A KNOWLEDGE-LEVEL VIEW OF CONSISTENT QUERY ANSWERS

by

Eric Evangelista

Honours BA, York University, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
September 2004

APPROVAL

Name: Eric Evangelista

Degree: Master of Science

Title of thesis: A Knowledge-Level View of Consistent Query Answers

Examining Committee: Dr. Eugenia Ternovska
Chair

Dr. James P. Delgrande, Senior Supervisor

Dr. Ke Wang, Supervisor

Dr. Leopoldo Bertossi, External Examiner
School of Computer Science
Carleton University

Date Approved: September 7th, 2004

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENCE

Abstract

Of the numerous formal approaches that deal with database inconsistencies with respect to integrity constraints, all share the view that such constraints are statements about the world the database models. An alternative perspective, however, considers constraints as statements about the knowledge the database has of its domain. The result of this shift in perspective allows us to regard integrity constraint violations as a fragment of the incomplete knowledge the system has of the world. We can then query the possibly inconsistent database for *consistent query answers.*

We address the above considerations with an epistemic query language $\mathcal{KL}$, where the possible ways to *repair* a database that violates its integrity constraints are characterized by a set of possible worlds, an *epistemic state* e_C. This culminates in a situation where only consistent information is known. We ascertain this by querying e_C with $\mathcal{KL}$, providing a knowledge-level formalization of consistent query answers. At the outset, we show that $\mathcal{KL}$ is an adequate language for querying databases by specifying a class of *admissible* formulas for which the set of answers to such queries are *safe* and *domain independent.* After formulating *database dependencies* in $\mathcal{KL}$, we prove that they are members of this class. A Prolog-like sound and complete query evaluator, *cqa*, for *admissible* $\mathcal{KL}$ formulas is presented. Finally, we completely characterize what is known in e_C with a set of first-order sentences.

The investigation of the truth is in one way hard, in another easy. An indication of this is found in the fact that no one is able to attain the truth adequately, while, on the other hand, no one fails entirely, but everyone says something true about the nature of things, and while individually they contribute little or nothing to the truth, by the union of all a considerable amount is amassed.

- Aristotle, *Metaphysics*

To my parents,
without whom not

Acknowledgments

My worn, extensively annotated and dog-eared copy of Hector Levesque and Gerhard Lakemeyer's wonderful *The Logic of Knowledge Bases* is a testament to the sheer exhilaration and enjoyment I had in studying their ideas. Their book is so rich with insights that I had not even read past Part One before I wanted to see their logic $\mathcal{KL}$ *in action.* And it was Jim Delgrande who gave me this opportunity not only by introducing me to $\mathcal{KL}$ but also by supplying the ideas that made this thesis a *thesis.*

Whenever the frustrations of research threatened to discourage my attitude or hinder my progress, Jim's suggestions were always refreshing and served either to clarify issues where I had been perplexed and inaccurate or to discharge any doubts that what I was doing was, in fact, interesting. Because of his rigorous and open-minded approach to scholarship, I not only understand more about the issues surrounding knowledge representation, I understand them *better.*

The following students, faculty and researchers made my time at SFU intellectually rewarding and fun: Leopoldo Bertossi helped clarify the logic of my presentation and provided insights that significantly enriched my understanding of consistent query answering; Ke Wang encouraged me to look at the practical motivations of my work and highlighted areas of my approach that were unclear; Eugenia Ternovska revealed aspects of standard names that I had barely grasped; Jeff Pelletier suggested a connection between work on vagueness and unknown instances; Guilin Qi suggested improvements to my first two chapters and helped me understand various aspects of inconsistency handling in knowledge bases; Michael Letourneau exposed a perspective on null constants that I had not considered; Mayu Ishida acted as co-conspirator in my forays into the

fertile and lucid territory of mathematical logic.

I must also extend my gratitude to the staff and faculty in the School of Computing Science at SFU and to the members of its Computational Logic Laboratory for providing an environment conducive to world-class research and interdisciplinary collaboration.

As always, friends and family provided the moral support and laughter to make everything worthwhile.

The research conducted in this thesis was financially supported by Simon Fraser University and its School of Computing Science.

Contents

Chapter 1

Introduction

> *[T]he computer is to be envisioned as obtaining data on which it is to base its inferences from a variety of sources, all of which indeed may be supposed to be on the whole trustworthy, but none of which can be assumed to be that paragon of paragons, a universal truth-teller... the essential feature is that there is no single, monolithic, infallible source of the computer's data, but that inputs come from several independent sources. In such circumstances* the *crucial feature of the situation emerges:* inconsistency threatens.
>
> - Nuel D. Belnap, "How A Computer Should Think"

The need to specify legal database states and relationships between values in a tuple for the relational model of databases has led to the study of integrity constraints and, particularly, data dependencies [1, 23, 52]. While the standard database management system (DBMS) supports transactions where inconsistent updates are rejected, the violation of these constraints is a recurring problem for systems that utilize information stored in distributed "islands of information", where the constraints are locally satisfied but (possibly) globally violated [15, 18, 49]. According to Stonebraker [49], the goal of these systems is to enrich the information available for decision making and to ensure that consistency is maintained across the various sources. The problem that emerges, however, is that these two goals are sometimes at odds with each other.

For example, information in data warehouses gives access to historical and/or

aggregated data that has been compiled from an organization's operational databases [18]. As the data is extracted from the operational sources, data is *cleaned* to ensure consistency with a set of constraints [34]. While there are algorithms to handle common errors [42], Greenfield [25] indicates that checking for errors consumes more time than cleaning data. Handling inconsistent data is thus left to the application developer, a task that is formidable should either the amount of data be large [22] or should the developer not have the domain knowledge to isolate relevant information. As a result of this, Mallach [34] explains that inconsistencies often frustrate decision makers, exacerbating the above noted tension between maintaining consistency while retaining as much of the source information as possible.

Multidatabases, on the other hand, transform a user's global query into a plan that resolves any schematic heterogeneity with autonomous data sources and checks the integrity of the query with respect to a set of global constraints, constraints that are possibly different from those defined at the local sites. To preserve local autonomy, these global constraints are not enforced at the local level, thereby retaining all information locally while filtering inconsistencies globally [4, 14, 15]. Thus, at any one time the global constraints may not be satisfied.

To be sure, industrial applications have a broader range of practical issues to deal with, but this dilemma nonetheless frames the various *formal* approaches for modeling and providing a semantics for relational database inconsistencies. Since the problem can be phrased at the level of the content of relational databases and not their implementation, these formal methods allow researchers and developers to think about the problem at a different, and arguably 'natural', level [16]. We thus represent *database instances* as finite sets of atomic sentences in some first-order language, or *database schema*, and view query evaluation as a form of logical entailment or deduction [43]. And the goal of the formal methods we discuss and use aim to handle *extensional inconsistency* as opposed to *intensional inconsistency* [38].

Intensional inconsistency deals with the resolution of different ontologies and schemas across various sources [32, 53]. This is a vexing problem in itself, but we assume a uniform ontology and database schema and consider the equally

challenging problem of extensional inconsistency. This simplifying assumption allows us to consider various database instances as a single instance [4, 8, 10], which reinforces the view that inconsistencies arise as a fragment of the global nature of the system. This is the perspective from which we view the problem of extensional inconsistencies; we hereafter refer to such global database instances as 'database instances' or 'databases'.

Extensional inconsistency involves the truth of conflicting sentences in the database with respect to a set of integrity constraints. Consider the constraint "Social security numbers of employees are unique". Suppose that an employee database of Acme Inc. in the City X has an employee named Jane Smith whose social security number is '123'. Jane Smith in City Y has a social security number '456' and also works for Acme. The individual employee databases *satisfy* the constraint, while the system violates it if we view the sources from a global perspective. Ultimately, we do not want to lose either piece of information but we also want to know that the constraint is violated, that an extensional inconsistency exists. This poses problems for regarding query evaluation as entailment since all sentences are entailed by an inconsistent instance.

What we propose is to treat consistent information as data that our system *knows*. We can ask "Does Jane have a social security number?" or "Is it possible that Jane has a social security number?" to which the system *should* respond with "Yes", and "Do you *know* Jane's social security number?", to which the system should answer "No"; in the *worlds* that the system considers possible, Jane has different social security numbers at different worlds and there is no *known* number for Jane that is common to all these states of affairs. So asking "Is Jane's social security number 123?" should result in the answer "I don't know." The system does not have any further information to prefer either social security number (ssn) for Jane. It tolerates uncertainty regarding Jane's ssn but the constraint is nonetheless satisfied since there is no known data that violates the constraint.

Adopting this approach is intuitive for a number of reasons. In the face of conflicting information, the system believes exactly what it is told to believe while enforcing the integrity constraints. Essentially, it treats integrity constraint violations as a fragment of the incomplete knowledge it has of the world;

additional information can expel this incompleteness. Until then, however, different states of affairs are possible and we can minimize information loss. Jane can either have '123' as her social security number or '456' but not both. Furthermore, information that is not involved in the violation of the constraints are quarantined from inconsistent data. The social security numbers of other employees, employees' salaries, etc. are still *known*. This is ascertained by querying the system about its knowledge. We call information a *consistent query answer* [4] if and only if it holds across all possible states of affairs. The system's understanding of the world is thus *textured* since we can ask about known, unknown or possibly known facts.

But why focus on querying the inconsistent system? Why not simply fix the problem before any querying whatsoever? As noted above, in contexts such as data warehousing, cleaning data is performed by an automated process and the problem of inconsistency is left in the hands of the application programmer who does not necessarily need specialized knowledge of the application domain to isolate data inconsistencies. Information is lost as a result since such errors should be dealt with by decision makers [34]. The proposed approach allows us to distinguish consistent from inconsistent information; to establish which data is in fact inconsistent, some querying will have to occur [4, 16]. In the context of multidatabases, this approach respects the local autonomy of distributed sources by not enforcing local consistency on global constraints [14]. The task therefore is to tolerate the inconsistencies until further information allows us to isolate and remove inconsistent data; this is what *consistent query answering* allows us to do.

What we present in this thesis is a formalization of a notion of *consistent query answers* in Levesque's [29] epistemic logic $\mathcal{KL}$, therefore providing a query language within which to reason about examples such as those presented above. Intuitively, a consistent query answer is an answer that holds in every *repair* of a possibly inconsistent database. We characterize these repairs as a set of possible worlds, or an *epistemic state* that represents the current state of knowledge of the database. We call this epistemic state e_C. A query answer that is consistent with the integrity constraints is then understood to hold in all possible worlds in e_C; it is *known*. We can then query the database with $\mathcal{KL}$ and ask what the

possibly inconsistent database knows, possibly knows or does not know. This provides a perspective of consistent query answers characterized by the knowledge of the system, a *knowledge-level* view [39].

We first need to refine many of the notions concerning *knowledge bases* to the special requirements of *databases*. This is the work of Chapters 3 and 4. In Chapter 3 we introduce Levesque's $\mathcal{KL}$ [29, 30] as a query language for first-order databases by ensuring finite answers, or *safety*, and *domain independence*. We call this class of formulae *admissible*. Chapter 4 reviews the work of Reiter [44] and shows how $\mathcal{KL}$ reinforces his understanding of integrity constraints as statements about the knowledge the system has of the world. We provide an extended discussion on the *allowed* form of integrity constraints in $\mathcal{KL}$ and isolate an important class of constraints called *database dependencies* [52]; we show that they can be expressed as admissible and allowed formulas in $\mathcal{KL}$. Chapters 3 and 4 therefore investigate $\mathcal{KL}$'s utility as a database query language and as a logic for integrity constraints.

Following this in Chapter 5, we define the epistemic state e_C and the notion of a *consistent query answer* in $\mathcal{KL}$. And unless we apply the *closed-world assumption* on the repairs, *tuple-generating dependency* violations lead to information loss. We extend the applicability of consistent query answering as conceived by Arenas et al. [4]. These considerations show how to effectively query e_C with a variation of Reiter's [45] query evaluator for $\mathcal{KL}$, which we call *cqa*. This evaluator is sound and complete with respect to e_C and admissible formulas.

Chapter 6 provides an analysis of the knowledge of e_C and characterizes the conditions under which something is known by formally investigating some properties of repairs with respect to a set of dependencies; this leads to a representation theorem for e_C. We then summarize our work and discuss directions for further research.

Chapter 2 explores existing work in the area and contextualizes the contribution of this thesis, which is a non-trivial application of $\mathcal{KL}$ in the context of handling database inconsistencies by formalizing the concept of consistent query answer.

Chapter 2

Literature Review

> *True stability results when presumed order and presumed disorder are balanced. A truly stable system expects the unexpected, is prepared to be disrupted, waits to be transformed.*
>
> - Tom Robbins

Copious philosophical and mathematical accounts exist discussing the nature of acquiring new information. It is not that a reasoning agent needs to simply *store* information indifferently, and that researchers are solely concerned with the myriad ways of doing this, a major problem is understanding how an agent should adjust itself with respect to how new information fits with the rest of what it knows. It is entirely possible that new information conflicts, overlaps or is irrelevant with what is already known. What does an agent choose to believe? Belnap [11] suggests that an agent should believe *exactly what it is told to believe while enforcing the integrity constraints*, even if what it is told conflicts with what it knows. This is the approach taken in this thesis. Thus it is worthwhile surveying related work that adheres to this assumption, along with those approaches that reject it.

We distinguish two categories that attempt to deal with inconsistencies in knowledge-based systems [8, 9]: paraconsistent-based and coherence-based. Paraconsistent-based methods use a parconsistent formalism to identify inconsistent information. One can then either remove inconsistencies or retain them. In the latter case, inconsistencies are accessible *as* inconsistent information. In

the former case, inconsistent information is first pinpointed and then removed. Coherence-based methods, on the other hand, use meta-logical techniques that attempt to restore consistency at the expense of losing (arguably) useless information without necessarily identifying the inconsistencies. In this chapter, we also explore approaches that attempt to harmonize the goals of maintaining information and ensuring consistency. These methods focus on *querying* inconsistent knowledge bases. This has the benefit of viewing knowledge-based systems functionally; the original information is not altered and only consistent *answers* are returned. How this relates to approaches dealing with *incomplete information* is likewise discussed.

This categorization is not meant to fully articulate the subtle and important differences between the approaches, nor is it meant to indicate that their goals are pairwise similar, but is intended merely to highlight the techniques and strategies of the formal methods used.

2.1 Paraconsistent Approaches

In [11, 12], Belnap discusses the utility of a four-valued logic for a computerized question-answering system. If we use a two-valued logic, he argues, the trivialization of entailment is a possibility. The system could be told that the Blue Jays won the Series in 1992 *and* that the Braves won the Series in the same year, for example. Assuming the system knows that only one team can win the Series per year, inconsistency emerges and threatens meaningful entailment. Since we do not want this inconsistent information to "pollute" the system's knowledge of crowd attendance, batting averages, runs-batted-in, etc., we can reject either fact. However, we assume that the sources of information are equally trustworthy in the sense that we have no reason to believe that they relate falsehoods. Alternatively, we have no reason to prefer any piece of information. So the computer should not be expected to do something other than report what it has been told; Belnap thus proposes a four-valued logic for this problem.

The idea is that the computer will code each fact it is told with one of four truth values at a particular point in time. So the tuple $\langle Blue\ Jays, 1992\rangle$ would be marked with *Both*, indicating that the computer was told it is true and that

it was told it is false, as would the tuple $\langle Braves, 1992\rangle$. This assumes that the computer knows that only one team can win the Series a year. The point is that nothing is ever subtracted from what is already known by the system; "inconsistent" information (in the two-valued sense) is stored as part of the database itself and reasoning proceeds according to some alternative notion of entailment.

By similarly labeling inconsistent information with a truth value, De Amo et al. [19] use a paraconsistent formalism to maintain all the information that a database system has been told by external sources with respect to a set of possibly conflicting integrity constraints. A method called REPAIR takes a database instance and a set of integrity constraints to produce a *paraconsistent database*, essentially labeling the "controversial" information with a third truth value. Like Belnap's [11] approach, the conflicting information is stored as such and logical entailment and constraint satisfaction are defined using this third truth value (see Example 2.1.2 below). Yet paraconsistent formalisms can also be used to remove inconsistency once the inconsistent information is discovered. The work of Arieli and Avron [5] aims at *recovering* consistent information from inconsistent knowledge-bases using an extension of Belnap's [11] formalism.

Their use of a paraconsistent logic, based on the notion of a bilattice, functions to pinpoint inconsistent information. The logic's associated consequence relation $\models_{con}$ allows them to isolate the core of the inconsistency and define various *support sets* for *recoverable literals* in the language of the knowledge base [5]. A knowledge base is *recoverable* if there exists at lease one recoverable literal, a literal that is entailed by all the *most consistent models*. Contradictory information is considered either *spoiled* or *damaged* and therefore discarded with respect to $\models_{con}$; consistency is maintained at the expense of some (arguably) useless information.

Example 2.1.1 Let (where x is universally quantified):

$$\begin{aligned} KB \;=\; & \{heavy(A), heavy(B), \neg on\_table(A), \neg red(A), \\ & heavy(x) \rightarrow on\_table(x), heavy(x) \rightarrow red(x)\} \end{aligned}$$

of which there are four support sets:

$$
\begin{aligned}
KBa &= \{heavy(A), heavy(B), heavy(A) \rightarrow red(A), heavy(B) \rightarrow on_table(B)\},\\
KBb &= \{heavy(A), \neg red(B), heavy(A) \rightarrow on\_table(A)\},\\
KBc &= \{\neg on\_table(A), heavy(B), heavy(B) \rightarrow on\_table(B)\},\\
KBd &= \{\neg on\_table(A), \neg red(B)\}.
\end{aligned}
$$

According to Arieli and Avron [5], KBa is to be preferred since it is the largest set and supports more literals than any other support set. Observe that if $heavy(x) \rightarrow red(x)$ is an integrity constraint, it is not a member of KBd. If integrity constraints are supposed to be enforced, this is undesirable.

A similar approach can be found in the work of Baeza [10] which uses an annotated predicate calculus with a ten-valued lattice to specify the "degree" of truth certain facts have in relation to integrity constraints. For instance, the values t_d and f_d indicate truth values according to what is actually in a database, while t_a and f_a are "advisory" values that aid in the specification of *repairs* of a database instance. These repairs can then be generated with a logic program, and querying can thus proceed over these repairs. This is an extension of *consistent query answering* in the framework of annotated logic. We will have more to say about *consistent query answering*, below.

The work of Arieli et al. [8, 9] analogously uses a three-valued logic to provide a model-theoretic characterization of inconsistent facts and uses this information to specify repairs. They then use an abductive logic program to generate such repairs.[1] No querying system is provided for this latter work, but it is an arguably straightforward extension. In Chapter 5, we suggest our extension of Reiter's [45] query evaluator *demo* could be used as a front-end for this system, or any logic programming system that generates repairs.

Although these last two approaches have implementations in Prolog, the semantics of such implementations are non-classical in the sense that other truth values are used. The benefit of this is that we can retain all information by formally characterizing inconsistent data. The dilemma, however, is that we

[1] See the work of Gertz [24] for techniques from model-based diagnosis to generate repairs.

inherit the underlying semantic characterization of integrity constraints and of databases. The **LFI1** logic used by De Amo et al. [19] labels violated integrity constraints with the same truth value as inconsistent database facts. And Arieli and Avron [6] consider *knowledge bases* without integrity constraints. So if constraint is a sentence in the knowledge base, it will contribute to the inconsistency and be labeled "spoiled" or "damaged" (see Example 2.1.1). In addition, De Amo et al.'s [19] *paraconsistent databases* are non-standard in that the third truth value is part of the object language and sentences labeled with this value are part of the database itself.

Example 2.1.2 From De Amo et al. [19], we have:

$$\begin{aligned} DB &= \{R(a), R(b), R(c), Q(a), Q(b)\}, \\ IC &= \{\forall x(\neg R(x) \vee Q(x)), \forall x(\neg R(x) \vee \neg Q(x)), \forall x(\neg Q(x) \vee R(x))\} \end{aligned}$$

where IC is a set of integrity constraints. Their method REPAIR produces the following paraconsistent database:

$$DB' = \{R(a), R(b), \bullet R(c), \bullet Q(a), \bullet Q(b)\}$$

where $\bullet$ indicates the associated fact contributes to the inconsistency. With DB' we can then ascertain entailments with respect to the logic **LFI1**. So a set of sentences Σ entails sentence α whenever α is true in all the models of Σ. α is true in a structure if and only if the model assigns α the truth value 1 or $\frac{1}{2}$. Therefore, the sentences in IC follow from DB' but their truth value is $\frac{1}{2}$. Information loss is minimized but the constraints have the same truth value as database facts. Since constraints are not part of DB, then this fact can be effectively ignored because they label inconsistent data with $\bullet$ in DB', indicating a truth value of $\frac{1}{2}$ in the models of DB'.

These properties make the above formalisms unsuitable to a relational database setting where integrity constraints enforce legal states and where we model the database in first-order logic.

2.2 Querying Possibly Inconsistent Databases

To isolate inconsistent information, Bry [16] uses classical logic with restrictions on the proof theory, forbidding deductions based on *reductio ad absurdum.* The restriction on *reductio*-based deductions ensures that $\not\vdash \bot \rightarrow A$ for any A; any inconsistencies are localized and effectively quarantined from the rest of the database. We can then focus on querying a database that violates its integrity constraints. A *consistent answer* F is a formula that can be deduced without information involved in the derivation of $\bot$. It is important to note that $\bot$ can still be derived, but from this neither can any other formula. An *inconsistent answer* is any data from the *inconsistency kernel*, which is a minimal set of facts that derive $\bot$ together with the integrity constraints. Like paraconsistent-based approaches, Bry does not attempt to remove any inconsistency in the original database but instead restricts deductions so that querying only returns consistent answers.

The work of Arenas et al. [4] extends the work of Bry [16] by revisiting the notion of a *consistent query answer*, which they define as an answer that holds in all *repairs* of a possibly inconsistent database.

Example 2.2.1 Let:

$$\begin{aligned} DB &= \{WorldSeries(Braves, 0304), WorldSeries(Jays, 0304), \\ &\quad HomeStadium(Jays, Skydome)\}, \\ IC &= \{\forall x, y, z[WorldSeries(x, y) \wedge WorldSeries(z, y) \rightarrow x = z]\} \end{aligned}$$

where $WorldSeries(x, y)$ asserts that x won the World Series in season y and $HomeStadium(a, b)$ indicates that b is the home stadium of team a. IC expresses the uniqueness of World Series winners. There are two repairs of this inconsistent database:

$$\begin{aligned} r_1 &= \{WorldSeries(Braves, 0304), HomeStadium(Jays, Skydome)\}, \\ r_2 &= \{WorldSeries(Braves, 0304), HomeStadium(Jays, Skydome)\}. \end{aligned}$$

The constant $Jays$ is a consistent query answer to the query $HomeStadium(x, Skydome)$. We denote this state of affairs with the relation

$\models_c$:

$$DB \models_c HomeStadium(x, Skydome)^x_{Jays}$$

where α^x_a is the textual substitution of free variable x with a. This is to be distinguished with the approach of Arieli and Avron's [5] since, in this latter work, "support sets" are required to be subsets of the original database. As we see in the next example, Arenas et al. [4] consider insertions as possible repairs of a database.

Arenas et al. [4] also define an operator T_ω on queries that rewrites the query such that an answer is entailed by all repairs of the possibly inconsistent database. Particularly, T_ω attaches an associated *residue* to the query, which is borrowed from work in semantic query optimization [17]. This rewritten query is then posed to the database *as if* it is querying the set of all repairs. This method is sound with respect to the notion of a repaired database and complete with respect to a certain class of database dependencies and queries.

