that they need not be explicitly stored because they could be derived by the rules as well. Minimization in the sense of lean graphs is always possible because this method is generic. For example in the RDF graph

$$\{\mathtt{T}(a_1, a_2, a_3),\mathtt{T}(X, a_2, Y)\}$$

the triple $\mathtt{T}(X, a_2, Y)$ can be eliminated (X, Y are blank nodes) because both X and Y are treated like existentially quantified variables in the RDF semantics [World Wide Web Consortium, 2003c] and the triple $\mathtt{T}(a_1, a_2, a_3)$ is a witness for the existence of such a resource $\mathtt{T}(X, a_2, Y)$.

We also want to mention that MINI-RDF is not an abduction problem in the sense of [Eiter et al., 1997]. An abduction problem is characterized as follows: if we observe C and have a rule of the form $C \leftarrow A$, then we can conclude the premise A, regardless whether A follows from our fact base or not. In our scenario, we would only delete C from our fact base if afterward A still follows from it. Knowledge assimilation in deductive databases [Decker, 1998] uses abduction techniques to rewrite updates in such a way that they can be performed on the extensional data only. Here it is already known in advance which data is extensional and which data is intensional. In a certain sense, in our scenario we want to compute the extensional part and delete all intensional data such that the constraints remain satisfied.

Chapter 11

Conclusions and Future Work

> **Riccardo:** “Is that all?”
> **Alice:** “Yes, except for some final remarks.”

We want to finish this thesis with a short high-level summary of our main results and some directions for future work.

11.1 Conclusions

Termination of the chase algorithm is a fundamental problem of interest to both database theory and practice. When it terminates, the chase is an all-purpose tool that can be used for semantic query optimization [Deutsch et al., 2006], certain answer computation in data integration [Lenzerini, 2002], constraint implication, query answering using views [Halevy, 2001], and probabilistic databases [Olteanu et al., 2009]. Recently, interest in the chase has been rekindled by its applications to the area of data exchange [Fagin et al., 2005; Gottlob and Nash, 2008]. The results in this thesis are extensions of the state of the art regarding sufficient termination conditions. They are therefore contributions to all above research sub-areas. Although theoretical by nature our results are of immediate practical interest.

As opposed to prior work, which used to put forward one new termination condition per paper that improved on the best condition so far, this thesis identifies four different flavors of termination and finds a hierarchy of such conditions for each flavor. Checking our conditions is not more expensive than the previous least restrictive ones. While checking the conditions may take exponential time, as for $\forall\forall$-T$[k]$, a coNP upper bound is given, this is only in the size of the constraints, and moreover, it is a test carried out offline, once and for all when the constraints are declared.

Two flavors of chase termination are related to data-independent techniques. We propose new techniques that allow us to guarantee chase termination on all instances and for all chase sequences. Furthermore, we prove that stratification does not guarantee chase termination on all instances and for all chase sequences but on all instances and for at least one chase sequence, thus entering the new area of sequence-dependent termination conditions.

We take one step further to the case of data-dependent techniques when termination on all instances and for all chase sequences or at least one chase sequence cannot be guaranteed statically, looking into guarantees for a given instance and for the termination of at least one chase sequence or all chase sequences. We also look into cases where no static termination guarantee can be made, proposing to nevertheless start the chase optimistically but monitor it at runtime to detect configurations when the divergence danger is high, prompting the cut-off of the chase sequence. These are all tools for pragmatic work-arounds in response to the undecidability of termination.

As a possible application scenario we have looked at semantic query optimization in the presence of types. In previous investigations semantic query optimization and optimizations based on complex type hierarchies have been studied separately although both topics have striking commonalities. Unifying these two research areas, we have developed a logical framework that seamlessly integrates both techniques and, in the general case, provides better optimization results than their application in two isolated, subsequent stages. We also provided algorithms to enumerate optimized queries as well as results on the complexity of related decision and satisfiability problems. The applicability of our method depends on chase termination, and in response we proposed novel termination conditions in the presence of negation and disjunction.