This approach is attractive because the database is not cleaned and we can query the system to debug faulty or conflicting information. The idea is that to establish which data is in fact inconsistent, some querying will have to occur [4, 16]. In the context of multidatabases, this approach respects the local autonomy of distributed sources by not enforcing local consistency on global constraints [14]. Consistent query answering therefore tolerates the inconsistencies until further information allows us to isolate and remove inconsistent data; we sacrifice consistency for some measure of completeness of information since only consistent answers are returned. Yet the following example shows that solely querying DB with respect to $\models_c$ results in information loss.

Example 2.2.2 Let DB, IC be the following:

$$\begin{aligned} DB &= \{P(a), Q(b), Q(c)\}, \\ IC &= \{\forall x(KP(x) \supset KQ(x))\}. \end{aligned}$$

We have:

$$\begin{aligned} r_1 &= \{P(a), Q(a), Q(b), Q(c)\}, \\ r_2 &= \{Q(b), Q(c)\} \end{aligned}$$

as the two repairs.

For $P(a) \in DB$, $DB \not\models_c P(a)$ and $DB \not\models_c \neg P(a)$. Notice, however, that for atomic $\alpha \notin DB$ we have $DB \not\models_c \alpha$ and $DB \not\models_c \neg\alpha$; completely absent information is indistinguishable from $P(a)$, which is in the database. The knowledge-level view of consistent query answers proposed in this thesis attempts to circumvent this shortcoming. See Chapter 5 for a formal discussion on Arenas et al. [4] and consistent query answering.

Likewise, Dung [22] focusses on querying an *integrated relational database*, allowing an epistemic query to quantify over the maximally consistent subsets of an *informationally closed* database. Aside from a special null constant $\perp$, representing unknown information, the semantic development is the same as the propositional fragment of modal logic $KD45$. "Worlds" in this work are single relations—such as $WorldSeries$ or $HomeStadium$—and the elements of these worlds are the tuples in the relation. Because of the special semantics for $\perp$, Dung uses two consequence relations $\models_t, \models_f$ to indicate when a formula F follows from a model, which is a set of worlds $\mathcal{W}$ and the real world W, denoted $(\mathcal{W}, W)$. The closed world assumption is built into this language called the *integrated relational calculus.* It is used to deal with integration on databases with respect to functional dependencies. A formula F is thus *known* if it holds in all the maximal consistent subsets of an integrated database.

While this approach has the benefit of an expressive query language, the underlying data model only considers maximally consistent *subsets*, appropriate for functional dependency violations but inapplicable to tuple-generating dependencies, in contrast to the work of Arenas et al. [4]. Moreover, the notion of a *query answer* considers the *union* of the answers for each of the worlds in $\mathcal{W}$ to obtain the *possible* instances to a *first-order* query.

Dung defines answers to a query with respect to $(\mathcal{W}, W)$, $ANS_Q(\mathcal{W}, W)$, to be the set of tuples $\vec{c}$ such that $(\mathcal{W}, W) \models_t F(\vec{c})$ for query $F(\vec{x})$. Furthermore, the set $ANS_Q(\mathcal{W})$ is defined as:

$$ANS_Q(\mathcal{W}) = \bigcup\{ANS_Q(\mathcal{W}, W) | W \in \mathcal{W}\}$$

Thus, queries of the form $K\alpha$ are evaluated $|\mathcal{W}|$ times because of the union. However, subjective formulas only depend on the epistemic state; it is enough

to call $ANS_Q(\mathcal{W}, W)$ *once* in the definition of $ANS_Q(\mathcal{W})$. Why Dung chose this definition for query answer seems to stem from the idea that the *possible* instances to a *first-order* query can only be obtained by taking the union of the answers to the individual worlds $W \in \mathcal{W}$. This is not the case, however. Consider the worlds w_1, w_2 defined as:

$$\begin{aligned} w_1 &= \{P(terry, 35), P(peter, 25)\} \\ w_2 &= \{P(terry, 20), P(peter, 28)\} \end{aligned}$$

where $P(a, b)$ is read "a has a salary of b thousand dollars".[2] Let $\mathcal{W} = \{w_1, w_2\}$. The answers to the query $Q_1 := K(\exists z P(x, z) \wedge z < 30)$ with respect to $\mathcal{W}$ are:

$$ANS_{Q_1}(\mathcal{W}) = \{peter\}$$

Now let $Q_2 := \exists z P(x, z) \wedge z < 30$, the answers to this query are as follows:

$$ANS_{Q_2}(\mathcal{W}) = \{peter, terry\}$$

In the latter case, we are asking the system to give the names of all employees whose salary is *possibly* less than 30. Since we take the union of the answers with respect to $\mathcal{W}$ for this first-order query, we get the answers that hold in *at least one of the worlds* $W \in \mathcal{W}$. That is, both Peter and Terry have salaries less than 30 thousand dollars in at least one world in $\mathcal{W}$. This seems to give Dung's system the power it needs to list *possible* instances of queries. However, the equivalent answers can be returned using the query $\neg K \neg \exists z P(x, z) \wedge z < 30$, which looks for an x such that it is not known that x does *not* have a salary less than 30 thousand dollars. Is there some world such that x has a salary less than 30 thousand dollars? If so, then any constant bound to x will be a query answer in Dung's language. Yet, we have simply used the dual of K to get possible instances to queries. Thus, that we need to take the union is redundant.

But this design decision stems from forbidding negation in the integrated relational calculus, into which Dung [22] transforms sentences with K operators, where the dual of K has no straightforward translation. Because we focus on the

[2]This is not exactly the way Dung [22] phrases worlds. Worlds are sets of tuples on a single relation; multiple relation worlds have not been defined. We have used sets of ground atomic formulas of one relation as the content of a world instead.

semantics of consistent query answering at the outset, we employ the expressive power to query possible instances using $\mathcal{KL}$. The approach presented in this thesis therefore uses a general knowledge representation query language [29, 30] that we refine with considerations stemming from database theory, deductive databases and logic programming. This refinement culminates in its application to the problem of database inconsistencies due to integrity constraint violations and a formalization of consistent query answers.

As a final consideration, Benferhat et al. [13] investigate inference relations on inconsistent knowledge bases. They propose a *safely supported consequence relation* defined over *consistent subbases* of the original knowledge base. The intuition is that if there is no *argument* in favour of the formula's negation, the formula is a *safely supported consequence.* So the entailment relation does not pick out inconsistences but uses the amount of support for a formula's negation. A notion of *level of paraconsistency* is associated with formulas in the knowledge base that aids in determining arguments in favour of the formula and its negation. Since inconsistent knowledge bases are semantically equivalent in a trivial way, this approach is syntactic in nature and so considers only formulas that explicitly appear in the stratified knowledge base, which is modeled in possibilistic logic; no repairing is attempted to regain consistency.

Nonetheless, what distinguishes these methods from others is that they focus on *query answering*: the rewritten query T_ω does not aim to produce a consistent instance, but is concerned only with those answers that are consistent with the integrity constraints; the query $K\alpha$ only asks whether something holds in all consistent subsets of an inconsistent database; the safely supported consequence considers subbases as arguments for the consequence but does not alter the original stratified knowledge base. In other words, the database instance or knowledge base need not be in any consistent state at all. Information is not removed. This understanding of querying over some set of information resembles work on handling *incomplete information.*

Querying Incomplete Databases

While we can model relational databases as sets of atomic sentences, the information of the world is nonetheless limited to *definite* information. The problem

is that if a database knows $\exists x P(x)$, then asking whether $P(a)$ holds for *any* constant a is only a possible answer. Moreover, if we use first-order entailment for answering queries, $\exists x P(x) \not\models P(a)$ for any constant a. Database systems on the other hand have a *definite answer property* where if a database $DB \models \exists x P(x)$ there *is* a witness such that $DB \models P(a)$ [35]. What if we have incomplete information of the world? How do we model this? The standard approach is to allow disjunction, and even existential quantification in the database, what Reiter [45] calls *elementary knowledge bases.* Querying thus proceeds on the models of this knowledge base, denoted $Mod(D)$. A *certain answer* is one that holds in all $M \in Mod(D)$. A *possible answer* is one that holds in at least one $M \in Mod(D)$ [1, 33, 35].

With disjunctive information, however, the notion of possible answer is obscured. If $Mod(D)$ is the set of structures in which $P(a) \vee P(b)$ is true, then neither $Mod(D) \models P(a)$ nor $Mod(D) \models P(b)$ hold. There are no certain answers to the query $P(x)$, which is to be expected. However, the set of possible answers is the set of all constants. This ignores the singular role of a and b with respect to $Mod(D)$. Various solutions to this problem have been suggested [35], but the system we propose does not suffer from this difficulty because it treats incomplete information as incomplete *knowledge.* We can ensure that if $KP(a) \vee KP(b)$ is known by our system, then it will know either $P(a)$ or $P(b)$; the set of possible answers to the query $P(x)$ consists only of a and b. This is called the *determinacy of knowledge* by Levesque and Lakemeyer [31] and Reiter [47], which we investigate in Chapter 3.

How do we find out whether a constant is a possible answer? We can ask whether $\exists x P(x) \models P(a)$ and $\exists x P(x) \not\models \neg P(x)$. This shows that $P(a)$ and $\neg P(a)$ are possibly known since $\exists x P(x)$ does not specify which x is a P; it could be a or not. In this thesis, we formalize this behaviour in $\mathcal{KL}$ and in what we call an *epistemic state* e_C. We can then query e_C for known ($K\alpha$), possibly known ($\neg K \neg \alpha$) or unknown ($\neg K \alpha$) facts using formulas in the language. Neither Arenas et al. [4] nor Dung [22] formally investigate whether possible answers are distinguishable from absent information—see Example 2.2.2 regarding [4]—or whether the query language is strong enough to ask about possible answers—see the discussion in the previous section regarding [22]. We envision the system we

develop to be a step in this direction, allowing us to retrieve answers that are possible using $\mathcal{KL}$ and to minimize information loss while maintaining consistency.

2.3 Coherence-Based Methods

Notice that the above querying techniques require the computation of *repairs* or *maximally consistent subsets* of the original database. In this sense, they differ from paraconsistent-based approaches that mark inconsistent data because an attempt is made to fix the inconsistency. These methods of revising an inconsistent knowledge base to restore consistency are what Arieli et al. [8, 9] call *coherence-based.*

Mendelzon and Lin [32] define a merging operator on finitely satisfiable knowledge bases with constraints that takes the *majority rule* of the knowledge bases that support a consequence. When there is no majority, we take disjunction of all the theories that need to be merged. This is a syntactic characterization of the operator. The semantics is based on the *minimum distance* between the models of the integrity constraints and the models of the first-order theories that need to be merged. Because of the majority rule property of this operator, some information may be lost. This work is also applied to *schematic heterogenity* where different vocabularies of different theories can also be merged.

The work of Motro [38] identifies consistent entailments of a database with respect to the *information goodness* of the information sources, and Qi et al.'s [41] work considers classical knowledge base merging as a specialization of possibilistic knowledge base merging; they sufficiently "weaken" inconsistent information so that not all information is lost. Since this latter work deals with inconsistent knowledge bases, integrity constraints are disjoined with any counterexamples, thus weakening the constraints. Finally, Delgrande and Schaub [20] extend their work in belief revision to support different consistency-based methods of knowledge base merging with entailment and consistency-based integrity constraints.

While the above approaches deal with a set of first-order formulas, Meyer [36] suggests merging *epistemic states.* He defines basic properties that a merging operation on *knowledge bases* should have, and distinguishes two extensions to a

merging operator on epistemic states: *arbitration*, which takes as many differing opinions as possible into account, and *consensus*, which is concerned with the opinion of the majority. This approach is relevant for the work presented in this thesis, but we consider this a direction for future study.

Suffice it to say, these approaches attempt to restore consistency at the expense of arguably useless information.

2.4 Conclusion

The purpose of exploring these approaches and their motivations is to evaluate whether we can employ their insights and methods to the special circumstances of *querying* possibly inconsistent databases. We adopt the perspective suggested in the work of Arenas et al. [4] and Dung [22] by considering *query answering* as a method that harmonizes the goals of pinpointing relevant, although possibly inconsistent information, and that of maintaining consistency. We accomplish this with an epistemic logic $\mathcal{KL}$ querying an *epistemic state*, which represents all that is known about the domain. This epistemic state is constructed using techniques from coherence-based methods of considering the *repairs* of a database instance. Information is not necessarily lost because we can quantify over these repairs, as suggested by techniques on handling incomplete information [33, 35]; consistent information are the only known facts, while those that are inconsistent are possibly but hitherto unknown. The hope is that while balancing these goals, we have a glimpse of Belnap's computer doing, and believing, exactly what it is told.

Chapter 3

The Logic $\mathcal{KL}$

To sing in truth is quite a different breath.

- Rainer Rilke, *Sonnets to Orpheus*

All our knowledge is, ourselves to know.

- Alexander Pope, *An Essay on Man*

If we adopt the standard view that a set of first-order sentences represents a knowledge base, we can understand query evaluation as a form of logical entailment where the instances of a query's free variables that make it true with respect to the knowledge base determines its value [43]. However, there is no reason to suppose that the query can only be phrased in the idiom of the relational calculus; the relational algebra and tableau's are equivalent in expressive power, and in some cases, better suited to certain tasks than the relational calculus for querying databases [1]. We can thus distinguish between the *query language* or *interaction language*, which is used to ask questions or relate facts about the world the knowledge base models, and the *representation language*, which consists of the symbolic structures representing that world [30].

By separating these concerns we can use an epistemic query language to ask what a knowledge base *knows*, in addition to asking about aspects of the external world. For instance, if the system knows that Jane *has* a social security number, but cannot identify it, we can distinguish between asking "Do you know that Jane has a social security number?" and "Do you know Jane's social

security number?". As we see in the next chapter, $\mathcal{KL}$ also allows us to view integrity constraints as statements about the knowledge the knowledge base has of the world, as opposed to statements about the world itself [45]. This perspective about integrity constraints and $\mathcal{KL}$ as query language allows us to treat database inconsistencies with respect to integrity constraints as properties of the knowledge the system has of its domain; consistent answers are those that are *known*. We can then formalize Arenas et al.'s [4] notion of a consistent query answer in $\mathcal{KL}$.

In this chapter we outline the basic syntax and semantics of a subset of $\mathcal{KL}$ that we call $KFOPCE$ and its first-order counterpart (without the modal operator K) $FOPCE$, which stands for "First-Order Predicate Calculus with Equality". We also consider some of the logic's properties as well as a basic operator $\models\!\!\!\approx$ that allows us to pose queries. While the motivation for using $\mathcal{KL}$ is to formalize consistent query answers, that we can use it as a query langauge for databases enhances its applicability. This means that we need to restrict the language to ensure that queries are *safe* and *domain independent*, both with and without closure constraints; this is the topic of the last section.

The majority of definitions and results presented in this chapter are from Levesque and Lakemeyer [30] and/or Reiter [45] unless there is no reference to these works in the definitions, lemmas or theorems. Section 3.4.2 on domain independent $KFOPCE$ formulas and Section 3.4.3 on AC formulas, however, can be seen as straightforward extensions of these works.

3.1 Syntax

$KFOPCE$ is a first-order modal language with equality augmented with the modal operator K, which stands for "know". Expressions in $KFOPCE$ are constructed from the following sets [30]:

Logical symbols consist of the distinct sets:

- a countably infinite set of *variables*, which we write as x, y or z, possibly with subscripts.
- a countably infinite set of *parameters* $\mathbb{P}$, written schematically as $p_1, p_2, \ldots$. We will also write parameters either as lower-case letters $a, b, c, \ldots$ or

as proper names (e.g. mary, car, coffee, etc.) Intuitively, parameters are the unique names of domain elements.

- the *equality* symbol, $=$.
- the *logical connectives* $\forall, \exists, \wedge, \vee, \neg, \supset$.
- the usual *punctuation symbols* (,),,.
- the single *modal operator*, K for "know".

Non-logical symbols are domain specific elements of the vocabulary distinct from the logical symbols. They consist of only:

- a set of *predicate symbols*, which we usually write as P, Q or R. These are understood to be domain specific relations or properties. Predicate symbols have an *arity*, which is a number indicating how many arguments it takes.

From these symbols, the *expressions* of $KFOPCE$ are built. There are two types of expressions: *terms*, which are used to identify individuals in the domain and which we schematically name as $t_1, t_2, \ldots$, and (well-formed) *formulas*, which are statements about the domain and which we schematically identify with lowercase Greek letters $\alpha, \beta, etc.$ Sets of wffs are usually denoted with Σ, possibly with subscripts.

Definition 3.1.1 (Term) *A* term *is either a* parameter *or a* variable.

Definition 3.1.2 (Atomic Formula) *An* atomic formula *or* atom *is of the form* $P(t_1, \ldots, t_k)$ *where* P *is a predicate symbol of arity* k *and* t_i *are terms. We write equality atoms as* $t_1 = t_2$.

Definition 3.1.3 (Well-Formed Formula) *A* well-formed formula *in* $KFOPCE$, *or* wff, *is one of the following where* α *and* β *are wffs: an atom,* $\neg\alpha$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \supset \beta)$, $\exists x\alpha$, $\forall x\alpha$, $K\alpha$.

Notice that *wffs* in $FOPCE$ are $KFOPCE$ formulas without the modal operator K; they will henceforth be called *objective* formulas. A variable x is *bound* if it appears in a subwff of the form $\exists x\alpha$ or $\forall x\alpha$, in which case we say that x is within the *scope* of the quantifier $\exists$ or $\forall$, respectively. A variable x is *free* otherwise.

We define the set of *sentences*, which are the only wffs that can receive a truth value.

Definition 3.1.4 (Sentence) *A* sentence *is a KFOPCE wff with no free variables. An* atomic sentence *is an atom $P(t_1, \ldots, t_k)$ where each t_i is a parameter.*

Finally, by α_t^x we mean the textual substitution of the variable x for the term t. $\alpha_{\vec{t}}^{\vec{x}}$ means that we simultaneously substitute each free x_i with the corresponding t_i, assuming that the sequences $\vec{x}$ and $\vec{t}$ are the same length. We will sometimes write textual substitution as $\alpha_1|_{\vec{t}}^{\vec{x}}$ to make it easier to read with the subscripted α.

Databases and Knowledge Bases

Our language for representing information about the world is $FOPCE$, since there is no reference to "knowledge" in such formulas.

Definition 3.1.5 (Database [45]) *A* database instance *or* database *is a finite set of non-equality atomic sentences, i.e. of wffs $P(p_1, \ldots, p_k)$ where P is a predicate symbol of arity k and p_i are parameters for $1 \leq i \leq k$.*

Where required, we will need to define supersets of databases, which are nonetheless proper subsets of an unrestricted set of $FOPCE$ sentences. We therefore define the widest class of $FOPCE$ sentences that we discuss.

Definition 3.1.6 (Knowledge Base [30]) *A* knowledge base *is a set of $FOPCE$ sentences.*

3.2 Semantics

We have seen how to build sentences and formulas in $KFOPCE$. In this section, we look at how the sentences we build can be *true* or *false*.

Definition 3.2.1 (World [45]) *A* world *is any set of atomic sentences that includes $p = p$ for each parameter p and does not include $p_1 = p_2$ for different parameters p_1, p_2.*

Parameters are therefore pairwise distinct at a world and are isomorphic to the application domain; every element has a name in the form of a parameter. The *unique names assumption* is therefore part of the logic [43].[1] This also means that our domain of quantification is countably infinite, but we shall see in the coming sections how to prove results for $KFOPCE$ analogous to the issue of *safety* in database theory which guarantee finite answers to queries [45].

Definition 3.2.2 (Epistemic State [30]) *An* epistemic state *is a (possibly empty) set of worlds.*

Definition 3.2.3 (Satisfaction [30, 45]) *We define how an epistemic state e and world w* satisfy *a $KFOPCE$ sentence ϕ (denoted $e, w \models \phi$) as follows:*

- *If ϕ is an atomic sentence, then $e, w \models \phi$ iff $\phi \in w$.*
- *If ϕ is of the form $\neg\pi$, then $e, w \models \phi$ iff $e, w \not\models \pi$.*
- *If ϕ is of the form $\pi_1 \wedge \pi_2$, then $e, w \models \phi$ iff $e, w \models \pi_1$ and $e, w \models \pi_2$.*
- *If ϕ is of the form $\forall x\pi$, then $e, w \models \phi$ iff for every parameter p we have $e, w \models \pi_p^x$.*
- *If ϕ is of the form $K\pi$, then $e, w \models \phi$ iff for every $s \in e$, we have that $e, s \models \pi$. Since ϕ does not depend on the world w, we sometimes write $e \models K\pi$.*

The cases for $\exists x\pi$, $\pi_1 \vee \pi_2$ and $\pi_1 \supset \pi_2$ are defined in the usual way in terms of $\neg, \wedge$ and $\forall$. When ϕ is objective, we sometimes write $w \models \phi$ to emphasize its independence from the epistemic state. We say that ϕ is true *in (e, w) whenever $e, w \models \phi$.*

Definition 3.2.4 (Model [45]) *For Σ a set of $KFOPCE$ sentences, (e, w) is a* model *of Σ iff $e, w \models \phi$ for all $\phi \in \Sigma$. If Σ is a set of $FOPCE$ sentences, the set of* first-order models *for Σ is denoted $\mathcal{M}(\Sigma) = \{w \mid w \models \phi, for\ all\ \phi \in \Sigma\}$.*

If there is at least one model for Σ, we say that Σ is *satisfiable.* If a $KFOPCE$ sentence α is true in all models of $\emptyset$, we say that α is *valid.*

[1] In $\mathcal{KL}$, parameters serve as names of equivalence classes of *co-referring* terms. Function symbols f are members of a particular equivalence class only when it is known that $f(t_1, \ldots, t_k) = p$ for any parameter p and $k \geq 0$.

We write $\Sigma \models_{KFOPCE} \phi$ whenever the $KFOPCE$ sentence ϕ is true in all models of Σ. Similarly, we write $\Sigma \models_{FOPCE} \phi$ just in case for all $w \in \mathcal{M}(\Sigma)$ we have $e, w \models \phi$ for any epistemic state e, a set of $FOPCE$ sentences Σ and $FOPCE$ sentence ϕ. Whenever Σ is strictly $FOPCE$ and α is $KFOPCE$, we use $\models\!\!\!\approx$, which we define below.

A sentence is *objective* if it is a $FOPCE$ sentence, since deciding its truth or falsity does not depend on the epistemic state e (i.e. on what is known). A *subjective* sentence is one where all predicate symbols appear in the scope of a K operator, thus whose truth or falsity solely depends on the epistemic state e. For instance, $K(P(p_1) \vee Q(p_2))$ is subjective, $\forall x(P(x) \supset Q(x))$ is objective and $(P(p_1) \wedge \neg KP(p_1))$ is neither. Intuitively, objective formulas say something about the *world* whereas subjective formulas refer to the knowledge a system has about the world, and hence only depend on the epistemic state. Mixed formulas such as $(P(p_1) \wedge \neg KP(p_1))$ allow us to address both aspects: the world and the epistemic state. Without knowing the sentences that are actually in a knowledge base, these sentences allow us to express facts about the world that would not have been possible otherwise [30]. They hence enhance the expressivity of the query language.

3.2.1 Some Properties of $KFOPCE$

For the propositional fragment of $KFOPCE$, the semantic specification given above is equivalent to the modal logic $K45$; in addition to the axiom K, the positive and negative introspection axioms (below) are valid in $KFOPCE$ [26, 30]. That is, for $KFOPCE$ sentence α, β:

Theorem 3.2.5 (Levesque and Lakemeyer [30]) $\models K(\alpha \supset \beta) \supset (K\alpha \supset K\beta)$

Lemma 3.2.6 ([30]) $\models K\alpha \supset KK\alpha$ *and* $\models \neg K\alpha \supset K\neg K\alpha$.

Theorem 3.2.7 ([30]) *For any subjective sentence* σ *we have that* $\models \sigma \supset K\sigma$.

This important property tells us that any knowledge base system has *complete* knowledge of its internal state; we cannot tell it a fact about itself that it does not already know. For our purposes, since we are only interested in querying a possibly inconsistent database, we can be sure that there are no questions

about its knowledge to which it does not have a "yes" or "no" answer (See Corollary 3.3.3 below).

Does the converse of the theorem hold? That is, are known subjective sentences true? Consider the *empty* epistemic state e, which represents inconsistent knowledge; all possible worlds have been ruled out as the actual one. If ϕ is any atomic sentence, then $\neg K\phi$ is true iff there exists some world w in e such that $w \not\models \phi$. Since e is empty, this cannot be the case and hence $\neg K\phi$ is false. However, $K\neg K\phi$ will be true trivially since e is empty. Hence, the known subjective sentence $\neg K\phi$ is not true. We can ensure consistent epistemic states, and hence accuracy of subjective knowledge, with the following theorem.

Theorem 3.2.8 (Levesque and Lakemeyer [30]) *For any α and any subjective σ, $\models (\neg K\alpha \supset (K\sigma \supset \sigma))$.*

The following is a useful corollary that will be employed in Chapter 6.

Corollary 3.2.9 *Whenever e is non-empty, then $e \models K\sigma \supset \sigma$ for subjective σ.*

If we assume that e is non-empty, the resulting logic, or propositional fragment thereof, is therefore equivalent to modal logic $KD45$ [45].

Properties of $KFOPCE$ With Respect To Parameters

$KFOPCE$'s treatment of parameters also gives us a uniform domain across all the world states. This means that the *Barcan formula* and its converse hold in $KFOPCE$.

Theorem 3.2.10 (Levesque and Lakemeyer [31]) $\models \forall x_1 \ldots x_k K\alpha \equiv K\forall x_1 \ldots x_k \alpha$

A similar result does not work for existential quantifiers, however.

Theorem 3.2.11 ([30]) $\models \exists x KP(x) \supset K\exists x P(x)$ *but not conversely.*

Theorem 3.2.12 (Determinacy of Knowledge [31]) *Suppose α is objective and β an objective formula with free variable x such that $\models K\alpha \supset \exists x K\beta$. Then for some parameter we have $\models K\alpha \supset K\beta_p^x$.*

So not only does our treatment of parameters let us move K operators around universal quantification, it allows us to distinguish when an individual is known to have a property from knowing *that* an individual has that property. The determinacy of knowledge theorem furthermore gives us what has been called the *definite answer property* in the context of databases [35]; in standard first-order logic $\exists x P(x) \not\models_{FOPCE} P(p)$ for any parameter p, but in $KFOPCE$ we can be sure that there will always be such a p whenever $\exists x KP(x)$ holds.

3.3 Querying a Knowledge Base

The definition of an *answer to a query* is given as follows:

Definition 3.3.1 (Answer to a Query [45]) *Let Σ be a knowledge base, and q a $KFOPCE$ formula with free variables $\overrightarrow{x}$. Let $e_\Sigma = \mathcal{M}(\Sigma)$. A tuple of parameters $\overrightarrow{p}$ is an* answer *to q (wrt Σ) iff $e_\Sigma, w \models q\frac{\overrightarrow{x}}{\overrightarrow{p}}$, for each $w \in e_\Sigma$ (i.e. $e_\Sigma \models Kq\frac{\overrightarrow{x}}{\overrightarrow{p}}$). We will sometimes write it as $\Sigma \models\!\!\approx q\frac{\overrightarrow{x}}{\overrightarrow{p}}$.*

Definition 3.3.2 (Instances of a Query [45]) *Suppose Σ is a set of $FOPCE$ sentences, and α a $KFOPCE$ formula with free variables $\overrightarrow{x}$. Define:*

$$Instances(\alpha, \Sigma) = \{\overrightarrow{p} \mid \overrightarrow{p} \text{ are parameters and } \Sigma \models\!\!\approx \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}\}$$

We overload this definition when the second argument is an epistemic state e:

$$Instances(\alpha, e) = \{\overrightarrow{p} \mid \overrightarrow{p} \text{ are parameters and } e \models K\alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}\}$$

When α is a $KFOPCE$ sentence, then $\Sigma \models\!\!\approx \alpha$ means "yes", $\Sigma \models\!\!\approx \neg\alpha$ means "no" and neither means "unknown". This latter case occurs when $e_\Sigma \models \neg K\alpha$ and $e_\Sigma \models \neg K\neg\alpha$; neither α nor $\neg\alpha$ are known. However, whenever σ is subjective, we have the following consequence of Theorem 3.2.7.

Corollary 3.3.3 (Levesque and Lakemeyer [30]) *If Σ is a set of $FOPCE$ sentences and σ is subjective, then either $\Sigma \models\!\!\approx \sigma$ or $\Sigma \models\!\!\approx \neg\sigma$.*

This reflects the intuition that we can ask a database something about its knowledge and be sure that there are no facts about itself to which it does not have *complete* knowledge.

What happens when $\Sigma = \{\}$? Based on the above properties and theorems we can show that, with the exception of tautologies, not only is *nothing* known in the corresponding epistemic state $e_0 = \mathcal{M}(\{\})$ but that the knowledge base *knows* this; it knows that it has incomplete knowledge of every aspect world.

Theorem 3.3.4 (Levesque and Lakemeyer [30]) *If* α *is* $[\exists x P(x) \supset \exists x(P(x) \wedge \neg KP(x))]$ *for any unary predicate symbol* P, *then* $\{\} \models\!\!\approx \alpha$.

What α says is that e_0 knows that it has incomplete knowledge of P: if one instance of P exists, it must be unknown.[2] Of course, the objective validities will also be known, and so will $\neg K\alpha$ when $\neg\alpha$ is satisfiable. That is, $\Sigma \models\!\!\approx \neg K\alpha$. Since we have assumed that our system has complete knowledge of its own epistemic state, like all other epistemic states, e_0 knows the valid sentences as well as the true subjective ones.

This epistemic state is important for integrity constraint satisfaction: since we view integrity constraints as statements about the knowledge a system has of its domain, they must be true when *nothing* is known that violates them. That an integrity constraint IC is known in e_0 will distinguish this approach from the standard definitions of constraint satisfaction. We investigate these remarks in the next chapter.

What if our query q is a mixed formula (i.e. neither objective nor subjective)? Consider the query $q := \exists x(P(x) \wedge \neg KP(x))$; q asks whether there are possible but hitherto unknown instances of P. Examine, however, the query $q' := P(x) \wedge \neg KP(x)$. Is q' asking the knowledge base to *list* the possible instances of P? Unfortunately, this is not the case. $Instances(q', \Sigma)$ is the set of parameters $\overrightarrow{p}$ such that for free variables $\overrightarrow{x}$ in q', $e_\Sigma, w \models q'|\frac{\overrightarrow{x}}{\overrightarrow{p}}$ for *every* $w \in e_\Sigma$. For the first conjunct in q' and parameter p_1, we are asking whether $P(p_1)$ holds in *all* worlds in e_Σ; we are asking $KP(p_1)$.

This "strengthening" of queries seems to prevent us from asking for a list of possible answers to a query as a naive use of q leads us to believe, but we can simply using the dual of K: $\neg K\neg$. The query $q'' := \neg K\neg P(x) \wedge \neg KP(x)$ lets us ask for a list of possible answers. Suppose we substitute p_1 for x in q''. Then we have $\neg K\neg P(p_1) \wedge \neg KP(p_1)$. We are asking whether there is some world in

[2]This theorem can easily be extended to the non-unary case.

which p_1 is a P and this p_1 is also *unknown*, or equivalently whether there is also some world in which p_1 is *not* a P. We henceforth leave it to the query writer to specify when he/she wants the possible instances of a query.

3.3.1 Other Properties of $\models\!\!\approx$

We have the following consequences of the above definitions from Reiter [45].

Proposition 3.3.5 (Reiter [45]) $\Sigma \models_{KFOPCE} \alpha$ *implies* $\Sigma \models\!\!\approx \alpha$ *but not conversely.*

Proposition 3.3.6 ([45]) *Whenever* α *is a* $FOPCE$ *sentence, then* $\Sigma \models\!\!\approx \alpha$ *iff* $\Sigma \models_{FOPCE} \alpha$.

Theorem 3.3.7 (Levesque and Lakemeyer [30]) *Suppose that* Σ *is a knowledge base and* α, β *are* $KFOPCE$ *sentences. Then* $\Sigma \models\!\!\approx \alpha$ *and* $\alpha \models_{KFOPCE} \beta$ *implies* $\Sigma \models\!\!\approx \beta$.

This means that if $\models \alpha \equiv \beta$, then we can use the query that is computationally more feasible and get the same results.

The following is a useful proposition on the equivalent formulation of $\models\!\!\approx$ in terms of an epistemic state e for $KFOPCE$ α.

Proposition 3.3.8 $e \models K\alpha$ *iff for all* $w \in e$ *we have* $\{w\} \models K\alpha$ *for* $KFOPCE$ *sentence* α.

Proof: The proof is by induction on the structure of α. When α is atomic, then $e \models K\alpha$ iff for all $w \in e$ we have $e, w \models \alpha$ iff $w \models \alpha$ iff $\{w\} \models K\alpha$.

The result follows trivially for α of the form $K\alpha_1$ and $\alpha_1 \wedge \alpha_2$. When α is of the form $\exists x \alpha_1$ then $e \models K\exists x \alpha_1$ iff $e, w \models \exists x \alpha_1$ for all $w \in e$. This holds iff $e, w \models \alpha_1|_p^x$ iff $e \models K\alpha_1|_p^x$ since $w \in e$. The result thus follows by induction. Finally, consider the case when α is of the form $\neg\alpha_1$. $e \models K\neg\alpha_1$ iff $e, w \models \neg\alpha_1$ iff $e, w \not\models \alpha_1$ for all $w \in e$. This is the same as saying $e \not\models K\alpha_1$ since there is at least one world where α_1 is false. But by the induction hypothesis, we have $\{w\} \not\models K\alpha_1$ for all $w \in e$. So, $\{w\}, w \not\models \alpha_1$ iff $\{w\}, w \models \neg\alpha_1$ since we are metalogically quantifying over *all* worlds in e. Thus, $\{w\} \models K\neg\alpha$. □

Thus, if we were to find a representative set of sentences for each world $w \in e$, then we can effectively use $\models\!\!\approx$ to get the same answers as if we were querying e.

In the next section we investigate what refinements are needed for $KFOPCE$ queries if we want to use $KFOPCE$ as a query language for relational databases, since certain restrictions need to be respected for this particular setting.

3.4 Specializations to Database Theory

This section presents a summary, specialization and extension of Reiter's [45] results on a query evaluator for $KFOPCE$. Up to this point, our assumptions about knowledge bases and $\models\!\!\approx$ supervene the requirements of relational database theory where the *relational theory* of Reiter [43] is assumed to hold. In particular, since the unique names assumption is built into the logic, our only concern are queries with possibly infinite answers and closed world databases. We see how Reiter [45] restricts $KFOPCE$ queries to guarantee finite answers, and show that the definition of admissible queries also ensures domain independence. We finally look at the closed-world assumption. The results of this section show that $KFOPCE$ can be used as a query language for databases.

3.4.1 On Finite Answers for $KFOPCE$ Queries

Since our domain of quantification is countably infinite, our definition of query answer does not guarantee finite answers. For instance, when our epistemic state is e_0, then the set $\{\overrightarrow{p} \mid e_0 \models \neg KP(\overrightarrow{p})\}$ is infinite. In this section we define a subset of $KFOPCE$ formulas that guarantee finite answers under $\models\!\!\approx$.

Definition 3.4.1 ([45]) *$\mathcal{F}_\Sigma$ is a set of FOPCE formulas with the property that whenever $f \in \mathcal{F}_\Sigma$, then $Instances(f, \Sigma)$ is finite.*

Henceforth, we call formulas $f \in \mathcal{F}_\Sigma$ *safe* [1, 51]. Reiter's [45] definition of safe queries is different from ours. *Safety* for Reiter means "safe for negation" in the context of a Prolog-like query evaluator where unbound variables on negated subgoals are discouraged. Aside from this difference in terminology, the definition above for $\mathcal{F}_\Sigma$ is Reiter's [45] original.

Remark 3.4.2 $\mathcal{F}_\Sigma$ need not be the entire set of formulas with the aforementioned property, but could be a proper subset. We define such a subset below once we establish that $Instances(\alpha, \Sigma)$ for $KFOPCE$ formula α built from $\mathcal{F}_\Sigma$ is finite as well.

Definition 3.4.3 (Almost Admissible Formulas wrt $\mathcal{F}_\Sigma$ [45]) *The $KFOPCE$ formulas which are* almost admissible with respect to (wrt) $\mathcal{F}_\Sigma$ *are the smallest set such that:*

1. *If $f \in \mathcal{F}_\Sigma$, then f is almost admissible wrt to $\mathcal{F}_\Sigma$.*
2. *If σ is an almost admissible formula wrt $\mathcal{F}_\Sigma$, so is $K\sigma$.*
3. *If σ is a subjective almost admissible sentence wrt $\mathcal{F}_\Sigma$, then $\neg\sigma$ is almost admissible wrt $\mathcal{F}_\Sigma$.*
4. *If σ is a subjective almost admissible formula wrt $\mathcal{F}_\Sigma$, then $(\exists x)\sigma$ is almost admissible wrt $\mathcal{F}_\Sigma$.*
5. *If σ_1 with free variables $\overrightarrow{x}$ is almost admissible wrt $\mathcal{F}_\Sigma$, and σ_2 is* any *$KFOPCE$ formula such that $\sigma_2|\frac{\overrightarrow{x}}{\overrightarrow{p}}$ is almost admissible wrt $\mathcal{F}_\Sigma$, then so is $\sigma_1 \wedge \sigma_2$.*

Definition 3.4.4 (Admissible Formulas wrt $\mathcal{F}_\Sigma$ [45]) *When all quantified variables of an* almost admissible formula wrt $\mathcal{F}_\Sigma$ *are distinct from one another and from the free variables of the query, then we call the formula* admissible wrt $\mathcal{F}_\Sigma$.

Example 3.4.5 Whenever $f, g \in \mathcal{F}_\Sigma$, the following formulas are admissible wrt $\mathcal{F}_\Sigma$: $\neg Kf, \exists x Kf, Kf$, and $f \wedge \neg Kg$. We will look at the form of f, g once we define the class of formulas $\mathcal{F}_\Sigma$. The following are not admissible: $\neg P(x), \exists x Q(x,y), \neg P(x) \wedge \neg KQ(x,y), K\exists x \neg Q(x,y)$.

Lemma 3.4.6 (Reiter [45]) *Whenever α is admissible wrt $\mathcal{F}_\Sigma$, $Instances(\alpha, \Sigma)$ is finite.*

Remark 3.4.7 Queries whose answers are finite are not necessarily admissible wrt $\mathcal{F}_\Sigma$, although we have proved that the converse holds for a particular $\mathcal{F}_\Sigma$. For example, $Instances(\forall y R(x,y), \Sigma)$ is necessarily finite for databases but is not admissible wrt $\mathcal{F}_\Sigma$. Since y ranges over the entire domain and since databases are finite, $Instances(q, \Sigma)$ is the empty set.

What remains is to find a set of $FOPCE$ formulas α such that $Instances(\alpha, \Sigma)$ is finite. The above lemma then guarantees that $KFOPCE$ formulas built from such a set has the desired property. Reiter [44] defines such a set, which he calls *positive existential* formulas.

Definition 3.4.8 (Positive Existential Formulas [45]) *The* positive existential *(p.e.) FOPCE formulas are the smallest set such that:*

1. *An atomic formula other than an equality atom is p.e.*
2. *If α is p.e., so is $(\exists x)\alpha$.*
3. *If α_1, α_2 are p.e., so are $\alpha_1 \wedge \alpha_2$ and $\alpha_1 \vee \alpha_2$.*

A *rule* is of the form $\forall \overrightarrow{x} A \supset B$ where A is a conjunction of atomic formulas other than equality and the variables of $\overrightarrow{x}$ occur free in A.

A knowledge base is *elementary* if it is a set of p.e. sentences and rules.

Remark 3.4.9 A database *is* therefore a finite elementary knowledge base, without disjunctions, conjunctions, existential quantifications and rules.

We need the following lemma to ensure that finite answers are returned for the class of queries we define below. So we need to move from a knowledge base Σ to one of its models that mentions only parameters occurring in Σ because querying is defined in terms of the models of Σ. From this point, if $\alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$ is true in that model, then the set of parameters $\overrightarrow{p}$ is be finite, which we prove later.

Lemma 3.4.10 (Reiter [44]) *Suppose Σ is an elementary knowledge base. Then Σ has a model with the property that each atomic non-equality sentence in that model mentions only parameters occurring in Σ.*

This is not enough, however. Consider the set of answers to the query $P(x, y) \vee Q(y, z)$. If either disjunct is true for parameters p_1, p_2 then the entire disjunction is true for these parameters as well as for *all* parameters. Why this is so depends largely on the fact that the free variables of the first disjunct are not the same as those of the second. If the free variables are the same, and if the query is true, then there must be some atomic sentence in a model of Σ that binds the variables to parameters mentioned only in this model. The variables in the other disjunct, whether true or false, would therefore be bound to the

same parameters *mentioned only in that model.* If we ensure that the knowledge base mentions only a finite number parameters, then the model will as well. We therefore need to restrict the free variables of queries, otherwise known as *range restriction* in the theory of query languages [1].

Definition 3.4.11 (Formulas with Disjunctively Linked Variables [45]) *Suppose α is an FOPCE formula with free variables $\vec{x}$. Then α has disjunctively linked variables iff for each of its subformulas of the form $\alpha_1 \vee \alpha_2$, those free variables of α_1 which are among $\vec{x}$ are precisely those of α_2 which are among $\vec{x}$.*

Example 3.4.12 The following do *not* have disjunctively linked variables: $\forall x P(x) \vee Q(p)$ for parameter p, $P(x,y) \vee Q(y,z)$ and $\forall x P(x) \vee Q(x,y)$.

With this restriction on disjunctively linked variables on p.e. formuals, we can now show that such formulas always have finite answers.

Lemma 3.4.13 ([45]) *If Σ is an elementary knowledge base mentioning only finitely many distinct parameters, and α is a p.e. formula with disjunctively linked variables, then $Instances(\alpha, \Sigma)$ is finite.*

Finally, we show that $KFOPCE$ formulas built from this set also guarantee finite answers.

Theorem 3.4.14 (Reiter [45]) *Suppose Σ is an elementary knowledge base mentioning only finitely many parameters. Let*

$$\begin{aligned}
\mathcal{F}_\Sigma \;=\; & \{\pi \mid \pi \text{ is a p.e. formula with disjunctively linked variables}\} \\
\cup\; & \{p = p' \mid p \text{ and } p' \text{ are parameters}\} \\
\cup\; & \{p \neq p' \mid p \text{ and } p' \text{ are parameters}\} \\
\cup\; & \{x = p, p = x \mid x \text{ is a variable and } p \text{ is a parameter}\}.
\end{aligned}$$

Then for all $KFOPCE$ formulas α that are admissible with respect to $\mathcal{F}_\Sigma$, $Instances(\alpha, \Sigma)$ is finite.

Example 3.4.15 The following formulas are admissible wrt $\mathcal{F}_\Sigma$:

$P(x,y) \wedge \neg KQ(x) \wedge \neg KR(y)$,
$KQ(x,y)$,
$\neg\exists x KP(x)$,
$\exists x KP(x)$,
$\neg(\exists x)K(male(x) \wedge female(x))$,
$\neg(\exists x,y,z)[KP(x,y) \wedge KP(x,z) \wedge \neg Ky = z]$,
$Q(x,y) \wedge \neg K(P(x) \vee \neg K(x,y))$.

The following are not admissible formulas wrt $\mathcal{F}_\Sigma$:

$\exists x K\neg P(x)$,
$\exists x[P(x) \wedge \neg KP(x)]$,
$\neg KQ(x,y) \wedge \neg KP(x)$,
$\neg K\neg P(x)$,
$K(P(x) \vee Q(x,y))$.

Unless specified otherwise, we assume that Σ is an elementary knowledge base and that $\mathcal{F}_\Sigma$ is as in Theorem 3.4.14, above (see Remark 3.4.2). With this assumption, we henceforth call *admissible* the admissible formulas wrt $\mathcal{F}_\Sigma$.

Remark 3.4.16 Since Reiter [45] defines *admissible* formulas as a superset of *admissible formulas wrt* $\mathcal{F}_\Sigma$ and since we will always consider *admissible* formulas wrt $\mathcal{F}_\Sigma$, we abuse the terminology here by describing these different classes with the same name.

A problem with these class of queries is that we lose the ability to ask the knowledge base about its possible but hitherto unknown knowledge: $\exists x(P(x) \wedge \neg KP(x))$ is not admissible. There does not seem to be a way of formulating an admissible query that addresses this aspect of the knowledge of our system. We can, however, ask whether p_1 is an unknown instance of P, $\neg KP(p_1)$, but this does not say that it is a possible instance. Similarly, we can ask if there is a possible instance with $K\exists x P(x)$, but this does not mean that such an x is an unknown instance. Intuitively, an unknown instance p_1 of P requires a world in which $\neg P(p_1)$ holds while a possible instance p_2 of P requires a world in which $P(p_2)$ holds. So it seems that we can only ask about unknown instances if we already have an individual in mind, and that we can ask about possible instances

only if we do not have such an individual in mind. Yet when a knowledge base is acquiring its knowledge incrementally, it is not unreasonable to prohibit questions asking about its incomplete knowledge requiring infinite answers.

Informally, these problems arise from the fact that our domain of quantification is countably infinite, and the set of unknown or possible instances for a query is potentially infinite because of this. Asking about an unknown or possible individual may require values from "outside" the values that actually appear in the knowledge base; such queries *depend* on the underlying domain. Alternatively, queries addressing this aspect of the knowledge base are not *domain independent.* We discuss domain independence and its importance in the next section. This does not mean that the power afforded by the K operator is lost, however; there is still an important distinction between $K\exists x P(x)$ and $\exists x K P(x)$, which are both admissible, and even being able to ask about unknown or possible instances at all distinguishes $KFOPCE$ as a useful query language for elementary knowledge bases. From an implementational point of view, moreover, the sizable class of queries we have defined above is suitable for Reiter's [45] $KFOPCE$ query evaluator *demo*, which is sound and complete with respect to these queries. We investigate an extension of *demo* for *consistent query answering* in Chapter 5.

3.4.2 Domain Independence for $KFOPCE$ Formulas

With the restriction on disjunctively linked variables, p.e. formulas as queries guarantee finite answers with respect to an elementary knowledge base. This restriction also entails that these formulas are *domain independent* [1, 50], as we prove below. This property is particularly important because the answers to any domain independent query are the same *regardless of the underlying domain*, up to isomorphism. An example of a domain *dependent* query is $\forall y R(x, y)$. As noted in the previous section, if y ranges over a countably infinite domain, $Instances(\forall y R(x, y), \Sigma)$ is the empty set. Otherwise, this set could be non-empty. This is undesirable because the information about the domain is not available to the user of the system when he/she phrases queries. It would be interesting to know whether $KFOPCE$ admissible formulas wrt $\mathcal{F}_\Sigma$ are also domain independent.