As a further application scenario we have studied rule-based minimization under constraints.

11.2 Perspectives

Topics related to the chase are a very active area of research and there are several challenging open questions left for future work. We list some of them here that are directly related to this thesis.

- Are $\mathrm{CT}_{\forall\forall}$ and $\mathrm{CT}_{\forall\exists}$ undecidable?
- Is $\bigcup_{i\geq 2}\forall\forall\text{-T}[i]$ decidable?
- Does it hold, for a fixed set of TGDs and EGDs, that membership in $\mathrm{CT}_{\forall\forall}$ or $\mathrm{CT}_{\forall\exists}$ always guarantees termination of the chase in polynomially many steps in the input database?
- Are there specific termination conditions for the core chase?

- Can the core chase be efficiently implemented?
- Is semantic query optimization in the presence of types possible for more expressive query languages? For example if we allow aggregates and an extended use of negation.

Bibliography

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

Foto Afrati, Chen Li, and Prasenjit Mitra. On Containment of Conjunctive Queries with Arithmetic Comparisons (Extended Version). In *EDBT*, pages 459–476, 2003.

Alfred V. Aho, Michael R. Garey, and Jeffrey D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

Alfred V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Efficient Optimization of a Class of Relational Expressions. *ACM Trans. Database Syst.*, 4(4):435–454, 1979.

Sihem Amer-yahia, Sungran Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. Tree Pattern Query Minimization. *VLDB Journal*, 11(4):315–331, 2002.

Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 2009.

Catriel Beeri and Moshe Y. Vardi. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, pages 34–43, May 2001.

Joachim Biskup, Pratul Dublish, and Yehoshua Sagiv. Optimization of a Subclass of Conjunctive Queries. *Acta Inf.*, 32(1):1–26, 1995.

Maurice Bruynooghe, John P. Gallagher, and Wouter Van Humbeeck. Inference of Well-Typings for Logic Programs with Application to Termination Analysis. In *SAS*, pages 35–51, 2005.

Peter Buneman, Wenfei Fan, and Scott Weinstein. Interaction between path and type constraints. *ACM Trans. Comput. Logic*, 4(4):530–577, 2003. ISSN 1529-3785. doi: http://doi.acm.org/10.1145/937555.937560.

Andrea Calì. Query Answering by Rewriting in GLAV Data Integration Systems Under Constraints. In *SWDB*, pages 167–184, 2004.

Andrea Calì and Michael Kifer. Containment of Conjunctive Object Meta Queries. In *VLDB*, pages 942–952, 2006.

Andrea Calì and Davide Martinenghi. Conjunctive Query Containment under Access Limitations. In *ER*, pages 326–340, 2008.

Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *KR*, pages 70–80, 2008.

Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog+-: A Unified Approach to Ontologies and Integrity Constraints. In *ICDT*, pages 14–30, 2009.

Diego Calvanese, Giuseppe Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *J. Autom. Reason.*, 39(3):385–429, 2007.

Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive Query Containment and Answering under Description Logic Constraints. *ACM Trans. Comput. Logic*, 9(3):1–31, 2008.

Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based Approach to Semantic Query Optimization. *ACM Trans. Database Syst.*, 15(2):162–207, 1990.

Edward P. F. Chan. Containment and Minimization of Positive Conjunctive Queries in OODB's. In *PODS*, pages 202–211, 1992.

Ashok K. Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Databases. In *STOC*, pages 77–90, 1977.

Chandra Chekuri and Anand Rajaraman. Conjunctive Query Containment Revisited. In *ICDT*, pages 56–70, 1997.

Peter Pin-Shan Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

Qi Cheng, Jarek Gryz, Fred Koo, T. Y. Cliff Leung, Linqi Liu, Xiaoyan Qian, and K. Bernhard Schiefer. Implementation of Two Semantic Query Optimization Techniques in DB2 Universal Database. In *VLDB*, pages 687–698, 1999.

Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, 1970.

Edgar F. Codd. Further Normalization of the Data Base Relational Model. *IBM Research Report, San Jose, California*, RJ909, 1971.

Edgar F. Codd. Recent Investigations in Relational Data Base Systems. In *IFIP Congress*, pages 1017–1021, 1974.

Edgar F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.

Stavros S. Cosmadakis and Paris C. Kanellakis. Functional and Inclusion Dependencies. *Advances in Computing Research*, 3:163–184, 1986.

Stéphane Coulondre. A Top-down Proof Procedure for Generalized Data Dependencies. *Acta Inf.*, 39(1):1–29, 2003.

Oege de Moor, Damien Sereni, Pavel Avgustinov, and Mathieu Verbaere. Type Inference for Datalog and its Application to Query Optimisation. In *PODS*, pages 291–300, 2008.

Hendrik Decker. Some Notes on Knowledge Assimilation in Deductive Databases. In *ILPS*, pages 249–286, 1998.

Alin Deutsch and Alan Nash. First Order Modeling of Integrity Constraints (DB Encyclopedia Entry). Springer-Verlag, 2008a.

Alin Deutsch and Alan Nash. Chase (DB Encyclopedia Entry). Springer-Verlag, 2008b.

Alin Deutsch and Val Tannen. MARS: A System for Publishing XML from Mixed and Redundant Storage. In *VLDB*, pages 201–212, 2003.

Alin Deutsch and Val Tannen. XML Queries and Constraints, Containment and Reformulation. *Theor. Comput. Sci.*, 336(1):57–87, 2005.

Alin Deutsch, Lucian Popa, and Val Tannen. Query Reformulation with Constraints. *SIGMOD Rec.*, 35(1):65–73, 2006.

Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting Queries Using Views with Access Patterns under Integrity Constraints. *Theor. Comput. Sci.*, 371(3): 200–226, 2007.

Alin Deutsch, Alan Nash, and Jeff Remmel. The Chase Revisited. In *PODS*, pages 149–158, 2008.

Guozhu Dong and Jianwen Su. Conjunctive Query Containment with respect to Views and Constraints. *Inf. Process. Lett.*, 57(2):95–102, 1996.

Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic*. Springer, second edition edition, 1996.

Thomas Eiter, Georg Gottlob, and Nicola Leone. Abduction from Logic Programs: Semantics and Complexity. *Theor. Comput. Sci.*, 189(1-2):129–177, 1997.

Floriana Esposito, Luigi Iannone, Ignazio Palmisano, Domenico Redavid, and Giovanni Semeraro. REDD: An Algorithm for Redundancy Detection in RDF Models. pages 138–152. 2005.

Ronald Fagin. Horn Clauses and Database Dependencies. *J. ACM*, 29(4):952–985, 1982.

Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

Wenfei Fan and Leonid Libkin. On XML Integrity Constraints in the Presence of DTDs. *J. ACM*, 49(3):368–406, 2002.

Wenfei Fan and Jérôme Siméon. Integrity Constraints for XML. In *PODS*, pages 23–34, 2000.

Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. Query Optimization in the Presence of Limited Access Patterns. In *SIGMOD*, pages 311–322, 1999.

Jörg Flum. Script to a lecture in model theory. http://home.mathematik.uni-freiburg.de/flum/ss06mod/skript.ps. 2002.

Jörg Flum and Martin Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., 2006.

Jörg Flum, Markus Frick, and Martin Grohe. Query Evaluation via Tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

Thom W. Frühwirth, Ehud Y. Shapiro, Moshe Y. Vardi, and Eyal Yardeni. Logic Programs as Types for Logic Programs. In *LICS*, pages 300–309, 1991.

Ariel Fuxman, Phokion G. Kolaitis, Renée J. Miller, and Wang-Chiew Tan. Peer Data Exchange. In *PODS*, pages 160–171, 2005.