Definition 3.4.17 (Domain Independence [1, 50, 51]) *Let Σ be a knowledge base and α a KFOPCE formula. For any two non-empty subsets $d, d' \subseteq \mathbb{P}$, if $Instances_d(\alpha, \Sigma) = Instances_{d'}(\alpha, \Sigma)$, then α is* domain independent, *where all free variables range over d, d' and where quantifiers range over d, d', respectively. We assume that the parameters mentioned in Σ and α are a subset of both d, d'.*

The intuition is that for every free variable x in a query, it does not make sense to exclude the parameters either in the query or the knowledge base Σ from the possible set of values that x can take. This set of values is called the *active domain* of the query [1]. However, if $Instances(\alpha, \Sigma)$ contains tuples whose parameters are not members of the active domain, then the query is not paying attention to the information that is given; data is being used from "somewhere else", i.e. from the underlying domain [51]. Since we have assumed that our domain is infinite, then it make sense to ensure that queries pay attention to the active domain.

It is interesting to note why we have not ignored domain independence and *directly* evaluated queries with respect to the active domain. This could happen if we let d in $Instances_d(\alpha, \Sigma)$ range over the parameters mentioned in Σ, or inserted *domain closure axioms* in Σ [43]. This no doubt guarantees that our queries are *safe*. However, this benefit is achieved at the expense of keeping the user ignorant of information that is not easily accessible to aid in the phrasing of queries [1]. Thus, queries that are domain independent ensure that the answers are true with respect to any underlying domain, up to isomorphism. The semantic specification is thus cleaner because the user need not worry about how queries are evaluated, or relative to what domain, but only on their specification. Hopefully a few examples will clarify this property.

Example 3.4.18 The following formulas are domain independent: $KP(x)$ and $\exists x \exists y(P(x) \vee Q(y))$ since, in the former case, x will be bound to parameters that actually occur in the elementary knowledge base. Because we assume domains are identical across possible worlds, then the K operator introduces no new parameters. In the latter case, it is a sentence which depends only on the content of P and Q relations. Similarly, $P(x) \vee Q(x)$ is domain independent since when either x is bound to a parameter p such that the disjunct is true, then the

other disjunct must be bound to that parameter is well; x does not depend on the underlying domain but only on the parameters in either relation P or Q. Clearly, all the valid first-order sentences are domain independent as well [50].

Example 3.4.19 The following are not domain independent: $\neg P(x)$ and $P(x) \vee KQ(y)$ since, in the former case, if we let $d = \mathbb{P}$, then $Instances_d(\neg P(x), \{P(p_1)\})$ is infinite while for $d = \{p_1, p_2, p_3\}$, it is not. In the latter case, if $d = \mathbb{P}$, then $Instances_d(P(x) \vee KQ(y), \{P(p_1)\})$ is infinite while for finite d, it is not because it is asking us to list the instances of P or the known instances of Q. So if $P(p_1)$ follows from Σ, then the disjunction is true and since y is free, we get known and *unknown* instances of Q, which is infinite if $d = \mathbb{P}$, and finite otherwise. Similarly, $P(x) \vee Q(y)$ is not domain independent. It is important to note that none of these examples are admissible.

From these last examples, one can gauge the importance of disjunctively linked variables, not only for safety, but also for domain independence. Without such a restriction, even positive existential formulas are not domain independent, as shown above.

Lemma 3.4.20 *Positive existential formulas with disjunctively linked variables α are domain independent queries with respect to an elementary knowledge base Σ mentioning only finite many parameters.*

Proof: By induction on the shape of the query α.

Base: If α is atomic with free variables $\overrightarrow{x}$ and if $\alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$ is true in a first-order model M with the property that it mentions only parameters occurring in Σ (guaranteed by Lemma 3.4.10), then $\overrightarrow{p}$ consists of parameters occurring only in Σ. Call this set of parameters $active(\Sigma)$. Therefore, $Instances_d(\alpha, \Sigma) = Instances_{d'}(\alpha, \Sigma)$ for any two $d, d' \subseteq \mathbb{P}$, since $active(\Sigma) \subseteq d, d'$ by definition. Therefore α is domain independent.

Inductive: The result follows trivially for α of the form $\alpha_1 \wedge \alpha_2$. When α is of the form $\alpha_1 \vee \alpha_2$ then either $\alpha_1|\frac{\overrightarrow{x}}{\overrightarrow{p}}$ is true in M or $\alpha_2|\frac{\overrightarrow{x}}{\overrightarrow{p}}$ is. Since $\alpha_1 \vee \alpha_2$ has disjunctively linked variables, every variable of $\overrightarrow{x}$ occurs in both α_1, α_2, so the result follows by induction. Finally, when α is of the form $\exists y \alpha_1$ then for

some parameter π, $\alpha_1|^{\vec{x},y}_{\vec{p},\pi}$ is true in M. By induction $\vec{p},\pi$ are mentioned in Σ, and we are done. □

We can now extend domain independence to $KFOPCE$ queries admissible wrt $\mathcal{F}_\Sigma$.

Theorem 3.4.21 *$KFOPCE$ formulas admissible wrt $\mathcal{F}_\Sigma$ are domain independent.*

Proof: By Proposition 3.3.6 above and the definition of $\approx$, $\Sigma \approx \alpha$ iff $\Sigma \models_{FOPCE} \alpha$ iff $\Sigma \models_{KFOPCE} K\alpha$ for p.e. α with disjunctively linked variables. Therefore, formulas $K\alpha$ for p.e. α with disjunctively linked variables are also domain independent. This completes the base case.

For the inductive case, if α is of the form $\neg\alpha_1$ then since α is an admissible formula wrt $\mathcal{F}_\Sigma$, then α_1 must be a sentence and domain independent by induction. Then $Instances_d(\alpha,\Sigma) = Instances_{d'}(\alpha,\Sigma) = \emptyset$ for any two $d, d' \subseteq \mathbb{P}$. If α is of the form $\exists y\alpha_1$ and if $\exists y\alpha_1$ is not domain independent, then $\pi \in d'$ but $\pi \notin d$ for parameter π such that $\Sigma \approx \alpha_1|^{y}_{\pi}$. So $\pi \in Instances_{d'}(\alpha,\Sigma)$ but $\pi \notin Instances_d(\alpha,\Sigma)$. This means that $\alpha_1|^{\vec{x},y}_{\vec{p},\pi}$ is not domain independent either. This contradicts our induction hypothesis. Finally, if α is of the form $\alpha_1 \wedge \alpha_2$ then the result follows trivially by induction. □

Observe that domain independence implies safety, since if a query q is domain independent, it must return the same answers in both a finite *and* infinite domain. The converse does not hold, however, as we have seen. Theorem 3.4.21 is therefore an alternative proof that the instances of admissible formulas are finite. We mention both proofs because Reiter's [45] development of safe queries is motivated by equally important, and perhaps more intuitive considerations of finiteness of query answers.

Corollary 3.4.22 (Abiteboul et al. [1]) *Domain independent formulas are safe.*

What we have is a class of $KFOPCE$ queries (i.e. those admissible wrt $\mathcal{F}_\Sigma$ for elementary knowledge base Σ) that are *domain independent* and hence *safe*. This is a straightforward yet important extension of Reiter's [45] results on the class of admissible formulas wrt $\mathcal{F}_\Sigma$ since it explicitly addresses domain independence.

3.4.3 On Closure Constraints

Relational databases are assumed to satisfy the *closed-world assumption*, where facts that are not in the database are assumed to be false [43]. In the context of $\models\!\!\!\approx$, we want to have queries such that they are answers wrt $\models\!\!\!\approx$ iff they are answers under a closed world assumption. And we certainly want these queries to be a subset of the admissible queries we defined above. However, this assumes that Σ has definite information about what is true of the world; incomplete knowledge characterized by $\exists\alpha$ and $\alpha \vee \beta$ will not be allowed.

To begin with, we need to define what we mean by "closure".

Definition 3.4.23 (Closure [45])

$$Closure(\Sigma) = \Sigma \cup \{\neg\pi \mid \pi \text{ is an atomic sentence and } \Sigma \not\models_{FOPCE} \pi\}$$

Some Properties of $Closure(\Sigma)$

Proposition 3.4.24 (Reiter [45]) $|\mathcal{M}(Closure(\Sigma))| \leq 1$.

This means that:

Lemma 3.4.25 ([45]) *If α is a FOPCE sentence and Γ is a knowledge base, then:*

$$Closure(\Gamma) \models_{FOPCE} \alpha \text{ or } Closure(\Gamma) \models_{FOPCE} \neg\alpha.$$

Theorem 3.4.26 *When Σ is a database, $\mathcal{M}(Closure(\Sigma))$ is non-empty.*

Proof: The only way for $\mathcal{M}(Closure(r))$ to be empty is when $Closure(r)$ has no models, or is unsatisfiable. Since r is satisfiable by Lemma 3.4.10, then $Closure(r)$ can only be unsatisfiable if $\alpha, \neg\alpha \in Closure(r)$. Since $\neg\alpha \in Closure(r)$ iff $r \not\models_{FOPCE} \alpha$. But if $\alpha \in Closure(r)$, then $\alpha \in r$ because r is a database with only atomic sentences. This means that $r \not\models_{FOPCE} \alpha$ if $\neg\alpha \in Closure(r)$. This is impossible. □

Corollary 3.4.27 $|\mathcal{M}(Closure(\Sigma))| = 1$

Proof: By Proposition 3.4.24 and Theorem 3.4.26. □

Now, since $\mathcal{M}(Closure(\Sigma)) \subseteq \mathcal{M}(\Sigma)$ when Σ is a database, then some queries are true with respect to $Closure(\Sigma)$ that are not true with respect to Σ. A simple example is querying negative information. If we let $\Sigma = \{Q(a)\}$ then $\Sigma \not\approx \neg K \neg Q(b)$ but $Closure(\Sigma) \not\approx K \neg Q(b)$.

So we define a set of sentences that will allow us to use $\not\approx$ to query Σ directly rather than on $Closure(\Sigma)$, since this set is potentially infinite, but will also give us equivalent answers as if we had queried $Closure(\Sigma)$ directly.

Definition 3.4.28 ([45]) *Let α be a FOPCE formula. The KFOPCE formula $\mathcal{K}(\alpha)$ is defined as follows:*

1. *If α is an atom, then $\mathcal{K}(\alpha) = K\alpha$.*
2. $\mathcal{K}(\neg\alpha) = \neg\mathcal{K}(\alpha)$.
3. $\mathcal{K}(\exists x\alpha) = \exists x\mathcal{K}(\alpha)$.
4. $\mathcal{K}(\alpha_1 \wedge \alpha_2) = \mathcal{K}(\alpha_1) \wedge \mathcal{K}(\alpha_2)$

Since we are essentially putting every atomic formula in the immediate scope of a K, every $\mathcal{K}(\alpha)$ formula is subjective without iterated modalities. We call such formulas K_1 formulas. We can now evaluate these queries with respect to knowledge base Γ with the following theorem.

Theorem 3.4.29 (Reiter [45]) *Suppose α is a FOPCE sentence and Γ is a knowledge base such that $Closure(\Gamma)$ is satisfiable. Then,*

$$Closure(\Gamma) \models_{FOPCE} \alpha \text{ iff } \Gamma \not\approx \mathcal{K}(\alpha).$$

Reiter [45] does not pursue further the topic of a class of formulas α for which $\mathcal{K}(\alpha)$ is admissible, which we do here.

Definition 3.4.30 *Let AC be the smallest set such that:*

- *If α is an atomic formula, then $\alpha \in AC$,*
- *if $\alpha \in AC$, then $\exists x\alpha \in AC$,*
- *if $\vec{x}$ are the free variables of $\alpha_1 \in AC$, and if $\alpha_2|^{\vec{x}}_{\vec{p}} \in AC$ for all parameters $\vec{p}$, then $\alpha_1 \wedge \alpha_2 \in AC$,*

- *if* $\alpha_1 \in AC$ *is a sentence, then* $\neg\alpha_1 \in AC$.

We furthermore assume that all quantified variables are distinct from each other and from the formula's free variables.

Example 3.4.31 The following are AC formulas: $P(x) \wedge \neg Q(x,y)$, $\exists x Q(x,y)$ and $\neg\exists x P(x)$. The following are not AC formulas: $\neg P(x) \wedge \neg Q(x,a)$ for parameter a, $\exists y \neg \exists x Q(x,y)$ and $P(x) \wedge \neg Q(x,y)$.

Theorem 3.4.32 $\mathcal{K}(\alpha)$ *is admissible wrt* $\mathcal{F}_\Sigma$ *for* $\alpha \in AC$.

Proof: Note that if Σ is a database, then it is also an elementary knowledge base (See Remark 3.4.9).

If α is an atomic formula, then $K\alpha$ is admissible by definition of admissiblity wrt $\mathcal{F}_\Sigma$ and Remark 3.4.16. This completes the base case. For the inductive cases, assume that $\mathcal{K}(\alpha_1), \mathcal{K}(\alpha_2)$ are admissible, for $\alpha_1, \alpha_2 \in AC$. For α of the form $\exists x \alpha_1$, then $\exists x \mathcal{K}(\alpha_1)$ is admissible since an admissible subjective formula is in the scope of $\exists$. When α is of the form $\neg\alpha_1$, then $\neg\mathcal{K}(\alpha_1)$ is admissible since α_1 is a subjective sentence and admissible by induction. Finally, when α is of the form $\alpha_1 \wedge \alpha_2$ then the result follows trivially by induction. □

Remark 3.4.33 Of particular interest in the definition of $\mathcal{K}(\alpha)$ is the construction rule for $\exists x \alpha$. It will always be replaced by a $KFOPCE$ formula of the form $\exists x K\alpha$; therefore, we will not be able to ask Σ whether there exists an x such that α without knowing what that x *is* (i.e. $K\exists x \alpha$). Therefore, the formulas $\mathcal{K}(\alpha)$ are a proper subset of the admissible formulas wrt $\mathcal{F}_\Sigma$.

This may seem like a limitation of the query language if we are discussing a single relational database, but for our particular application, we will be quantifying over the first-order *models* of a *set* of databases, so we can evaluate queries with respect to $Closure(\Sigma)$ as opposed to Σ itself. But it is comforting to know that we can use $\mid\!\approx$ to query Σ directly *as if* $Closure$ is applied.

We have a stronger and novel result, however. Admissible formulas can also be used to query a database Σ *as if* we were querying $Closure(\Sigma)$.

Lemma 3.4.34 *Whenever* α *is a positive existential formula and* Σ *a database instance,* $Closure(\Sigma) \mid\!\approx \alpha$ *iff* $r \mid\!\approx \alpha$.

Proof: ($\Leftarrow$) This direction is trivial since $\mathcal{M}(Closure(\Sigma)) \subseteq \mathcal{M}(\Sigma)$.

($\Rightarrow$) The proof is by induction. For the base case, if α is an atomic formula, then we have $Closure(\Sigma) \models\!\!\sim \alpha|^{\vec{x}}_{\vec{p}}$. This means that $\alpha|^{\vec{x}}_{\vec{p}} \in \Sigma$ otherwise $r \not\models_{FOPCE} \alpha|^{\vec{x}}_{\vec{p}}$. But then $Closure(\Sigma) \models\!\!\sim \neg\alpha|^{\vec{x}}_{\vec{p}}$, contradiction.

For the inductive cases, assume that for α_1, α_2 we have $Closure(\Sigma) \models\!\!\sim \alpha_1|^{\vec{x}}_{\vec{p}} \wedge \alpha_2|^{\vec{y}}_{\vec{q}}$ iff $\Sigma \models\!\!\sim \alpha_1|^{\vec{x}}_{\vec{p}} \wedge \alpha_2|^{\vec{y}}_{\vec{q}}$. If α is of the form $\alpha_1 \wedge \alpha_2$, the the result follows trivially by induction. When α is of the form $\alpha_1 \vee \alpha_2$, then we have $Closure(\Sigma) \models\!\!\sim (\alpha_1 \vee \alpha_2)^{\vec{x},\vec{y}}_{\vec{p},\vec{q}}$. If $\Sigma \not\models\!\!\sim (\alpha_1 \vee \alpha_2)^{\vec{x},\vec{y}}_{\vec{p},\vec{q}}$, then we have $\Sigma \models\!\!\sim \neg K(\alpha_1 \vee \alpha_2)^{\vec{x},\vec{y}}_{\vec{p},\vec{q}}$. This holds iff $\Sigma \models\!\!\sim \neg K\alpha_1|^{\vec{x}}_{\vec{p}}$ and $\Sigma \models\!\!\sim \neg K\alpha_2|^{\vec{y}}_{\vec{q}}$. By ($\Leftarrow$) we have $Closure(\Sigma) \models\!\!\sim \neg K\alpha_1|^{\vec{x}}_{\vec{p}}$ and $Closure(\Sigma) \models\!\!\sim \neg K\alpha_2|^{\vec{y}}_{\vec{q}}$. This implies that $Closure(\Sigma) \not\models\!\!\sim (\alpha_1 \vee \alpha_2)^{\vec{x},\vec{y}}_{\vec{p},\vec{q}}$, contradiction. Finally, when α is of the form $\exists x\alpha_1$, then we have $Closure(\Sigma) \models\!\!\sim \alpha_1|^{x,\vec{y}}_{p,\vec{q}}$. By ($\Leftarrow$), we have $\Sigma \models\!\!\sim \alpha_1|^{x,\vec{y}}_{p,\vec{q}}$ which implies $\Sigma \models\!\!\sim \exists x\alpha_1|^{\vec{y}}_{\vec{q}}$. □

Theorem 3.4.35 *Whenever α is admissible wrt $\mathcal{F}_\Sigma$ and Σ a database instance, $Closure(\Sigma) \models\!\!\sim \alpha$ iff $r \models\!\!\sim \alpha$.*

Proof: ($\Leftarrow$) This direction is trivial since $\mathcal{M}(Closure(\Sigma)) \subseteq \mathcal{M}(\Sigma)$.

($\Rightarrow$) The base case follows by Lemma 3.4.34. The inductive cases proceed thus: when α is of the form $K\alpha_1$, then by Corollary 3.2.9 and that database instances are satisfiable (Theorem 3.4.26) we have $Closure(\Sigma) \models\!\!\sim \alpha_1$. By induction the result follows. When α is of the form $\neg\alpha_1$, since α is admissible, then α_1 must be a subjective sentence. So either $\Sigma \models\!\!\sim \alpha_1$ or $\Sigma \models\!\!\sim \neg\alpha_1$ by Corollary 3.3.3. If the latter, then we are done. Otherwise, by ($\Leftarrow$) we have $Closure(\Sigma) \models\!\!\sim \alpha_1$, contradiction. Whenever α is of the form $\exists x\alpha_1$, then we have that $Closure(\Sigma) \models\!\!\sim \alpha_1|^{x,\vec{y}}_{p,\vec{q}}$. By induction, $\Sigma \models\!\!\sim \alpha_1|^{x,\vec{y}}_{p,\vec{q}}$ which implies $\Sigma \models\!\!\sim \exists x\alpha_1|^{\vec{y}}_{\vec{q}}$. Finally, when α is of the form $\alpha_1 \wedge \alpha_2$, the result follows trivially since for free variables $\vec{x}$ of α_1, $\alpha_2|^{\vec{x}}_{\vec{p}}$ is admissible. □

With this result, our AC formulas seem redundant. Yet we see in Chapter 4 that a set of first-order constraints called *dependencies* are AC formulas. And since we construe integrity constraints as subjective K_1 formulas, we can then use the results in this section to show that formulas $\mathcal{K}(AC)$ are admissible. So whenever Σ is a database instance, we can query it directly *as if* we were querying its $Closure$.

If, however, our query q is evaluated with respect to $Closure(\Sigma)$ then we have the following:

Theorem 3.4.36 (Reiter [45]) *If Σ is a knowledge base and q a $KFOPCE$ formula with free variables $\vec{x}$, then for all parameters $\vec{p}$*

$$Closure(\Sigma) \approx q^{\vec{x}}_{\vec{p}} \text{ iff } Closure(\Sigma) \models_{FOPCE} \overline{q}^{\vec{x}}_{\vec{p}}$$

where $\overline{q}$ is q with all occurrences of the K operator removed.

Under the closed-world assumption, query evaluation is equivalent to first-order entailment. We thus lose the distinctions provided by the K operator. Since we plan to query over a *set* of database instances, K allows us quantify over these possible states of affairs, or possible worlds.

3.5 Summary

Representing facts about some domain in some formalism is distinct from *interacting* with the system housing those facts; there is no reason to suppose that the representation language should be the same as the interaction language. Indeed, there are good reasons why we would want these to be different. One reason investigated in this chapter is that we are then able to ask a database system about *itself*, or about what it knows. We have phrased this specification in the epistemic query language $KFOPCE$, a subset of $\mathcal{KL}$.

We introduced the basic syntax and semantics of $KFOPCE$ and noted that our countably infinite supply of *parameters* required that our domain of quantification be countably infinite and pairwise distinct. This is enforced by defining a *world* as a set of *atomic sentences*, of which $p = p$ for parameters p are included and atoms $p = p'$ for distinct parameters p, p' are excluded. As a result of this, the *unique names assumption* is part of the logic.

As a set of world states, an *epistemic state* contains information about what the system *knows*; each world in the epistemic state is considered possible, where the system has not ruled out all possibilities as to the actual state of affairs. Thus, any formula within the scope of a K depends solely on the epistemic state. Since the semantics is equivalent to that of the logic $K45$, the *positive introspection*

and *negative introspection* axioms hold. Consequently, any *subjective sentence* is either known to be true or known to be false. This gave us that epistemic states always know what is true about its own knowledge, as well as that what is known about its knowledge will be true. Logical entailment between a set of $KFOPCE$ sentences Σ and $KFOPCE$ query α was defined as was entailment for $FOPCE$, the first-order counterpart of $KFOPCE$. If Σ is strictly first-order, and α $KFOPCE$, then we defined a new operator $\models\!\!\!\approx$. $\Sigma \models\!\!\!\approx \alpha$ iff $e_\Sigma \models K\alpha$, where e_Σ is the set of first-order models of Σ.

Up to this point in our explorations, the logic only satisfied the unique names assumption of Reiter's relational theory [43] for relational database systems. We therefore reiterated Reiter's [45] results ensuring finite answers for $KFOPCE$ queries with respect to a class of sets of sentences: *elementary knowledge bases.* We considered the effect such knowledge bases had on queries, with and without closure constraints. What we showed is that *admissible* $KFOPCE$ queries are sufficiently expressive yet scalable enough to prevent infinite answers, in addition to being *domain independent.* We furthermore isolated a class AC of queries that are admissible for closed-world databases. $KFOPCE$ as a query language is thus made more flexible with these results.

In the following chapter, we see that an important class of constraints, *dependencies* in $KFOPCE$, are admissible with respect to $\mathcal{F}_\Sigma$. And one of the benefits of treating integrity constraints as statements in $KFOPCE$ is that they capture intuitions about what an integrity constraint *is* and how they are *satisfied.* In accordance with Corollary 3.3.3, we show that such statements are either always known to be true or known to be false since they are purely subjective; integrity constraints or their negations are never unknown.

Moreoever, in the state in which no definite objective facts are known, e_0, we have that the system knows that it has incomplete knowledge of every aspect of the world. This epistemic state is important for integrity constraint satisfaction, because the idea is that if $\{\}$ is all that is known, then constraints should still be satisfied. All these properties are made explicit by defining *allowed* constraints as those that are subjective sentences without iterated modalities and are known in e_0.

Chapter 4

A Knowledge-Level View of Integrity Constraints

> *Integrity without knowledge is weak and useless, and knowledge without integrity is dangerous and dreadful.*
>
> - Samuel Johnson, *Rasselas*

From the definition of a *database instance* (see Definition 3.1.5), we have a limited way of representing truths about the world. Various frameworks for adding a richer semantics to the representation language have therefore been proposed. One such framework involves *integrity constraints*, which are statements that either restrict the allowable states of a database or specify relationships between data, relationships that aid in designing the database schema. We can further distinguish two types of integrity constraints: static and dynamic. Static integrity constraints enforce valid *current* states of a database while dynamic integrity constraints enforce valid *changes* of states. Since the problem we are modeling involves individual databases whose current state satisfies a given set of constraints but possibly globally violates those constraints, we focus on static integrity constraints.

In this chapter, we explore Reiter's [44] results on treating integrity constraints as statements about the *knowledge* a knowledge-based system has of its domain. This perspective nourishes certain intuitions about what it means for

a knowledge base to *satisfy* an integrity constraint, and makes $KFOPCE$ a formalization of such properties. Integrity constraints are sentences in our epistemic language $KFOPCE$ and constraint checking is equivalent to querying a knowledge base with a constraint. While Reiter [44] briefly outlines some restrictions on integrity constraints in $KFOPCE$, we provide an extended discussion of the various forms that such formulas can take and a desiderata on their appropriateness; any $KFOPCE$ sentence that satisfies these properties are *allowed*. We then isolate an important class of constraints called *dependencies* and show that they can be expressed in $KFOPCE$ and that in such form they are allowed and admissible wrt $\mathcal{F}_\Sigma$, where Σ is an elementary knowledge base and $\mathcal{F}_\Sigma$ is defined as in Theorem 3.4.14.

4.1 Integrity Constraints in $KFOPCE$

The standard view of integrity constraints is that they are statements about the world the database should model. They have been used to describe relationships between information in databases [1, 51, 52] and traditionally formulated in the same language as the model itself, which has been first-order logic [40, 43]. As such, integrity constraint *satisfaction* is defined in terms of either first-order entailment or satisfiability [20].

Consider the *entailment-based* definition given by Reiter [43] which states that a database Σ satisfies an integrity constraint IC iff $\Sigma \models IC$. Contrast this with the *consistency-based* definition given by Kowalski [28]: Σ satisfies IC iff $\Sigma + IC$ is satisfiable. Under the closed world assumption, these two are equivalent [45] but for general knowledge bases they are not. This divergence can be found in how these definitions of constraint satisfaction do not respect our intuitions as to what an integrity constraint *is*. Let IC be the $FOPCE$ constraint $\forall x[emp(x) \supset (\exists y)ssn(x,y)]$, which states that all employees must have a social security number.

1. Let $\Sigma = \{emp(mary)\}$. $\Sigma + IC$ is satisfiable in a world w such that $w \models emp(mary) \wedge ssn(mary, 123)$. Yet it seems that Σ should not satisfy IC if Σ is *all that is known.*
2. If $\Sigma = \{\}$, then $\Sigma \not\models IC$ since IC is not a logical truth. Intuitively, this

should satisfy IC since *nothing is known* that violates IC.

Reiter [44] suggests that integrity constraints should be construed as statements about *knowledge*, or about what a knowledge base knows. Rewriting the above IC in the idiom of $KFOPCE$, the constraint should be read as $\forall x[Kemp(x) \supset (\exists y)Kssn(x,y)]$, or "For every *known* employee there is a *known* social security number." The functional dependency $\forall x, y, z[ssn(x,y) \wedge ssn(x,z) \supset y = z]$ should be rewritten as $\forall x, y, z[Kssn(x,y) \wedge Kssn(x,z) \supset Ky = z]$. These modalized sentences represent truths about the knowledge of a knowledge base Σ, whereas the sentences in Σ represent truths about the world; integrity constraints are not sentences of Σ itself. Thus, to say that a knowledge base Σ *satisfies* an integrity constraint is to say that the constraint is *known to be true.* To test whether a constraint is satisfied or not, we query Σ using $\models\!\!\!\approx$.

Definition 4.1.1 (Integrity Constraint Satisfaction [45]) *When σ is a $KFOPCE$ sentence, and Σ is a knowledge base,* Σ satisfies the integrity constraint σ *iff* $\Sigma \models\!\!\!\approx \sigma$, *or equivalently* $e_\Sigma \models K\sigma$. *We say that a constraint is* violated *when* $\Sigma \not\models\!\!\!\approx \sigma$, *or equivalently when* $e_\Sigma \models \neg K\sigma$.

Let us return to our two counterexamples above. Do they apply to integrity constraint satisfaction in $KFOPCE$? Consider $\Sigma = \{emp(mary)\}$ and $\sigma := \forall x[Kemp(x) \supset (\exists y)Kssn(x,y)]$, where *mary* is assumed to be a parameter. It is easy to see that $\Sigma \not\models\!\!\!\approx \sigma$ since there is no known social security number for Mary. That is, there is no single social security number that is Mary's for every world in e_Σ. For $\Sigma = \{\}$, however, e_Σ is the set of *all* worlds, which represents the epistemic state in which no definite objective facts are known, e_0 (Recall Theorem 3.3.4). The antecedent in σ therefore never holds in e_0 and $K\sigma$ is trivially true in e_0; $\Sigma \models\!\!\!\approx \sigma$. So integrity constraint satisfaction in $KFOPCE$ does not run afoul of these counterexamples.

That integrity constraints should hold in e_0 does not mean that they need to be *valid*, since this is not the case, but only that they do not specify that any objective formula need be known without there being any sentences in Σ, apart from the first-order valid sentences [30]. Since integrity constraints are statements about the knowledge a database has of its domain, and are hence subjective sentences, the question is how to characterize the subjective sentences

that are true in e_0. The problem, however, is that the definition of constraint satisfaction only requires that σ be a $KFOPCE$ sentence. No other restrictions on the form of σ are stated, or that it be satisfied by e_0. Thus, constraints can be represented in any number of ways.

Example 4.1.2 The functional dependency in $FOPCE$:

$$\forall x, y, z[ssn(x, y) \wedge ssn(x, z) \supset y = z]$$

can be represented as either:

$$\forall x, y, z[Kssn(x, y) \wedge Kssn(x, z) \supset Ky = z], \tag{4.1}$$

$$\forall x, y, zK[ssn(x, y) \wedge ssn(x, z) \supset y = z] \text{ or}, \tag{4.2}$$

$$K\forall x, y, z[ssn(x, y) \wedge ssn(x, z) \supset y = z]. \tag{4.3}$$

Observe that 4.3 logically implies the other two but only 4.1 holds in e_0.[1]

Example 4.1.3 Consider the constraint that no individual is both a male and a female. Since we have specified this in natural language, it is not immediately obvious what $KFOPCE$ sentence should be used. This constraint can be represented as one of the following:

$$\forall x\neg(Kmale(x) \wedge Kfemale(x)), \tag{4.4}$$

$$\forall x\neg K(male(x) \wedge female(x)), \tag{4.5}$$

$$\forall xK\neg(male(x) \wedge female(x)), \text{ or} \tag{4.6}$$

$$K\forall x\neg(male(x) \wedge female(x)). \tag{4.7}$$

The first and second are implied by the other two, which are equivalent, and only 4.4 and 4.5 are satisfied by e_0 since we can find a $w \in e_0$ such that $w \models male(a) \wedge female(a)$ in which case neither 4.6 nor 4.7 are satisfied. In this case, however, both 4.4 and 4.5 are satisfied since there is no single individual a that is both male and female in *all* worlds $w \in e_0$.

What if we chose to ignore the property of constraints that they be known in e_0? Is there an intuitive reason that we should choose 4.4 or 4.5? In a system that

[1]Formula 4.2 is actually equivalent to 4.3 by Theorem 3.2.10.

acquires its knowledge incrementally (i.e. does not have a complete description of the world at any one time), are we to enforce, as in 4.6 and 4.7, that the knowledge of the system should never consider that an individual a is both male and female? If so, this interpretation precludes situations where an individual's gender *becomes known* after it acquires more knowledge; in a situation in which an agent considers all worlds possible, nothing is known about the world, except objective validities, and nothing is known that violates the constraints. Similarly, if the system receives conflicting reports from multiple sources, the system will be inconsistent with its integrity constraints. But if we want to maintain integrity and retain as much information as possible, it seems plausible that we prescribe incomplete knowledge of definite *objective* information, as opposed to complete knowledge of *objective* negative information. So we do not want to consider an agent's knowledge inconsistent *until* it receives more information about the world.

That being said, since our system has complete knowledge of its epistemic state, it is not at all obvious why iterated modalities should be considered [44]. We enumerate some properties that a $KFOPCE$ integrity constraint should (arguably) have and then go on to show in the next section that a class of useful constraints have these properties when phrased in $KFOPCE$.

Definition 4.1.4 (Allowed Integrity Constraint) *An integrity constraint σ will be* allowed *if it satisfies the following properties:*

1. *σ is a subjective sentence.*
2. *σ does not have iterated modalities. We call subjective formulas without iterated modalities K_1 formulas.*
3. *$\Sigma \not\approx \sigma$ where $\Sigma = \{\}$.*

Remark 4.1.5 The set of allowed integrity constraints is not a subset of the admissible formulas wrt $\mathcal{F}_\Sigma$, where Σ is an elementary knowledge base. Why is this true? Consider $\sigma := \exists x \neg KP(x)$. This holds in e_0 since nothing is known in that epistemic state; there are no known P instances in e_0. However, σ is not admissible wrt $\mathcal{F}_\Sigma$ because the negation sign does not quantify over a sentence.

Why is admissibility not a stipulated property of allowed constraints? In Reiter's [44] definition of constraint satisfaction, given above, Σ is a knowledge base; it need not be elementary. Since admissibility is defined relative to Σ and $\mathcal{F}_\Sigma$, and since we have assumed that Σ is an elementary knowledge base, we did not choose to restrict allowed constraints by the assumption of the knowledge base to which they apply. Admissibility is particular to the Prolog-style evaluator that Reiter [45] provides, and is therefore not a general property that integrity constraints should have. Admissibility implies a certain syntactic form for constraints, and it is not immediately obvious what form allowed integrity constraints should take aside from those enforced by the properties above. Because of this—or in spite of this—we isolate a large class of constraints called *dependencies* studied extensively in the relational database literature [1, 23, 52], which have a particular syntactic form which most integrity constraints, we will assume, possess. We show that they can be expressed in $KFOPCE$ and prove that they are allowed and admissible with respect to $\mathcal{F}_\Sigma$.

4.2 Dependencies in $KFOPCE$

Not only does the use of integrity constraints enhance the expressivity of the relational model, by allowing the designer to specify legal database states, but it also permits the designer to design a *better* relational schema using relationships that are already inherent in the database [52]. We focus on a specific class of constraints called *dependencies*, which have a form general enough for many relationships to be expressed [1, 24, 52].

We present two examples that highlight the form these dependencies take before proceeding with the formal development.

Consider the database instance Σ:

$$\begin{aligned}\Sigma \;=\; & \{P(hilbert, math, gauss), P(pythagoras, math, gauss),\\ & P(turing, cmpt, von\_neumann)\}\end{aligned}$$

where $P(a, b, c)$ says that a is in department b, whose manager is c. The problem is that that Gauss is the manager of the Math department is redundant, and the updating of Hilbert's manager to Von Neumann indicates that Math has two

managers, a change that was probably not intended [52]. So the dependency we isolate here is that every department has a unique manager; there is a *functional dependency* between departments and managers. We can define two new relations S, T that do not suffer from these "anomalies".

$$\begin{aligned}\Sigma \;=\; & \{S(math, gauss), S(cmpt, von\_neumann),\\ & T(hilbert, math), T(pythagoras, math), T(turing, cmpt)\}\end{aligned}$$

$S(a, b)$ says that department a has b as a manager. $T(c, d)$ says that c is a member of department d.

Written in first-order logic, the functional dependency (fd) between departments and managers can be written as:

$$\forall x, y, z \; (S(x, y) \wedge S(x, z) \supset y = z).$$

However, even in the absence of functional dependencies, dependencies still exist. Other dependencies may be inherent in the data. We consider *multivalued dependencies* as an example. Let Σ be the following database:

$$\Sigma = \{R(pythagoras, peter, math), R(pythagoras, paul, math)\}$$

where $R(a, b, c)$ says that b is the child of a and that a has the skill c. Again, there is redundancy in the fact that Pythagoras is skilled in mathematics and the deletion of any one of these tuples could be seen as an indication that Pythagoras is no longer skilled in math and that Peter or Paul is no longer his child [52]. The multivalued dependency (mvd) inherent in this database is that the first argument determines a *set* of children and a *set* of skills. The solution to this problem is to decompose R into two relations relating a to b and another relating a to c. In first-order logic, this mvd can be written as the following sentence:

$$\forall u, x_1, x_2, y_1, y_2 \; (R(u, x_1, y_1) \wedge R(u, x_2, y_2) \supset R(u, x_1, y_2)).$$

There are many more semantic constraints and dependencies that have been studied, which we do not explore [e.g. embedded mvds, inclusion dependencies, join dependencies, etc.]. Fortunately, a unifying characterization has been developed for all these types of dependencies that relies on the fact that they say,

roughly: "if you see a certain tuple or tuples in a relation, then you will also see *this* particular pattern or tuple", where *this* can either refer to the equality of certain values, as in the case of fds, or can refer to the existence of a tuple in a particular relation, as in the case of mvds [52]. Other classes of constraints can be transformed into this form, according to Gertz [24].

Formalizing these intuitions, we have the following definition:

Definition 4.2.1 (Dependencies [52]) *A* dependency *is a FOPCE sentence*

$$\forall x_1 \ldots x_k \exists y_1 \ldots \exists y_l [(A_1 \wedge \ldots \wedge A_p) \supset (B_1 \wedge \ldots \wedge B_q)]$$

where the A_i's are non-equality atomic formulas in $x_1 \ldots x_k$, B_j's are atomic formulas and $p, q, k \geq 1$ and $l \geq 0$. When $l = 0$, the dependency is called full. *If the B_j's consist of equality formulas, then the dependency is an* equality-generating dependency *(egd). Otherwise, it is called a* tuple-generating dependency *(tgd).*

Remark 4.2.2 There is another restriction when $l \geq 1$: no equality atoms in the consequent can involve existentially quantified variables [1].

In $KFOPCE$, we write dependencies according to the following definition.

Definition 4.2.3 *A $KFOPCE$* dependency *is a $KFOPCE$ sentence*

$$\forall x_1 \ldots x_k \exists y_1 \ldots y_l [(KA_1 \wedge \ldots \wedge KA_p) \supset (KB_1 \wedge \ldots \wedge KB_q)].$$

where the A_i's are non-equality atomic formulas in $x_1 \ldots x_k$, B_j's are atomic formulas and $p, q, k \geq 1$ and $l \geq 0$. When $l = 0$, the dependency is called full. *We overload the definition of egds and tgds for $KFOPCE$ dependencies where required. So, if the B_j's consist of equality formulas, then the dependency is an* equality-generating dependency *(egd). Otherwise, it is called a* tuple-generating dependency *(tgd).*

Does satisfaction of the first-order dependency imply satisfaction of the $KFOPCE$ dependency? Unfortunately, this is not the case because of the existential quantifiers. Since $\Sigma \models\!\!\!\approx \alpha$ iff $\Sigma \models\!\!\!\approx K\alpha$ by Theorem 3.2.6, we can "distribute" K past the universal quantifiers due to Theorem 3.2.10. Therefore, for full dependencies, satisfaction of the first-order dependency implies satisfaction of the $KFOPCE$ dependency (see Example 4.1.2), so no expressive power is lost by considering full dependencies in $KFOPCE$.

Proposition 4.2.4 *When Σ is a knowledge base, σ is a first-order full dependency and σ' is its KFOPCE dependency written as:*

$$\forall x_1 \ldots x_k \exists y_1 \ldots y_l [(KA_1 \wedge \ldots \wedge KA_p) \supset (KB_1 \wedge \ldots \wedge KB_q)]$$

then

$$\Sigma \models_{FOPCE} \sigma \text{ implies } \Sigma \mid\!\approx \sigma'.$$

Proof: By Proposition 3.3.6, we have that $\Sigma \models_{FOPCE} \sigma$ iff $\Sigma \mid\!\approx \sigma$. $\models K\sigma \supset \sigma'$ by Theorems 3.2.5 and 3.2.10. Finally, by Theorem 3.3.7, $\Sigma \mid\!\approx \sigma'$. □

Yet the existential quantifiers pose a problem for dependencies in $KFOPCE$. In general, it is the case that $\not\models K\exists x P(x) \supset \exists x KP(x)$ by Theorem 3.2.11 but $\models \exists x KP(x) \supset KP(p)$ for some parameter p. But when Σ is a database, we have $\Sigma \mid\!\approx \exists x(KP(x) \wedge x = p)$ iff $\Sigma \mid\!\approx \exists x(P(x) \wedge x = p)$ when $P(a) \in \Sigma$. Thus, having the B_j immediately within the scope of K has the advantage of allowing the dependency to talk about *knowing* individuals *in the database*, which seems like the intended interpretation of first-order dependencies in a database setting since $\Sigma \mid\!\approx \exists x P(x)$ means that $P(p)$ is in the database for some parameter p. This reveals an ambiguity between $\exists x K\alpha$ and $K\exists x\alpha$ when Σ is a database.[2]

So atomic formulas are immediately in the scope of K operators because $FOPCE$ dependencies say "if you see a certain tuple or tuples in a relation, then you will also see *this* particular pattern or tuple". However, if we use the language of $KFOPCE$, then this could be read as: "if a certain tuple or tuples in a relation *are known*, then *this* particular pattern or tuple will also be *known*". Since the contents of databases are atomic sentences, it makes intuitive sense to specify what is known in terms of the types of formulas that appear in the actual database. Consequently, a *referential constraint* such as $\forall x[emp(x) \supset \exists y\ salary(x, y)]$ would be reinterpreted as $\forall x[Kemp(x) \supset \exists y K salary(x, y)]$.

We can thus rewrite this $KFOPCE$ dependency in an equivalent form based on the transformation rules given by Lloyd and Topor [27] for extended logic programs, and adapted by Reiter [45] for $KFOPCE$ formulas (note that the A_i

[2]See Example 5.1.13 in Chapter 5 for an extended discussion.

are atomic formulas are formulas in $x_1, \ldots, x_k$ as per Definition 4.2.3):

$$\neg[\exists x_1 \ldots x_k]((KA_1 \wedge \ldots \wedge KA_p) \wedge \neg[\exists y_1 \ldots \exists y_l](KB_1 \wedge \ldots \wedge KB_q))$$

Admissibility of Dependencies

The above form is also motivated by the following observation. Consider the *FOPCE* dependency:

$$\forall x_1 \ldots x_k[(A_1 \wedge \ldots \wedge A_p) \supset \exists y_1 \ldots \exists y_l(B_1 \wedge \ldots \wedge B_q)].$$

It turns out that this *FOPCE* dependency is an *AC* formula (recall Definition 3.4.30) when written in the following form (again, by the Lloyd and Topor [27] rules):

$$\neg[\exists x_1 \ldots x_k][(A_1 \wedge \ldots \wedge A_p) \wedge \neg[\exists y_1 \ldots \exists y_l](B_1 \wedge \ldots \wedge B_q)].$$

Theorem 4.2.5 *FOPCE Dependencies written as*

$$\sigma := \neg[\exists x_1 \ldots x_k][(A_1 \wedge \ldots \wedge A_p) \wedge \neg[\exists y_1 \ldots \exists y_l](B_1 \wedge \ldots \wedge B_q)].$$

are AC formulas.

Proof: The A_i are members of *AC*; they are atomic. Since conjunctions are left associative in the definition of *AC*, then the entire $A_1 \wedge \ldots \wedge A_p$ is admissible. This latter is of course not a formal inductive proof.

Lemma 4.2.6 *$A_1 \wedge \ldots \wedge A_i$ is AC for all i.*

Proof: A_1 is *AC* by construction. This completes the base case.

Assume that $\alpha_i := A_1 \wedge \ldots A_i$ is *AC* for $i \geq 1$. Consider $\alpha_i \wedge A_{i+1}$ and let $\overrightarrow{x}$ be the free variables of α_i. $A_{i+1}|\frac{\overrightarrow{x}}{\overrightarrow{p}}$ is *AC* since it is an atomic formula. Since α_i is *AC* by induction, $\alpha_i \wedge KA_{i+1}$ is *AC* for all i. //[3]

Thus, $KA_1 \wedge \ldots \wedge KA_p$ is *AC* for $i = p$.

Next, the inner negation is permissible because by substituting all the free variables $\overrightarrow{x}$ of the subformula $A_1 \wedge \ldots \wedge A_p$ with parameters $\overrightarrow{p}$ in the subformula $\neg\exists y_1 \ldots \exists y_l(B_1 \wedge \ldots \wedge B_q)$, the resulting subformula is a sentence. The outer

[3] "//" indicates the completion of a proof of a lemma within the proof of the enclosing theorem.

negation is applied to a sentence. This makes σ AC. □

Notice then that if σ is the $FOPCE$ dependency as in the above theorem, then $\mathcal{K}(\sigma)$ (see Definition 3.4.28) is equivalent to the form of $KFOPCE$ dependencies in Definition 4.2.3. This shows that $KFOPCE$ dependencies are admissible.

Theorem 4.2.7 *When an $FOPCE$ dependency is written as*

$$\sigma := \neg[\exists x_1 \ldots x_k][(A_1 \wedge \ldots \wedge A_p) \wedge \neg[\exists y_1 \ldots \exists y_l](B_1 \wedge \ldots \wedge B_q)].$$

then $\mathcal{K}(\sigma)$ is admissible.

Proof: By Theorems 4.2.5 and 3.4.32. □

Henceforth, when we mention $KFOPCE$ dependencies, we envision $\mathcal{K}(\sigma)$ where σ is a $FOPCE$ dependency transformed by the Lloyd and Topor [27] rules, the final form of which is illustrated in the following theorem.

Theorem 4.2.8 *Dependencies written as*

$$\sigma := \neg[\exists x_1 \ldots x_k]((KA_1 \wedge \ldots \wedge KA_p) \wedge \neg\exists y_1 \ldots \exists y_l(KB_1 \wedge \ldots \wedge KB_q))$$

are allowed *integrity constraints.*

Proof: By construction, σ is subjective without iterated modalities. We have to show that $\Sigma \models \sigma$ for $\Sigma = \{\}$. Since no objective information is known in e_0, the antecedent of σ does not hold. The implication thus holds trivially. □

Corollary 4.2.9 *When Γ is a set of dependencies in $KFOPCE$, Γ is satisfiable.*

Proof: Since every dependency in $KFOPCE$ is satisfied in e_0, it has at least one model, (e_0, w) for any w. □

Notice that $KFOPCE$ dependencies exclude admissible and allowed constraints such as $\forall x KP(x) \supset K\exists x Q(x)$, or $\forall x KP(x) \supset \neg KQ(x)$ since neither are in the form of a dependency. Nonetheless, we consider such constraints when the need arises, but it is comforting to know that we have a large and useful class of $KFOPCE$ constraints that are applicable to databases.