John P. Gallagher and German Puebla. Abstract Interpretation over Non-deterministic Finite Tree Automata for Set-Based Analysis of Logic Programs. In *PADL*, pages 243–261, 2002.

Georg Gottlob and Alan Nash. Efficient Core Computation in Data Exchange. *J. ACM*, 55(2):1–49, 2008.

Georg Gottlob, Nicola Leone, and Francesco Scarcello. The Complexity of Acyclic Conjunctive Queries. *J. ACM*, 48(3):431–498, 2001.

Claudio Gutierrez, Carlos Hurtado, and Alberto Mendelzon. Formal Aspects of Querying RDF Databases. In *SWDB*, pages 293–307, 2003.

Claudio Gutierrez, Carlos Hurtado, and Alberto O. Mendelzon. Foundations of Semantic Web Databases. In *PODS*, pages 95–106, 2004.

Alon Y. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, 10 (4):270–294, 2001.

Michael M. Hammer and Dennis J. McLeod. Semantic Integrity in a Relational Data Base System. In *VLDB*, pages 25–47, 1975.

Jakob Henriksson and Jan Maluszynski. Static Type-Checking of Datalog with Ontologies. In *PPSWR*, pages 76–89, 2004.

Luigi Iannone, Ignazio Palmisano, and Domenico Redavid. Optimizing RDF Storage Removing Redundancies: An Algorithm. In *IEA/AIE*, pages 732–742, 2005.

David S. Johnson and Anthony Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. In *PODS*, pages 164–169, 1982.

Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-oriented and Frame-based Languages. *J. ACM*, 42(4):741–843, 1995.

Jonathan J. King. QUIST: A System for Semantic Query Optimization in Relational Databases. *Distributed systems, Vol. II: distributed data base systems*, pages 287–294, 1986.

Phokion G. Kolaitis, David L. Martin, and Madhukar N. Thakur. On the Complexity of the Containment Problem for Conjunctive Queries with Built-in Predicates. In *PODS*, pages 197–204, 1998.

Georg Lausen and Fang Wei. On the Containment of Conjunctive Queries. *Computer Science in Perspective: Essays Dedicated to Thomas Ottmann*, pages 231–244, 2003.

Georg Lausen, Michael Meier, and Michael Schmidt. SPARQLing Constraints for RDF. In *EDBT*, pages 499–509, 2008.

Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

Alon Y. Levy and Dan Suciu. Deciding Containment for Queries with Complex Objects (Extended Abstract). In *PODS*, pages 20–31, 1997.

Witold Litwin and Tore Risch. Main Memory Orientated Optimization of OO Queries Using Typed Datalog with Foreign Predicates. *IEEE Trans. on Knowl. and Data Eng.*, 4(6):517–528, 1992.

Michael J. Maher and Divesh Srivastava. Chasing Constrained Tuple-generating Dependencies. In *PODS*, pages 128–138, 1996.

David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

Bruno Marnette. Generalized Schema-Mappings: From Termination to Tractability. In *PODS*, pages 13–22, 2009.

Michael Meier. Towards Rule-Based Minimization of RDF Graphs under Constraints. In *RR*, pages 89–103, 2008.

Michael Meier, Michael Schmidt, and Georg Lausen. Stop the Chase, Technical Report. *CoRR*, abs/0901.3984, 2009a.

Michael Meier, Michael Schmidt, and Georg Lausen. Stop the Chase, Extended Abstract. *AMW*, 2009b.

Michael Meier, Michael Schmidt, and Georg Lausen. On Chase Termination Beyond Stratification. *CoRR*, abs/0906.4228, 2009c.

Michael Meier, Michael Schmidt, and Georg Lausen. On Chase Termination Beyond Stratification. *PVLDB*, 2(1):970–981, 2009d.

Michael Meier, Michael Schmidt, Fang Wei, and Georg Lausen. Semantic Query Optimization in the Presence of Types. *PODS*, 2010. To appear.