Known Instances of Dependencies

Can we also ask the knowledge base to *list* all the known instances of these constraints? What does the resulting query look like? If we simply omit the universal quantifiers then we get:

$$\neg((KA_1 \wedge \ldots \wedge KA_p) \wedge \neg\exists y_1 \ldots y_l(KB_1 \wedge \ldots \wedge KB_q))$$

Unfortunately, this is not admissible since the outer negation is not applied to a subjective sentence. This makes intuitive sense. Consider the query $KP(x) \supset KQ(x)$. Whenever a parameter p is not known to have property P, then the entire implication is true and p is a known instance. However, the set of x for which $\neg KP(x)$ is true is potentially infinite if our knowledge base is finite. We therefore want to ask about the known *definite* instances. We can do that with the following formula, which we call σ_{free} whenever σ is a dependency:[4]

Definition 4.2.10 *Whenever σ is a dependency in KFOPCE, we denote σ_{free} as the following formula:*

$$(KA_1 \wedge \ldots \wedge KA_p) \wedge \neg((KA_1 \wedge \ldots \wedge KA_p) \wedge \neg[\exists y_1 \ldots y_l](KB_1 \wedge \ldots \wedge KB_q))$$

Theorem 4.2.11 *σ_{free} is admissible with respect to $\mathcal{F}_\Sigma$, where Σ is an elementary knowledge base and σ a dependency in KFOPCE.*

Proof: The A_i are members of $\mathcal{F}_\Sigma$; they are atomic and hence p.e. with disjunctively linked variables. Because the K operators have them as their scope, A_i are formulas in $x_1, \ldots, x_k$, and since conjunctions are left associative in the definition of admissibility, then the entire $KA_1 \wedge \ldots \wedge KA_p$ is admissible. This latter is of course not a formal inductive proof.

Lemma 4.2.12 *$KA_1 \wedge \ldots \wedge KA_i$ is admissible for all i.*

Proof: KA_1 is admissible by construction. This completes the base case.

Assume that $\alpha_i := KA_1 \wedge \ldots KA_i$ is admissible for $i \geq 1$. Consider $\alpha_i \wedge KA_{i+1}$ and let $\overrightarrow{x}$ be the free variables of α_i. $KA_{i+1}|\frac{\overrightarrow{x}}{\overrightarrow{p}}$ is admissible since the scope of K is an atomic formula. Since α_i is admissible by induction, $\alpha_i \wedge KA_{i+1}$ is

[4]Observe that σ_{free} is not allowed.

admissible for all i. //

Thus, $KA_1 \wedge \ldots \wedge KA_p$ is admissible for $i = p$.

For all parameters $\overrightarrow{p}$, $\neg((KA_1 \wedge \ldots \wedge KA_p) \wedge \neg[\exists y_1 \ldots y_l](KB_1 \wedge \ldots \wedge KB_q))|^{\overrightarrow{x}}_{\overrightarrow{p}}$ is a sentence. Therefore, the outer negation is applied to a subject sentence and is hence admissible. □

Henceforth, whenever we ask for the instances of a dependency σ, this form is the one we envision, which we write as σ_{free}.

4.3 Summary

The results presented in this chapter are supported by Reiter's [44, 45] results on treating integrity constraints as statements about the knowledge a knowledge base has of its domain, as opposed to statements about the world. While Reiter's work provides a simple discussion on the actual form that such constraints take, that they should hold in e_0 has not been subsequently acknowledged; we made such obligations explicit here by defining *allowed* integrity constraints. We furthermore showed that dependencies expressed in $KFOPCE$ are both admissible and allowed.

What remains is to show how the foundation we have built in previous chapters provides a knowledge-level formalization of what Arenas et al. [4] call *consistent query answers*, which are query answers that are true in all repairs of a database inconsistent with its integrity constraints. This is the topic of the next chapter.

Chapter 5

Consistent Query Answers

It was her voice that made The sky acutest at its vanishing.

- Wallace Stevens, "The Idea of Order at Key West"

We begin with an example. Consider the constraint "Social security numbers of employees are unique". Suppose that an employee database of Acme Inc. in the City X has an employee named Jane Smith whose social security number is '123'. Jane Smith in City Y has a social security number '456' and also works for Acme. The individual employee databases *satisfy* the constraint, while the system taken as a whole produces a database instance where social security numbers are not unique.

Now, we want to retain as much information as possible but we would also like to distinguish which answers are consistent with our integrity constraints, and whether a constraint has been violated. Using our epistemic query language $KFOPCE$, we want to be able to ask "Does Jane have a social security number?", to which the system responds with "Yes", and "Do you *know* Jane's social security number?", to which the system answers "No". The system considers possible worlds in which Jane has the social security number 123 and worlds in which she has the number 456, yet there is no perspective from which the system *knows* that either of them are in fact Jane's social security number. This is because the integrity constraint enforces that the system does not consider known any state of affairs in which Jane's social security number is not unique. Any consistent information is therefore known and is not affected by inconsistent

information.

Example 5.0.1 *(Running Example 1)* Formalizing this in our language $KFOPCE$, we have the following scenario:

$$\Sigma = \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\},$$
$$IC := \forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z).$$

$\exists x\ ssn(jane, x)$ formalizes the question asking whether Jane has a social security number without necessarily knowing what that number is. In this case, the answer is "Yes". What if we ask whether there is a *known* social security number, $\exists x Kssn(jane, x)$? In this case, the answer is still "yes" because Jane in fact has two numbers. How do we incorporate the integrity constraint IC? Simply adding IC to Σ will not work because the resulting epistemic state is the inconsistent state in which everything is known. We can attempt to formalize what we *want* the system to answer. In this case, we want $\Sigma \not\models\!\!\approx \exists x Kssn(jane, x)$ but where $\Sigma \models\!\!\approx \exists x\ ssn(jane, x)$. Using Σ, however, will not work. So we could find some Σ' that gives us the behaviour required; this is what we do in Chapter 6. Another avenue is available to us. Since $\Sigma \models\!\!\approx \alpha$ is equivalent to $e_\Sigma \models K\alpha$, we find some epistemic state e_C that does the exact same thing (i.e. $e_C \not\models \exists x Kssn(jane, x) \wedge K\exists x\ ssn(jane, x)$).

In this chapter, we present a perspective of Arenas et al.'s [4] notion of a *consistent query answer* that treats the consistent information as the only facts that are known. Although this is proved in Chapter 6, the intuition is that we do not *remove* any information from the database but isolate the information that is consistent with a set of integrity constraints. The developments of the previous chapters allow us to formalize this notion using our epistemic query language $KFOPCE$. Furthermore, we find that the notion of a *consistent query answer* can be extended to consider sets of *dependencies*, as opposed to simply *full dependencies*, as has been studied by [4]. And unless we apply the *closed-world assumption* on the repairs, *tuple-generating dependency* violations lead to information loss.

The incompleteness illustrated in the example above regarding Jane's social security number is characterized as a set of possible worlds, what we define as the epistemic state e_C, in which consistent answers are those that hold across all

worlds in e_C. This state is constructed from the possible *repairs* of a database which themselves satisfy the integrity constraints.

We not only provide examples illustrating these ideas but go on to show that we can use $\models\!\!\approx$ to query e_C by querying the repairs. This leads us to a Prolog-like sound and complete query evaluator with respect to e_C and admissible formulas, which we call *cqa*. This is an extension of Reiter's [45] query evaluator *demo* for admissible formulas. We conclude this chapter with a correspondence theorem, linking the pioneering notion of *consistent query answers* with the one developed here.

In the remainder of this chapter, Σ is a database instance Σ and IC is a set of *dependencies* in $KFOPCE$.

5.1 Basic Definitions

In the following, when IC is a set of dependencies in $KFOPCE$, $\Sigma \models\!\!\approx IC$ means that $\Sigma \models\!\!\approx \sigma$ for every $\sigma \in IC$. Next, we define a general notion of inconsistent database with respect to any set of allowed constraints IC.

Definition 5.1.1 (Inconsistent Databases) *When Σ is a database and IC a set of $KFOPCE$ allowed constraints, then Σ is* inconsistent *just in case Σ violates some constraint $\sigma \in IC$ (see Definition 4.1.1).*

Example 5.1.2 *(Running Example 1)* Define:

$$\begin{aligned} \Sigma &= \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}, \\ IC &= \{\forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z)\}. \end{aligned}$$

It is clear that $\Sigma \not\models\!\!\approx IC$. Thus, Σ violates integrity constraint σ and is inconsistent.

To reproduce the behaviour described in the introduction, we attempt to *repair* Σ so that we generate instances that satisfy the constraints based on the information in Σ. We first define the *distance* between databases:

Definition 5.1.3 (Distance [4]) *Let r, r' be database instances. Then the* distance *between them is their symmetric difference, denoted $\Delta(r, r')$. That is:*

$$\Delta(r, r') = (r - r') \cup (r' - r).$$

Next we define a partial order $\leq_r$ on the set of database instances with respect to another database instance r.

Definition 5.1.4 ([4]) *For databases instances r, r', r'', we have that:*

$$r \leq_r r' \text{ iff } \Delta(r, r') \subseteq \Delta(r, r'')$$

While this definition of *distance* uses a set-inclusion notion of ordering, others have been used, such as set cardinality [8, 9]. Gertz [24] suggests that the set-inclusion definition of repair is in fact less restrictive than the set-cardinality definition, and goes on to define properties of repair strategies that allow him to order them with respect to "permissibility". This is one reason we choose this particular notion. Also, since we are investigating the semantics of consistent query answers, we use Arenas et al.'s [4] original definition although different notions of distance can be used.

Example 5.1.5 We can see that $\Delta(r, r') = \{P(b), P(c), Q(b)\}$ and $\Delta(r, r'') = \{P(c)\}$. Thus $r'' \leq_r r'$. Let:

$$\begin{aligned} r &= \{P(a), P(b), Q(b), Q(c)\} \\ r' &= \{P(a), P(c), Q(c)\} \\ r'' &= \{P(a), P(b), P(c), Q(b), Q(c)\} \end{aligned}$$

We now define what we mean by a *repair* of a database [4] when IC is a set of $KFOPCE$ dependencies. It is important to note that while IC is a set of admissible formulas, their admissibility is merely a side effect of the particular form they take, which is agreed to account for a majority of constraints that appear in relational and deductive databases [24]. That is, we focus here on *dependencies-qua-constraints* as opposed to *dependencies-qua-admissible formulas.* That they are admissible merely makes it easier for us to use IC_{free} (see Definition 4.2.10) to ask for known instances when finite answers are required.

Definition 5.1.6 (Repair) *We say that r' is a repair of r if $r' \models\!\!\approx IC$ and r' is $\leq_r$-minimal with respect to other r'' such that $r'' \models\!\!\approx IC$. That is, if there is some r_0 such that $r_0 \leq_r r'$, then $r_0 = r'$. Let $repairs(\Sigma, IC)$ be the set of repairs for Σ with respect to IC.*

Remark 5.1.7 Observe that this definition is not equivalent to the definition of *repair* in Arenas et al.'s [4] work since we assume that IC is a set of $KFOPCE$ dependencies. As a result, our notion of constraint satisfaction employs $\approx$ as opposed to $\models_{FOPCE}$. See Section 5.4 below for a discussion on this issue.

We prove the following properties for this notion of repair.

Proposition 5.1.8 *If $\Sigma \approx IC$, for database Σ and a set of $KFOPCE$ dependencies IC, then $\Sigma \leq_\Sigma \Sigma'$ for any other database Σ'.*

Proof: $\Delta(\Sigma, \Sigma) = \emptyset$ and $\Sigma \approx IC$ by assumption. □

Proposition 5.1.9 *There exists a database r such that $r \leq_\Sigma r'$ for any other database r'. That is, a database can always be repaired with respect to a set of $KFOPCE$ dependencies IC.*

Proof: When $\Sigma \approx IC$, then $r = \Sigma$ by Proposition 5.1.8. Otherwise, when $\Sigma \not\approx IC$ assume that there are no repairs of Σ. This means that there are no database instances that could satisfy IC. Let $r = \{\}$. Moreover, $\Delta(r, \Sigma)$ is finite since both r, Σ are databases. Then by Corollary 4.2.9, $r \approx IC$. This is a contradiction.□

Example 5.1.10 *(Running Example 1)*

$$\begin{aligned} \Sigma &= \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}, \\ IC &= \{\forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z)\}. \end{aligned}$$

It is clear that $\Sigma \not\approx IC$. We consider possible repairs for Σ:

$$\begin{aligned} r_1 &= \{ssn(jane, 123), ssn(james, 234)\}, \\ r_2 &= \{ssn(jane, 456), ssn(james, 234)\}, \\ r_3 &= \{ssn(jane, 123), ssn(james, 234), ssn(joe, 789)\}, \\ r_4 &= \{ssn(jane, 123)\}. \end{aligned}$$

Observe that each $r_i \not\approx IC$ but that:

$$\begin{aligned}
\Delta(\Sigma, r_1) &= \{ssn(jane, 456)\}, \\
\Delta(\Sigma, r_2) &= \{ssn(jane, 123)\}, \\
\Delta(\Sigma, r_3) &= \{ssn(jane, 456), ssn(joe, 789)\}, \\
\Delta(\Sigma, r_4) &= \{ssn(jane, 456), ssn(james, 234)\}.
\end{aligned}$$

r_1, r_2 are therefore the only repairs of Σ.

Not only does the notion of repair consider permit *deletions* of facts, but also *insertions* [4, 7], as the following example shows.

Example 5.1.11 *(Running Example 2)* Let Σ, IC be the following:

$$\begin{aligned}
\Sigma &= \{P(a), Q(b), Q(c)\}, \\
IC &= \{\forall x(KP(x) \supset KQ(x))\}.
\end{aligned}$$

Since $\Sigma \not\models IC$, consider the following possible repairs of Σ:

$$\begin{aligned}
r_1 &= \{Q(c)\}, \\
r_2 &= \{P(a), Q(a)\}, \\
r_3 &= \{Q(b), Q(c)\}, \\
r_4 &= \{P(a), Q(a), Q(b), Q(c)\} \\
&\quad then \\
\Delta(\Sigma, r_1) &= \{P(a), Q(b)\}, \\
\Delta(\Sigma, r_2) &= \{Q(a), Q(b), Q(c)\}, \\
\Delta(\Sigma, r_3) &= \{P(a)\}.
\end{aligned}$$

Thus, r_3, r_4 are the only repairs of Σ.

It may be worthwhile to consider constraints with existential quantification. These examples show the presence of anomalies with our notion of a repair satisfying $KFOPCE$ dependencies.

Example 5.1.12 *(Running Example 3a)* Let Σ and IC be the following sets:

$$\begin{aligned}
\Sigma &= \{R(a), Q(b, c), Q(b, d)\}, \\
IC &= \{\forall x(KR(x) \supset K\exists y Q(x, y))\}.
\end{aligned}$$

Notice that IC is not a dependency but it is admissible. We use it here merely as a contrast to the next example. It is clear that $\Sigma \not\models IC$ since there does not exist a y such that $Q(a, y)$ is in our instance. Our set of repairs are the following:

$$\begin{aligned} r_1 &= \{Q(b,c), Q(b,d)\}, \\ r_2 &= \{R(a), Q(b,c), Q(b,d), Q(a,p_1)\}, \\ r_3 &= \{R(a), Q(b,c), Q(b,d), Q(a,p_2)\}, \\ r_3 &= \{R(a), Q(b,c), Q(b,d), Q(a,p_3)\}, \\ &\ldots \end{aligned}$$

Because each of these repairs differ from Σ by a single tuple, we have a countably infinite set of repairs. The idea is that each p_i should be a possible but unknown instance of $Q(a, x)$.

When we have existential quantification outside the scope of a K operator, however, the situation becomes confusing.

Example 5.1.13 *(Running Example 3b)* Consider:

$$\begin{aligned} \Sigma &= \{R(a), Q(b,c), Q(b,d)\}, \\ IC &= \{\forall x(KR(x) \supset \exists y KQ(x,y))\}. \end{aligned}$$

Again $\Sigma \not\models IC$; not only do we need *some* y such that $Q(a, y)$ is in Σ, we need a *known* y. But how are to know what y? Notice that the repairs are exactly the same as those above:

$$\begin{aligned} r_1 &= \{Q(b,c), Q(b,d)\}, \\ r_2 &= \{R(a), Q(b,c), Q(b,d), Q(a,p_1)\}, \\ r_3 &= \{R(a), Q(b,c), Q(b,d), Q(a,p_2)\}, \\ r_3 &= \{R(a), Q(b,c), Q(b,d), Q(a,p_3)\}, \\ &\ldots \end{aligned}$$

In each of these repairs $r_i, i \geq 2$, there *is* a known y such that $Q(a, y)$ holds. But none of these repairs can agree what that y is.

When $\forall x KR(x) \supset \exists y KQ(x, y)$ is violated by database Σ, y can range over

the entire domain since there is no contextual information *linking* the set of values contributing to the violation and the set of values that would lead to its satisfaction. So it seems that integrity constraints with existential quantifiers provide very little definite information for building repairs. Not only that, the distinctions provided by $\exists x K\alpha$ and $K\exists x\alpha$ disappear. When you must insert an atomic sentence in a repair r, then whether you do so to satisfy $\exists x KP(x)$ or $K\exists x P(x)$ makes no difference. Technically, this means that if r is a database and $P(p) \in r$ for some parameter p, we have $r \models\!\!\approx \exists x(KP(x) \wedge x = p)$ iff $r \models\!\!\approx \exists x(P(x) \wedge x = p)$ (see discussion preceding Section 4.2).

This is why dependencies with existential quantifiers are called *embedded*; their satisfaction depends on some possibly infinite domain providing contextual information [52]. One possibility is to extend the semantics of $KFOPCE$ to include *null constants*. However, the ambiguity of the concept of incomplete information becomes apparent: P(a) is unknown not only when there is a *world* w such that $w \not\models P(a)$ but also when the null constant indicates an unknown individual that is a P. We suggest a rigorous analysis of this issue for future research.

Regardless, that we have a potentially infinite set of repairs with $KFOPCE$ dependencies is not much of a problem if our concern is with a principled or formal way of reasoning with inconsistent databases; the analysis is sufficiently general to permit considerations of optimization.

In this section, we looked at *repairing* a database and how this is carried out in the presence of $KFOPCE$ dependencies. Next, we apply these definitions to *consistent query answering*.

5.2 The Epistemic State e_C

Recall from the introduction that we want our system to be able to answer $\Sigma \not\models\!\!\approx \exists x K ssn(jane, x)$ but where $\Sigma \models\!\!\approx \exists x\ ssn(jane, x)$. However, since the former is not true with respect to Σ and since $\Sigma \models\!\!\approx \alpha$ is equivalent to $e_\Sigma \models K\alpha$, we suggest finding some epistemic state e that will give us the knowledge, and answers, that we want. The idea is that we construct an epistemic state e_C from the set of repairs and then query the epistemic state about its knowledge.

There is a further consideration, however. What happens with *possible* answers? How are we to distinguish inserted information from completely absent information? So if we have one state of affairs in which the Blue Jays won the World Series and another state of affairs where the Braves won, then either fact is possibly known. However, since nothing was said about the truth or falsity of the Red Sox winning the Series, is this fact also possible? It seems that we would want to distinguish the Blue Jays and the Braves from other teams such as the Red Sox. Since Σ is a database instance, and since database instances are supposed to satisfy the closed-world assumption, it makes sense to treat absent information as *false*. See Example 5.2.2 below for a formal example regarding the importance of this assumption.

We have the following definition of e_C.

Definition 5.2.1 (Consistent Epistemic State wrt IC) *Let Σ be a database, IC a set of $KFOPCE$ dependencies and $R = repairs(\Sigma, IC)$. Define:*

$$\mathbb{C}[\![\Sigma, IC]\!] = \{\mathcal{M}(Closure(r)) \mid r \in R\}$$

When Σ and IC are clear from the context, we denote this epistemic state as e_C.

Example 5.2.2 *(Running Example 1)* Let:

$$\begin{aligned} \Sigma &= \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}, \\ IC &= \{\forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z)\}. \end{aligned}$$

Our two repairs of Σ and IC are:

$$\begin{aligned} r_1 &= \{ssn(jane, 123), ssn(james, 234)\}, \\ r_2 &= \{ssn(jane, 456), ssn(james, 234)\}. \end{aligned}$$

We therefore have:

$$e_C = \{\mathcal{M}(Closure(r_1)), \mathcal{M}(Closure(r_2))\}$$

Notice that in each of the sets of worlds $\mathcal{M}(Closure(r_i))$, IC is satisfied by definition since $\mathcal{M}(Closure(r_i)) \subseteq \mathcal{M}(r_i)$. With respect to the union of worlds,

however, IC is also satisfied (see Theorem 6.1.3). We have:

$$\begin{aligned} e_C &\models KIC, \\ e_C &\models K\exists x\ ssn(jane, x),\ but \\ e_C &\not\models Kssn(jane, 123) \wedge Kssn(jane, 456),\ and \\ e_C &\not\models K\exists x Kssn(jane, x). \end{aligned}$$

Moreover, $Instances(IC_{free}, e_C) = \{< james, 234 >\}$, where IC is our integrity constraint asking about definite known instances (see Definition 4.2.10). That is, there is one known definite instance of our constraint. Since neither of Jane's social security numbers are known to be hers, we know that a constraint has been violated. This is as it should be, even though $e_C \models KIC$ since integrity constraints should be satisfied at all times. Note that even if the given social security numbers for Jane are unknown, they are nonetheless *possible.*

$$\begin{aligned} e_C &\models \neg K\neg ssn(jane, 123)\ and \\ e_C &\models \neg K\neg ssn(jane, 456)\ but \\ e_C &\models K\neg ssn(jane, 000). \\ e_C &\not\models \neg K\neg ssn(tarzan, 000)\ since \\ e_C &\models K\neg ssn(tarzan, 000). \end{aligned}$$

This last example illustrates the importance of using the $Closure$ of the repairs. Otherwise, we would not be able to distinguish between absent information and possible but unknown information.

We next consider the case when IC consists of a single tuple-generating dependency.

Example 5.2.3 *(Running Example 2)* Let Σ, IC be the following:

$$\begin{aligned} \Sigma &= \{P(a), Q(b), Q(c)\}, \\ IC &= \{\forall x(KP(x) \supset KQ(x))\}. \end{aligned}$$

As noted earlier, there are two repairs of Σ, one in which we insert the missing tuple and the other where we remove the "enabling" tuple $P(a)$:

$$\begin{aligned} r_1 &= \{P(a), Q(a), Q(b), Q(c)\}, \\ r_2 &= \{Q(b), Q(c)\}. \end{aligned}$$

Defining e_C we have:

$$e_C = \{\mathcal{M}(Closure(r_1)), \mathcal{M}(Closure(r_2))\}$$

With this, we can get:

$$\begin{aligned} e_C &\models KIC, \\ e_C &\models KQ(b) \wedge KQ(c), \\ e_C &\models \neg K\neg Q(a) \wedge \neg K\neg P(a), \ but \\ e_C &\not\models KP(a). \end{aligned}$$

Observe that even though $P(a) \in \Sigma$, it becomes unknown after defining e_C. That is, we lose the information that it is a member of Σ. Again, without *Closure* on repairs, $P(a)$ will be possible, but so will other $\alpha \notin \Sigma$. *Closure*, in the case of tuple-generating dependencies, aids in pinpointing information that was in Σ at the outset.

Under this notion of repair, all conflicting tuples will become possible and unknown instances even if it is a member of the original database. However, in the work of Arenas et al. [4], no notion of *Closure* is applied. We have extended this notion by explicitly including our reliance on this closure assumption. We explore this work, and its relation to the work presented here, in Section 5.4, below.

We finally return to Examples 3a and 3b.

Example 5.2.4 *(Running Examples 3a and 3b)* Consider:

$$\begin{aligned} \Sigma &= \{R(a), Q(b,c), Q(b,d)\}, \\ IC &= \{\forall x(KR(x) \supset K\exists y Q(x,y))\}, \\ IC' &= \{\forall x(KR(x) \supset \exists y KQ(x,y))\}. \end{aligned}$$

Recall that for both IC and IC', Σ has exactly the same sets of repairs. Is this really a problem? When $\Sigma \not\models IC$, the point is to isolate the inconsistent information so that the consistent data is not affected. In both cases, $e_C \models K(Q(b,c) \wedge Q(b,d))$. All is not lost, however, since $e_C \models \neg K\neg R(a)$, $e_C \models \neg K\neg Q(a,p)$ for every parameter p and $e_C \not\models \neg K\neg R(d)$.

5.2.1 Consistent Query Answers

We are ready to define *consistent query answers.*

Definition 5.2.5 (Consistent Query Answers) *Let Σ be a database instance, IC a set of $KFOPCE$ dependencies, and q a $KFOPCE$ formula with free variables $\vec{x}$. Let $e_C = \mathbb{C}[\![\Sigma, IC]\!]$. A tuple $\vec{p}$ is a* consistent query answer *to q (wrt Σ and IC) iff $e_C \models Kq^{\vec{x}}_{\vec{p}}$.*

Definition 5.2.6 (Consistent Answers to a Query) *Define*

$$CInstances(\Sigma, IC, q) = Instances(q, \mathbb{C}[\![\Sigma, IC]\!])$$

to be the set of consistent query answers with respect to database Σ, a set of $KFOPCE$ dependencies IC and $KFOPCE$ query q.

We have the following consequence.

Proposition 5.2.7 *When $\Sigma \approx IC$, then $CInstances(\Sigma, IC, \alpha) = Instances(\alpha, \Sigma)$.*

Proof: From the definition of $CInstances(\Sigma, IC, \alpha)$ and Proposition 5.1.8. □

Example 5.2.8 *(Running Example 1)* Let

$$\begin{aligned} \Sigma &= \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}, \\ IC &= \{\forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z)\}. \end{aligned}$$

Consider the following consistent query answers for a particular query.

$$\begin{aligned} CInstances(\Sigma, IC, ssn(jane, x)) &= \emptyset, \\ CInstances(\Sigma, IC, \neg K\neg ssn(jane, x)) &= \{\langle 123\rangle, \langle 456\rangle\}, \\ CInstances(\Sigma, IC, \neg Kssn(jane, x)) &= \{\langle 123\rangle, \langle 456\rangle\}, \\ CInstances(\Sigma, IC, K\neg ssn(jane, x)) &= \{\langle p_1\rangle, \langle p_2\rangle, \ldots\}, \\ &\quad \text{where } p_i \neq 123 \text{ and } p_i \neq 456. \\ CInstances(\Sigma, IC, ssn(x, y)) &= \{\langle james, 234\rangle\}, \\ CInstances(\Sigma, IC, IC_{free}) &= \{\langle james, 234\rangle\}. \end{aligned}$$

Example 5.2.9 *(Running Example 2)* Let Σ and IC be defined as follows:

$$\begin{aligned} \Sigma &= \{P(a), Q(b), Q(c)\}, \\ IC &= \{\forall x(KP(x) \supset KQ(x))\}. \end{aligned}$$

Then the consistent instances are:

$$\begin{aligned} CInstances(\Sigma, IC, Q(x)) &= \{\langle b\rangle, \langle c\rangle\}, \\ CInstances(\Sigma, IC, \neg K\neg Q(x)) &= \{\langle a\rangle, \langle b\rangle, \langle c\rangle\}, \\ CInstances(\Sigma, IC, \neg K\neg R(x)) &= \emptyset, \\ CInstances(\Sigma, IC, IC_{free}) &= \emptyset. \end{aligned}$$

Example 5.2.10 *(Running Examples 3a and 3b)* Consider:

$$\begin{aligned} \Sigma &= \{R(a), Q(b,c), Q(b,d)\}, \\ IC &= \{\forall x(KR(x) \supset K\exists y Q(x,y))\}, \\ IC' &= \{\forall x(KR(x) \supset \exists y KQ(x,y))\}. \end{aligned}$$

Thus, we have:

$$\begin{aligned} CInstances(\Sigma, IC, Q(x,y)) &= \{\langle b,c\rangle, \langle b,d\rangle\}, \\ CInstances(\Sigma, IC, \neg K\neg Q(a,x)) &= \{\langle b\rangle, \langle c\rangle, \langle d\rangle, \langle p_1\rangle, \langle p_2\rangle, \ldots\}, \\ CInstances(\Sigma, IC, \neg KQ(a,x)) &= \{\langle b\rangle, \langle p_1\rangle, \langle p_2\rangle, \ldots\}, \\ CInstances(\Sigma, IC, K\neg Q(a,x)) &= \emptyset, \\ CInstances(\Sigma, IC, \neg K\neg R(x)) &= \{\langle a\rangle\}, \\ CInstances(\Sigma, IC, \neg KR(x)) &= \{\langle b\rangle, \langle c\rangle, \langle d\rangle, \langle p_1\rangle, \langle p_2\rangle, \ldots\}, \\ CInstances(\Sigma, IC, K\neg R(x)) &= \{\langle b\rangle, \langle c\rangle, \langle d\rangle, \langle p_1\rangle, \langle p_2\rangle, \ldots\}, \\ CInstances(\Sigma, IC, IC_{free}) &= \emptyset, \\ CInstances(\Sigma, IC', IC'_{free}) &= \emptyset. \end{aligned}$$

Admissible Queries and e_C

In the examples given above, queries such as $\neg K\neg ssn(jane, x)$ or $\neg K\neg R(x)$ were not admissible because asking about negative objective information is forbidden

by the definition of admissibility. Is there any way for us to recover *possible but unknown* instances from e_C with admissible queries? As noted in Section 3.4.1, we can ask *whether* there exists a possible but unknown instance with $\exists x\ ssn(jane, x)$ or $\exists x R(x)$ but finding possible instances may require "looking" into values from a possibly infinite domain. Nonetheless, using $KFOPCE$ as a query language allows us to distinguish between asking $\exists x K P(x)$ and $K\exists x P(x)$ and to ask about particular unknown instances with $\neg K P(p)$ for some parameter p; these are non-trivial distinctions. But this should not distract us from the goal of providing a formalization of consistent query answers in $KFOPCE$.

Given these disclaimers, in the next section we design a sound and complete query evaluator with respect to e_C and admissible queries.

5.3 e_C and $\not\approx$

In this section we prove that we can use $\not\approx$ to query the set of repairs of Σ.

Theorem 5.3.1 *Whenever IC is a set of $KFOPCE$ dependencies, Σ a database and α an $KFOPCE$ sentence, we have that:*

$$e_C \models K\alpha \text{ iff for all } r \in repairs(\Sigma, IC), Closure(r) \not\approx \alpha.$$

Proof: By Proposition 3.3.8, $e_C \models K\alpha$ iff $\{w\} \models K\alpha$ for all $w \in e$. By definition of e_C and $\not\approx$, $\mathcal{M}(Closure(r)) \models K\alpha$ iff $Closure(r) \not\approx \alpha$ for all repairs r. □

Corollary 5.3.2 *Whenever IC is a set of $KFOPCE$ dependencies, Σ a database and α an AC $FOPCE$ sentence (see Definition 3.4.30), we have:*

$$e_C \models K\alpha \text{ iff } r \not\approx \mathcal{K}(\alpha), r \in repairs(\Sigma, IC)$$

Proof: $Closure(r) \not\approx \alpha$ iff $Closure(r) \models_{FOPCE} \alpha$ by Proposition 3.3.6. Because of Theorem 3.4.29, we have $Closure(r) \models_{FOPCE} \alpha$ iff $r \not\approx \mathcal{K}(\alpha)$. □

Example 5.3.3 *(Running Example 1)* Let:

$$\begin{aligned} \Sigma &= \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}, \\ IC &= \{\forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z)\}. \end{aligned}$$

Our two repairs of Σ and IC are:

$$\begin{aligned} r_1 &= \{ssn(jane, 123), ssn(james, 234)\}, \\ r_2 &= \{ssn(jane, 456), ssn(james, 234)\}. \end{aligned}$$

We therefore have:

$$e_C = \{\mathcal{M}(Closure(r_1)), \mathcal{M}(Closure(r_2))\}$$

If our query is $q := \exists x K ssn(james, x)$ then we want to know whether for all repairs r, $Closure(r) \approx q$. But $Closure(r) \approx q$ holds iff $\mathcal{M}(Closure(r)) \models K\exists x K ssn(james, x)$ iff $\mathcal{M}(Closure(r)) \models K ssn(james, x)|_p^x$ for all repairs r and for some parameter p. For the parameter 234, the query holds for all repairs r.

For $q := \exists x K ssn(jane, x)$, for each parameter p there is some repair r such that $\mathcal{M}(Closure(r)) \not\models ssn(jane, x)|_p^x$. Thus, $Closure(r) \approx q_p^x$ does *not* hold for all repairs r and for each parameter p; Jane does not have a known social security number.

Example 5.3.4 Let Σ, IC be the following:

$$\begin{aligned} \Sigma &= \{P(a), Q(b), Q(c)\}, \\ IC &= \{\forall x(KP(x) \supset KQ(x))\}. \end{aligned}$$

We have:

$$\begin{aligned} r_1 &= \{P(a), Q(a), Q(b), Q(c)\}, \\ r_2 &= \{Q(b), Q(c)\}. \end{aligned}$$

as the two repairs.

If our query is $q := P(x)$, then we are asking for known instances of P. By Theorem 5.3.1, we are asking whether for all repairs r, $Closure(r) \approx q|_p^x$. Since for each parameters p there is a repair r' for which $Closure(r') \not\approx q|_p^x$, then $q|_p^x$ does not hold for all repairs; there are no known instances of P in e_C.

For $q := \neg K \neg P(x)$, we are asking whether there is a possible instance of P. So for all repairs r, we ask $Closure(r) \approx q_p^x$. But this holds iff $\mathcal{M}(Closure(r')) \not\models K\neg P(x)|_p^x$ for some repair r'. Let $r' = r_1$ and we are done.

This means that we can query the *Closures* of Σ's repairs. All we need is a sound and complete $KFOPCE$ evaluator for $\approx$, then we can iterate through $repairs(\Sigma, IC)$ to retrieve consistent answers. This is what we investigate next.

5.3.1 A Query Evaluator for Querying Consistent Answers

From these results, we can use an extension of Reiter's [45] query evaluator *demo* to obtain consistent answers to our admissible queries. This assumes that a set of repairs R has been constructed from database Σ and $KFOPCE$ dependencies IC:

$$\begin{aligned} qrepairs(\alpha, [r \mid []]) &\leftarrow prove(\alpha, r). \\ qrepairs(\alpha, [r_1 \mid R]) &\leftarrow qrepairs(\alpha, [r_1]), qrepairs(\alpha, R). \end{aligned}$$

$$\begin{aligned} cqa(\alpha, R) &\leftarrow firstorder(\alpha), qrepairs(\alpha, R). \\ cqa(\neg\alpha, R) &\leftarrow modal(\alpha), not\ cqa(\alpha, R). \\ cqa(K\alpha, R) &\leftarrow cqa(\alpha, R). \\ cqa(\exists x\alpha, R) &\leftarrow modal(\alpha), cqa(\alpha, R). \\ cqa(\alpha_1 \wedge \alpha_2, R) &\leftarrow modal(\alpha_1 \wedge \alpha_2), cqa(\alpha_1, R), cqa(\alpha_2, R). \end{aligned}$$

qrepairs iterates through the set of repairs R for the first-order query α. Why a *first-order* query? Suppose our original query is $K\alpha$ and we ask e_C. Then $e_C \models K\alpha$ iff for all $w \in e$ we have $w \models \alpha$; *cqa* mimics this behaviour before passing the "stripped" query to *qrepairs*.

firstorder checks whether α is a first-order formula while *modal* checks whether α has any K operators. Notice that *qrepairs* is the only difference between *cqa* and Reiter's [45] *demo*. In order to prove soundness and completeness, therefore, it suffices to show that *qrepairs* terminates.

Like *demo*, we assume that *prove* is a non-standard sound and complete first-order theorem prover that respects the semantics of parameters. So let π be an enumeration of all the parameters $\overrightarrow{p}$ such that $r \models_{FOPCE} \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$. *prove* iterates through π. If it fails, then the second time *prove* is called when *qrepairs* fails, *prove* binds $\overrightarrow{x}$ to the second $\overrightarrow{p}$ in the enumeration, and so on. *prove* fails when π runs out of tuples. When π is infinite, then *qrepairs* and hence *cqa* will

not terminate, so it is important to use *admissible* formulas on *cqa*. Similarly, when the set of repairs $repairs(\Sigma, IC)$ is infinite, then *cqa* will not terminate. If $repairs(\Sigma, IC)$ is finite, then we can guarantee that *cqa* is both sound and complete for admissible queries.

The set of repairs R and a $KFOPCE$ query are inputs for the program. If there are any free variables in the query α, success implies that there is at least one tuple of parameters $\overrightarrow{p}$ such that $e_C \models K\alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$. See Reiter [45] for details on how to retrieve all the answers to a query α.

Soundness of *cqa*

Lemma 5.3.5 *Suppose that α is first-order, that α has free variables $\overrightarrow{x}$, and that the set of repairs $repairs(\Sigma, IC)$ is finite.*

1. *If $qrepairs(\alpha, repairs(\Sigma, IC))$ succeeds, then the variables $\overrightarrow{x}$ are all bound to parameters $\overrightarrow{p}$ such that $r \models\!\approx \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$ for all $r \in repairs(\Sigma, IC)$.*
2. *If $qrepairs(\alpha, repairs(\Sigma, IC))$ finitely fails, then for all parameters $\overrightarrow{p}$, for some $r \in repairs(\Sigma, IC)$ we have $r \not\models\!\approx \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$.*

Proof: When α is first-order, $repairs(\Sigma, IC)$ finite and $qrepairs(\alpha, repairs(\Sigma, IC))$ succeeds, then, by the assumed properties of *prove*, the free variables of α are bound to parameters $\overrightarrow{p}$ such that $r \models_{FOPCE} \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$. This holds iff $r \models\!\approx \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$ by Proposition 3.3.6.

If $qrepairs(\alpha, repairs(\Sigma, IC))$ finitely fails, then by the properties of *prove*, we have $r \not\models_{FOPCE} \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$ for some repair $r \in repairs(\Sigma, IC)$. Again, by Proposition 3.3.6, we have $r \not\models\!\approx \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$ for all parameters $\overrightarrow{p}$. □

Theorem 5.3.6 (Soundness of *cqa*) *Suppose that $KFOPCE$ α is admissible, that α has free variables $\overrightarrow{x}$, and that the set of repairs $repairs(\Sigma, IC)$ is finite.*

1. *If $cqa(\alpha, repairs(\Sigma, IC))$ succeeds, then the variables $\overrightarrow{x}$ are all bound to parameters $\overrightarrow{p}$ and $r \models\!\approx \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$ for all $r \in repairs(\Sigma, IC)$.*
2. *If $cqa(\alpha, repairs(\Sigma, IC))$ finitely fails, then for all parameters $\overrightarrow{p}$, $r \not\models\!\approx \alpha\frac{\overrightarrow{x}}{\overrightarrow{p}}$ for some repair $r \in repairs(\Sigma, IC)$.*

Proof Idea: Lemma 5.3.5 establishes the base case when α is first-order. The inductive cases are similar to Reiter's [45] soundness proof for *demo*, which we omit here and to which we refer the reader. □

Completeness of *cqa*

Lemma 5.3.7 *If α is first-order with free variables $\vec{x}$ and $repairs(\Sigma, IC)$ finite, then $qrepairs(\alpha, repairs(\Sigma, IC))$ terminates.*

Proof: Since $repairs(\Sigma, IC)$ is finite and since *prove* is a sound and complete first-order query evaluator, $qrepairs(\alpha, repairs(\Sigma, IC))$ returns. □

Theorem 5.3.8 (Completeness of *cqa*) *If $KFOPCE$ α is admissible and $repairs(\Sigma, IC)$ finite, then $cqa(\alpha, repairs(\Sigma, IC))$ returns.*

Proof Idea: Lemma 5.3.7 establishes the base case when α is first-order. As with the proof of soundness, we refer the reader to Reiter [45] for the proof of completeness for *demo*, which is equivalent to *cqa* except for the base case. □

Soundness and Completeness With Respect To e_C

Notice that these results do not entail that *cqa* is a sound and complete query evaluator with respect to e_C. This is because of our closure assumptions on the repairs.

Example 5.3.9 *(Running Example 2)* Let Σ, IC be the following:

$$\begin{aligned} \Sigma &= \{P(a), Q(b), Q(c)\}, \\ IC &= \{\forall x(KP(x) \supset KQ(x))\}. \end{aligned}$$

We have:

$$\begin{aligned} r_1 &= \{P(a), Q(a), Q(b), Q(c)\}, \\ r_2 &= \{Q(b), Q(c)\} \end{aligned}$$

as the two repairs.

Notice that $e_C \models \exists x \neg K \neg P(x)$ but that $cqa(\exists x \neg K \neg P(x), repairs(\Sigma, IC))$ is not guaranteed to terminate since $\exists x \neg K \neg P(x)$ is not admissible.

This gives us the following:

Corollary 5.3.10 *$cqa(\alpha, repairs(\Sigma, IC))$ succeeds iff $e_C \models K\alpha$ for admissible formulas α. Also, $cqa(\alpha, repairs(\Sigma, IC))$ finitely fails iff $e_C \models \neg K\alpha$ for admissible formulas α.*

Proof: For admissible formulas α, we have $Closure(r) \approx\!\!\models \alpha$ iff $r \approx\!\!\models \alpha$ by Theorem 3.4.35. So for all repairs r, $r \approx\!\!\models \alpha$ iff $Closure(r) \approx\!\!\models \alpha$. By Theorems 5.3.6 and 5.3.8, we are done. □

This means that our closure constraints on repairs does not lead to any decrease in expressivity with respect to admissible formulas. These results show that we have extended Reiter's [45] *demo* to the domain of querying possibly inconsistent databases.

Example 5.3.11 *(Running Example 1)* Let

$$\begin{aligned} \Sigma &= \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}, \\ IC &= \{\forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z)\}. \end{aligned}$$

Then

$$\begin{aligned} r_1 &= \{ssn(jane, 123), ssn(james, 234)\}, \\ r_2 &= \{ssn(jane, 456), ssn(james, 234)\}. \end{aligned}$$

To evaluate $ssn(james, x) \wedge \neg\exists y\ ssn(tarzan, y)$ we ask $cqa(Kssn(james, x) \wedge \neg\exists y Kssn(tarzan, y), repairs(\Sigma, IC))$. *cqa* terminates with success since there is no y that is Tarzan's social security number. x will also be bound to 234.

5.4 Correspondence Theorem

In this section, we prove that, using e_C, we capture Arenas et al.'s [4] notion of *consistent query answer.*

In their work, however, a *database instance* is represented model-theoretically as a structure mapping attributes to domain elements and relational schemas to relations on the domain [1]. This is not a big issue in itself because the notion of

distance is defined on the atomic formulas that follow from a database instance and constraint satisfaction is defined in terms of this structure. That is, a *first-order* integrity constraint is satisfied by a database iff it is true in that structure. In addition, the unique names assumption is not built-in to these notions but is assumed to hold.

Given these *caveats*, we can nonetheless provide a mapping to $KFOPCE$ since queries will strictly be first-order. If this is the case, then recall Proposition 3.3.6 which states that whenever α is a $FOPCE$ sentence, then $\Sigma \approx \alpha$ iff $\Sigma \models_{FOPCE} \alpha$. The difference between $\models_{FOPCE}$ and the standard entailment relation in standard first-order logic, however, is $FOPCE$'s treatment of parameters [30]. Since we assume the unique names assumption holds, we can use $\models_{FOPCE}$ safely and view database instances, in the sense employed by Arenas et al. [4], as a finite set of non-equality atomic sentences (i.e. databases in our sense).

Definition 5.4.1 ([4]) *For database Σ, a satisfiable set of* first-order full dependencies IC, *and first-order query q with free variables $\vec{x}$, define $\models_c$ as follows:*

$$\Sigma \models_c q\frac{\vec{x}}{\vec{p}}$$

if for every repair r of Σ, we have that $r \models_{FOPCE} q\frac{\vec{x}}{\vec{p}}$.

There are two further complications, however. Arenas et al. [4] consider *first-order* full dependencies, while we consider $KFOPCE$ dependencies. Moreover, their notion of constraint satisfaction is based on entailment: r satisfies IC iff $r \models_{FOPCE} IC$ for first-order IC. In particular, we want to show that for first-order full dependency σ_{fo}, we have that $r \models_{FOPCE} \sigma_{fo}$ implies $r \approx \mathcal{K}(\sigma_{fo})$.

Lemma 5.4.2 *If r is a repair for Σ under $FOPCE$ full dependency σ_{fo}, then it is a repair for Σ for $KFOPCE$ full dependency $\mathcal{K}(\sigma_{fo})$.*

Proof: We have that $r \models_{FOPCE} \sigma_{fo}$ iff $r \approx \sigma_{fo}$ by Proposition 3.3.6. By Proposition 4.2.4, we have that $r \approx \mathcal{K}(\sigma_{fo})$. □

Remember that $r \approx IC$ iff $r \models_{FOPCE} IC$ for first-order IC (see Proposition 3.3.6), so we do not really need the above lemma. Yet we use $\mathcal{K}(\sigma_{fo})$ since we

understand integrity constraints as statements about the knowledge a system has of the world.

The second problem is that Arenas et al. [4] consider sets of *satisfiable* first-order full dependencies whereas we consider $KFOPCE$ full dependencies that are always (trivially) satisfiable (recall Corollary 4.2.9). We must therefore restrict our full dependencies by ensuring their first-order versions σ_{fo} in $\mathcal{K}(\sigma_{fo})$ are satisfiable. But Lemma 5.4.2 guarantees that whenever a set IC of first-order dependencies is satisfied, so is $\mathcal{K}(IC)$ (i.e. $\mathcal{K}$ applied to all dependencies in IC).

Theorem 5.4.3 (Correspondence Theorem) *For database Σ, a set of satisfiable* first-order full dependencies *IC, and $FOPCE$ query q with free variables $\vec{x}$:*

$$\Sigma \models_c q\frac{\vec{x}}{\vec{p}} \textit{ implies } \mathbb{C}[\![\Sigma, \mathcal{K}(IC)]\!] \models Kq\frac{\vec{x}}{\vec{p}}.$$

Proof: Lemma 5.4.2 gives us that any repair r by IC is also a repair by $\mathcal{K}(IC)$.

By Proposition 3.3.6, $r \models_{FOPCE} q\frac{\vec{x}}{\vec{p}}$ iff $r \models\!\!\approx q\frac{\vec{x}}{\vec{p}}$. Since $\mathcal{M}(Closure(r)) \subseteq \mathcal{M}(r)$, any first-order α such that $\mathcal{M}(r) \models K\alpha$ implies $\mathcal{M}(Closure(r)) \models K\alpha$. Therefore, since $r \models\!\!\approx q\frac{\vec{x}}{\vec{p}}$ implies $Closure(r) \models\!\!\approx q\frac{\vec{x}}{\vec{p}}$ for all repairs r. By Theorem 5.3.1, $e_C \models Kq\frac{\vec{x}}{\vec{p}}$. □

The converse of this theorem does not hold because of the closure assumption on the repairs.