Tova Milo and Dan Suciu. Type Inference for Queries on Semistructured Data. In *PODS*, pages 215–226, 1999.

Dan Olteanu, Jiewen Huang, and Christoph Koch. SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases. In *ICDE*, pages 640–651, 2009.

Christos H. Papadimitriou and Mihalis Yannakakis. On the Complexity of Database Queries (Extended Abstract). In *PODS*, pages 12–19, 1997.

Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object Exchange Across Heterogeneous Information Sources. In *ICDE*, pages 251–260, 1995.

Lucian Popa. *Object/Relational Query Optimization with Chase and Backchase.* PhD thesis, University of Pennsylvania, 2000.

Lucian Popa and Val Tannen. An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In *ICDT*, pages 39–57, 1999.

Lucian Popa, Alin Deutsch, Arnaud Sahuguet, and Val Tannen. A Chase Too Far? In *SIGMOD*, pages 273–284, 2000.

Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering Queries Using Templates with Binding Patterns (Extended Abstract). In *PODS*, pages 105–112, 1995.

Max Schäfer and Oege de Moor. Type Inference for Datalog with Complex Type Hierarchies. In *POPL*, 2010. To appear.

Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL Query Optimization. *CoRR*, abs/0812.3788, 2008.

Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL Query Optimization. In *ICDT*, 2010. To appear.

Larry J. Stockmeyer. The Polynomial-Time Hierarchy. *Theor. Comput. Sci.*, 3: 1–22, 1976.

Herman J. ter Horst. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *ISWC*, pages 668–684, Heidelberg, 2005.

David Toman and Grant E. Weddell. On Path-functional Dependencies as First-class Citizens in Description Logics. In *Description Logics*, 2005.

Moshe Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*, pages 137–146, 1982.

Dubois Vincent and Bothorel Cecile. Transitive Reduction for Social Network Analysis and Visualization. In *WI*, pages 128–131, 2005.

Fang Wei and Georg Lausen. Conjunctive Query Containment in the Presence of Disjunctive Integrity Constraints. In *Description Logics*, 2002a.

Fang Wei and Georg Lausen. Containment of Conjunctive Queries with Safe Negation. In *ICDT*, pages 346–360, 2002b.

Fang Wei and Georg Lausen. A Unified Apriori-like Algorithm for Conjunctive Query Containment. In *IDEAS*, pages 111–120, 2008.

World Wide Web Consortium. W3C XML Query (XQuery), 2000. `http://www.w3.org/XML/Query/`. W3C Recommendation, November 10, 2009.

World Wide Web Consortium. Resource Description Framework (RDF): Concepts and Abstract Syntax, 2003a. `http://www.w3.org/TR/rdf-concepts/`. W3C Recommendation, February 10, 2004.

World Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema, 2003b. `http://www.w3.org/TR/rdf-schema/`. W3C Recommendation, February 10, 2004.

World Wide Web Consortium. RDF Semantics, 2003c. `http://www.w3.org/TR/rdf-mt/`. W3C Recommendation, February 10, 2004.

World Wide Web Consortium. W3C Extensible Markup Language (XML), 2003d. `http://www.w3.org/XML/`. W3C Recommendation, April 16, 2009.

Mihalis Yannakakis. Algorithms for Acyclic Database Schemes. In *VLDB*, pages 82–94, 1981.

Dimitris Zeginis, Yannis Tzitzikas, and Vassilis Christophides. On the Foundations of Computing Deltas Between RDF Models. In *ISWC/ASWC*, pages 637–651, 2007.

Xubo Zhang and Z. Meral Ozsoyoglu. On Efficient Reasoning with Implication Constraints. In *DOOD*, pages 236–252, 1993.

David Zook, Emir Pasalic, and Beata Sarna-Starosta. Typed Datalog. In *PADL*, pages 168–182, 2009.

Index