Example 5.4.4 *(Running Example 2)* Let Σ, IC be the following:

$$\begin{aligned} \Sigma &= \{P(a), Q(b), Q(c)\}, \\ IC &= \{\forall x(KP(x) \supset KQ(x))\}. \end{aligned}$$

We have:

$$\begin{aligned} r_1 &= \{P(a), Q(a), Q(b), Q(c)\}, \\ r_2 &= \{Q(b), Q(c)\} \end{aligned}$$

as the two repairs.

So $e_C \models \neg K\neg P(a)$ but neither $\Sigma \not\models_c \neg P(a)$ nor $\Sigma \not\models_c P(a)$ hold. Notice also that $e_C \models K\neg P(c)$ but $\Sigma \not\models_c \neg P(c)$ and $\Sigma \not\models_c P(c)$; under $\models_c$, $P(a)$ and $P(c)$ are indistinguishable.

Otherwise, Arenas et al.'s [4] notion of consistent query answering is equivalent to the one presented here with respect to first-order queries and full dependencies.

We furthermore have the following:

Corollary 5.4.5 *Whenever* $repairs(\Sigma, IC)$ *is finite,* $cqa(w, repairs(\Sigma, IC))$ *is a sound and complete* $\models_c$ *query evaluator for all admissible* $FOPCE$ *queries* w *and* $FOPCE$ *full dependencies* IC*:*

$$\Sigma \models_c w \text{ iff } cqa(w, repairs(\Sigma, IC)) \text{ returns with success.}$$

In addition, $cqa(w, repairs(\Sigma, IC))$ *finitely fails iff* $\Sigma \not\models_c w$.

Proof: By Theorems 5.3.6 and 5.3.8 and the definition of $\models_c$, Definition 5.4.1. □

Corollary 5.4.6 *For admissible* $FOPCE$ *queries* w, $e_C \models Kw$ *iff* $\Sigma \models_c w$.

Proof: This result follows by Corollary 5.3.10 for $FOPCE$ admissible formulas, and Corollary 5.4.5. □

We have thus formalized the notion of consistent query answer in our query language $KFOPCE$. By the above theorem, whenever $e_C \not\models K\neg\alpha$ and $e_C \not\models K\alpha$ then $\Sigma \not\models_c \alpha$ and $\Sigma \not\models_c \neg\alpha$. But this is equivalent to $e_C \models \neg K\neg\alpha \wedge \neg K\alpha$, for $FOPCE$ α. Moreover, for admissible $FOPCE$ queries, our notion of consistent query answer is equivalent to that defined by $\models_c$.

5.5 Summary

In answer to the question "What should our knowledge-based system believe if it contains conflicting information?", we suggested that it believe consistent data but still consider possible and unknown those data that are inconsistent with the given set of dependencies. We formalized this suggestion in our query language $KFOPCE$: databases inconsistent with their constraints are *repaired* either by inserting or deleting conflicting tuples. The resulting candidate *repairs* were chosen under a *set-inclusion* notion of preference.

Thus, if α holds in all repairs of a database then it must be consistent with the integrity constraints. If α conflicts with an integrity constraint but is a member of the original database, we must be able to distinguish it from formulas not in the original database. To do this, we took the *Closure* of each repair r; atomic sentences not in any of these repairs are considered false. That is, they are not possibly known. In addition to being able to use dependencies, departing from the full dependencies used in Arenas et al. [4], we enhanced the dexterity of our system in handling such possible and unknown instances. A *consistent query answer* is an instance to a query such that that instantiated query holds in all repairs of a database, with respect to a set of dependencies.

This querying is accomplished using an epistemic state e_C, which is the union of all the models of the closure of our repairs. Throughout, we gave examples containing equality and tuple-generating dependencies as well as an example containing an embedded dependency.

Although admissibility considerations prevented us from asking for a list of possible or unknown instances. We nonetheless formulated a sound and complete query evaluator *cqa* for admissible formulas with respect to e_C. *cqa* was shown to be sound and complete with respect to $\models_c$, for which we provided a correspondence theorem formalizing the semantics of $\models_c$ in $KFOPCE$ and e_C. Reiter's [45] *demo* was thus extended to the domain of querying inconsistent databases.

In the next chapter, we investigate some properties of our epistemic state e_C and show that consistent facts are always known. This *consistent kernel* of Σ suggests how we can completely characterize the known objective sentences in e_C by isolating an infinite objective set of $FOPCE$ sentences, R_{ec}, such that e_C knows α iff $\mathcal{M}(R_{ec})$ knows α also. Thus, we can use $\models\!\!\approx$ to query R_{ec} as well as use *cqa*.

Chapter 6

Analysis of e_C

Do I contradict myself?
Very well then ... I contradict myself;
I am large ... I contain multitudes.

- Walt Whitman, *Leaves of Grass*

What is known in e_C? Do we lose any information from Σ? Do we gain any? Is e_C a consistent epistemic state? What if Σ is consistent with its integrity constraints? Is e_C *representable* by some set of sentences? These questions and their answers occupy us in this chapter.

Not only do we show that e_C satisfies the integrity constraints of Σ, we completely characterize the known $KFOPCE$ sentences in e_C with a set of objective sentences R_{ec}. This gives us an alternative method of querying e_C using $\models\!\!\!\approx$. To do this, we formally analyze how repairs are constructed with respect to a set of $KFOPCE$ dependencies. Although tedious, these investigations allow us to say that e_C knows α iff $\mathcal{M}(R_{ec})$ knows α. This is the main result of this chapter.

6.1 $KFOPCE$ Dependencies and e_C

Before showing that $KFOPCE$ dependencies IC are always known by e_C, it makes sense to ask whether e_C trivially satisfies them by being empty. That e_C is non-empty is important, otherwise there were no repairs of Σ under IC. This

can only happen when IC is unsatisfiable. But by Corollary 4.2.9, $KFOPCE$ dependencies are always satisfiable by $\{\}$. So e_C is built from at least one repair, even if it is the empty set (Proposition 5.1.9).

Proposition 6.1.1 *For database Σ and a set of $KFOPCE$ dependencies IC e_C is non-empty.*

Proof: The only way for $e_C = \emptyset$ is when each of the $\mathcal{M}(Closure(r))$ are empty, since we take their union to construct e_C. By Theorem 3.4.26, there is at least one model of a repair r and at least one repair, by Proposition 5.1.9. □

This proposition will be useful below for proving that we can *represent* e_C using a set of objective sentences.

We next consider what happens when Σ already satisfies IC. Does our definition of e_C alter the way we answer queries? If we assume that Σ is a database where the closed-world assumption applies, then the answer is "No".

Proposition 6.1.2 *Whenever $\Sigma \not\approx IC$ for a set of $KFOPCE$ dependencies IC and database Σ, we have that:*

$$e_C \models K\alpha \text{ iff } Closure(\Sigma) \not\approx \alpha.$$

Proof: By Proposition 5.1.8 and the definition of e_C. □

The problem is what Theorem 3.4.36 says about querying under the closed-world assumption: it is equivalent to first-order entailment. Whenever Σ satisfies IC, then $e_C \models K\alpha$ iff $Closure(\Sigma) \models_{FOPCE} \overline{\alpha}$ where $\overline{\alpha}$ is α with all the K operators removed. $KFOPCE$ thus provides no distinctions as a query language. Since we are interested in characterizing database inconsistencies over a set of possible worlds in e_C, the above is a straightforward result.

We can now show that $KFOPCE$ dependencies are always satisfied in e_C.

Theorem 6.1.3 *Let $e_C = \mathbb{C}[\![\Sigma, IC]\!]$ for database Σ and a set of $KFOPCE$ dependencies IC. Then*

$$e_C \models KIC$$

Proof: Without loss of generality, assume that IC is a single dependency (see Corollary 4.2.9). $e_C \models KIC$ iff for every $w \in e_C$ we have $e_C, w \models IC$. If there is a world $w' \in e_C$ such that $e_C, w' \not\models IC$, then let $w' = \mathcal{M}(Closure(r))$ for some repair r since $\mathcal{M}(Closure(r))$ is singleton (Corollary 3.4.27), $\mathcal{M}(Closure(r)) \in e_C$ by definition of e_C, and that e_C is non-empty by Proposition 6.1.1. In this case, since $\mathcal{M}(Closure(r)) \subseteq \mathcal{M}(r)$ then $w' \in \mathcal{M}(r)$. Therefore, $r \not\models\!\approx IC$ since $r \models\!\approx IC$ iff for every $w \in \mathcal{M}(r)$ we have $\mathcal{M}(r), w \models IC$. Then r is not a repair of Σ, contradiction. □

Isolating consistent information entails looking for answers that apply to all repairs. And since there may be known instances of an integrity constraint in Σ, it makes sense that e_C satisfy IC, which we could not possibly identify if $e_C \not\models KIC$. From this we can conclude that $KFOPCE$ dependencies are known in e_C.

6.2 The Consistent Kernel of Σ

Since e_C is an epistemic state, it reflects the beliefs the system has of the world. How do we characterize such beliefs? Since we defined e_C with the hope of isolating data consistent with a set of integrity constraints, we want to show that no consistent information is lost. That is, in the previous chapter we considered what it meant for an answer to be consistent; here we briefly explore the conditions under which data from Σ is consistent.

So how are we to isolate information from Σ that comes out known in e_C? One way to do this is to *delete* certain atoms in Σ until we get a consistent database. So for $\Sigma = \{P(c), Q(a), Q(b)\}$ and $IC = \forall x KP(x) \supset KQ(x)$ deleting $P(c)$ leads to a consistent instance. $\Sigma \setminus \{P(c)\}$ is known in e_C. However, for $\Sigma = \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}$ and $IC = \forall x, y, z[Kssn(x, y) \wedge Kssn(x, z) \supset Ky = z]$, $\Sigma \setminus \{ssn(jane, 123)\}$ is a consistent database but e_C does not know $ssn(jane, 456)$. This is because deleting $ssn(jane, 123)$ *also* leads to a consistent database. So we must incorporate the information about other

possible repairs of Σ in isolating the consistent information in Σ.[1]

Definition 6.2.1 (Consistent Kernel) *Define the* consistent kernel *of* Σ *to be the set of sentences* C*:*

$$C(\Sigma) = \{\alpha \mid \alpha \in \Sigma \text{ and } \alpha \notin \Delta(r, \Sigma) \text{ for all } r \in repairs(\Sigma, IC)\}$$

for KFOPCE dependencies IC*.*

Example 6.2.2 *(Running Example 1)* Define:

$$\begin{aligned} \Sigma &= \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}, \\ IC &= \{\forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z)\}. \end{aligned}$$

We have the following repairs:

$$\begin{aligned} r_1 &= \{ssn(jane, 123), ssn(james, 234)\}, \\ r_2 &= \{ssn(jane, 456), ssn(james, 234)\}. \end{aligned}$$

Note that for each $\alpha \in \Sigma$, only $ssn(james, 234) \in C(\Sigma)$ because:

$$\begin{aligned} \Delta(\Sigma, r_1) &= \{ssn(jane, 456)\}, \\ \Delta(\Sigma, r_2) &= \{ssn(jane, 123)\}. \end{aligned}$$

Example 6.2.3 *(Running Example 2)* Let Σ, IC be the following:

$$\begin{aligned} \Sigma &= \{P(a), Q(b), Q(c)\}, \\ IC &= \{\forall x(KP(x) \supset KQ(x))\}. \end{aligned}$$

Since $\Sigma \not\models IC$, consider the following repairs of Σ:

$$\begin{aligned} r_1 &= \{Q(b), Q(c)\}, \\ r_2 &= \{P(a), Q(a), Q(b), Q(c)\}. \end{aligned}$$

$$\begin{aligned} \Delta(\Sigma, r_1) &= \{P(a)\}, \\ \Delta(\Sigma, r_2) &= \{Q(a)\}. \end{aligned}$$

We therefore have $C(\Sigma) = \{Q(b), Q(c)\}$.

[1]The following definition of $C(\Sigma)$ is a cumbersome way of saying $\bigcap r$ for $r \in repairs(\Sigma, IC)$ but we leave it as is since the accompanying proofs are easier to understand with the given definition.

Maximality and Uniqueness of $\mathcal{C}(\Sigma)$

That $\mathcal{C}(\Sigma)$ is a *maximal* subset of Σ is shown by the following result.

Theorem 6.2.4 *Suppose that* $\Gamma \subseteq \Sigma$ *for database* Σ, *that* $\Gamma \not\approx IC$ *and that* $e_C \models K\Gamma$, *then* $\Gamma \subseteq \mathcal{C}(\Sigma)$.

Proof: The proof is by contradiction. Assume that $\Gamma \not\subseteq \mathcal{C}(\Sigma)$. Then for some $\alpha \in \Gamma$ we have $\alpha \notin \mathcal{C}(\Sigma)$. Since $\alpha \in \Sigma$, then $\alpha \in \Delta(r, \Sigma)$ for some r. This holds by definition of $\mathcal{C}(\Sigma)$. But since $\alpha \in \Sigma$, then $\alpha \notin r$. Therefore, $r \not\approx \alpha$. But since α is atomic then we have $Closure(r) \not\approx \neg\alpha$ by Lemma 3.4.25. This holds iff $Closure(r) \not\approx \alpha$. By Theorem 5.3.1 , we have $e_C \models \neg K\alpha$, contradiction. □

There are thus no other maximal consistent subsets of Σ that come out known in e_C. In the next two sections, we completely characterize what is known in e_C for any $KFOPCE$ α.

6.3 Characterization of Known Sentences in e_C

With the definition of $\mathcal{C}(\Sigma)$, we can show that each sentence therein comes out known in e_C, and are the *only* atomic sentences known. We need the following lemma.

Lemma 6.3.1 $\alpha \in \mathcal{C}(\Sigma)$ *iff* $\alpha \in r$ *for all* $r \in repairs(\Sigma, IC)$.

Proof: ($\Rightarrow$) Assume that $\alpha \in \mathcal{C}(\Sigma)$. Then we know that $\alpha \in \Sigma$. If $\alpha \notin r$, then $\alpha \in \Delta(r, \Sigma)$; that is, we delete α from Σ to obtain r. This is a contradiction.

($\Leftarrow$) This direction is a bit more involved. We want to say that if α is in every repair r, then it must be in $\mathcal{C}(\Sigma)$. So, let us look at the structure of the repairs closely. There are 4 cases to consider for atomic α:

1. $\alpha \in \Delta(r, \Sigma)$ and $\alpha \in \Sigma$,
2. $\alpha \notin \Delta(r, \Sigma)$ and $\alpha \notin \Sigma$,
3. $\alpha \notin \Delta(r, \Sigma)$ but $\alpha \in \Sigma$,
4. $\alpha \in \Delta(r, \Sigma)$ but $\alpha \notin \Sigma$.

Case 1: If $\alpha \in \Sigma$ and $\alpha \in \Delta(r, \Sigma)$ then obviously $\alpha \notin r$ otherwise $\alpha \notin \Sigma$. Thus, α cannot be a member of *all* r in this case.

Case 2: Similarly, if $\alpha \notin \Delta(r, \Sigma)$ and $\alpha \notin \Sigma$ then $\alpha \notin r$ since if it was in r, then it must either be in $\Delta(r, \Sigma)$ or Σ.

Case 3: When $\alpha \notin \Delta(r, \Sigma)$ but $\alpha \in \Sigma$ then $\alpha \in \mathcal{C}(\Sigma)$ by definition. This is also the only case where $\Sigma \not\approx IC$ gives us the result directly since Σ is the only repair of itself in this case. So $\mathcal{C}(\Sigma) = \Sigma$ by definition of $\mathcal{C}(\Sigma)$ and Proposition 5.1.8.

Case 4: Finally, if $\alpha \in \Delta(r, \Sigma)$ but $\alpha \notin \Sigma$ then α must have been *inserted* into all repairs r. We show that this can never happen when $\Sigma \not\approx IC$. When $\Sigma \not\approx IC$, then this case is impossible by the foregoing remarks in Case 3. In the remainder of this proof, we consider a single dependency IC where its violation forces the insertion of atomic sentences into Σ to form a repair. If these insertions violate other dependencies, then the other cases above apply for whatever repairs remain.

So how is a repair constructed by inserting atomic sentences into Σ to maintain consistency? Let us look at the form of our $KFOPCE$ dependencies.

$$\forall x_1 \ldots x_k \exists y_1 \ldots y_l [(KA_1 \wedge \ldots \wedge KA_p) \supset (KB_1 \wedge \ldots \wedge KB_q)].$$

Lemma 6.3.2 *If $\alpha \in \Delta(r, \Sigma)$ but $\alpha \notin \Sigma$ then there exists a repair r'' such that $\alpha \notin \Delta(r'', \Sigma)$ and $\alpha \notin \Sigma$.*

Proof: The only way for Σ to violate this constraint is when $A_i|^{\vec{x_i}}_{\vec{p_i}} \in \Sigma$ but some $B_j|^{\vec{x_j},\vec{y_j}}_{\vec{p_j},\vec{s_j}} \notin \Sigma$ for parameters $\vec{p_j}, \vec{s_j}$. So, we construct a repair r by *completing* the dependency.[2] That is, let $r = \Sigma \cup \{B_1|^{\vec{x_1},\vec{y_1}}_{\vec{p_1},\vec{s_1}}, \ldots, B_q|^{\vec{x_q},\vec{y_q}}_{\vec{p_q},\vec{s_q}}\}$ for $1 \leq j \leq q$. When IC is an egd, then no *completions* occur since repairs are databases, and databases only contain non-equality atomic sentences.

Is r a repair? That is, is it minimal with respect to $\leq_\Sigma$?[3] Since $B_j \in r$ and $B_j \notin \Sigma$ then $B_j \in \Delta(r, \Sigma)$. Suppose there is another repair r' such that $r' \leq_\Sigma r$. This means that $\Delta(r', \Sigma) \subseteq \Delta(r, \Sigma)$. If they are equal then $r' = r$ so r is a

[2]This terminology is from Gertz [24].

[3]Observe that the following deliberations do not work if we use a *set-cardinality* notion of minimality with respect to $\leq_\Sigma$.

repair. Otherwise, there must be some $\alpha \in \Delta(r, \Sigma)$ such that $\alpha \notin \Delta(r', \Sigma)$. But $\Delta(r, \Sigma) = \{B_k|^{\overrightarrow{x_1},\overrightarrow{y_1}}_{\overrightarrow{p_1},\overrightarrow{s_1}}, \ldots, B_q|^{\overrightarrow{x_q},\overrightarrow{y_q}}_{\overrightarrow{p_q},\overrightarrow{s_q}}\}$ by definition of r, so some $B_m|^{\overrightarrow{x_m},\overrightarrow{y_m}}_{\overrightarrow{p_m},\overrightarrow{s_m}} \notin \Delta(r', \Sigma)$ for $k \leq m \leq q$. Since $A_i|^{\overrightarrow{x_i}}_{\overrightarrow{p_i}} \in r'$ otherwise $\Delta(r', \Sigma) \not\subseteq \Delta(r, \Sigma)$. Therefore $r' \not\models IC$ and is not a repair. So r is a repair.

This establishes that when repairing entails completing the dependency—in this case tgds—then the resulting set is a repair. If we have such a repair then we next prove that there is another repair r'' for which there are no insertions but deletions. This means that there is always another repair where nothing is inserted and so no α that is inserted is *always* inserted. So if α is in all repairs, then Case 3 above is the only case that applies.

When $\Sigma \not\models IC$ in this case (Case 4), then we can construct a repair r'' such that $r'' \not\leq_\Sigma r$ and $r \not\leq_\Sigma r''$. Define r'' as follows: $r'' = \Sigma - A_i|^{\overrightarrow{x_i}}_{\overrightarrow{p_i}}$ for some i. We know that $A_i|^{\overrightarrow{x_i}}_{\overrightarrow{p_i}} \in \Sigma$ by assumption above, so it is clear then that $r'' \not\models IC$ since the antecedent does not hold. Is r'' minimal? By a similar argument to the one above, we can show that it is. Assume that there is some repair r' such that $r' \leq_\Sigma r''$. This means $\Delta(r', \Sigma) \subseteq \Delta(r'', \Sigma)$. But $\Delta(r'', \Sigma) = \{A_i|^{\overrightarrow{x_i}}_{\overrightarrow{p_i}}\}$ by definition. So either $\Delta(r', \Sigma) = \Delta(r'', \Sigma)$, in which case $r' = r''$, or $\Delta(r', \Sigma) = \emptyset$, in which case $r' = \Sigma$ and $r' \not\models IC$, contradiction. So r'' is a repair. This shows that there are at least two repairs r, r'' when some α such that $\alpha \in \Delta(r, \Sigma)$ but $\alpha \notin \Sigma$. This means that α can not be inserted into all repairs, due to the existence of r''. This completes the proof of Lemma 6.3.2. //

We can therefore conclude that if $\alpha \in r$ for all $r \in \mathit{repairs}(\Sigma, IC)$, then Case 3 above is the only case that holds, and $\alpha \in \mathcal{C}(\Sigma)$. □

Theorem 6.3.3 *For atomic α, $\alpha \in \mathcal{C}(\Sigma)$ iff $e_C \models K\alpha$.*

Proof: ($\Rightarrow$) By Lemma 6.3.1, $\alpha \in r$ for every $r \in \mathit{repairs}(\Sigma, IC)$. Therefore, $\mathit{Closure}(r) \not\models \alpha$ since $r \subseteq \mathit{Closure}(r)$ and $\alpha \in r$. By Theorem 5.3.1, α is known.

($\Leftarrow$) By Theorem 5.3.1, $\mathit{Closure}(r) \not\models \alpha$ for all repairs r. Since α is atomic, then $\alpha \in r$ for all r. By Lemma 6.3.1, $\alpha \in \mathcal{C}(\Sigma)$. □

Thus, $\mathcal{C}(\Sigma)$ contains the only known atomic sentences in e_C. Are there other known objective sentences in e_C other than those entailed by $\mathcal{C}(\Sigma)$? It turns out that there are.

Definition 6.3.4 $\mathcal{CL}(\Sigma) = \{\neg\alpha \mid \alpha$ *atomic and* $\neg\alpha \in Closure(r)$ *for every* $r \in repairs(\Sigma, IC)\}$.

Theorem 6.3.5 $\neg\alpha \in \mathcal{CL}(\Sigma)$ *iff* $e_C \models K\neg\alpha$ *for atomic* α.

Proof: ($\Rightarrow$) We have $\mathcal{CL}(\Sigma) \subseteq Closure(r)$, so $\mathcal{CL}(\Sigma) \models\!\!\sim \neg\alpha$ for every r. By Theorem 5.3.1, $\neg\alpha$ is known in e_C.

($\Leftarrow$) By Theorem 5.3.1, $Closure(r) \models\!\!\sim \neg\alpha$ for every r. Therefore, because $Closure(r) \models\!\!\sim \beta$ for every atomic $\beta \in r$, since $r \subseteq Closure(r)$, then $r \not\models\!\!\sim \alpha$ otherwise $Closure(r) \models\!\!\sim \alpha$. Since this is not the case, and by definition of $Closure$, we have $\neg\alpha \in Closure(r)$ for every r. Thus, by definition of $\mathcal{CL}(\Sigma)$, $\neg\alpha \in \mathcal{CL}(\Sigma)$. □

6.3.1 Representation Theorem for e_C

We now have two sets of objective sentences, $\mathcal{C}(\Sigma)$ and $\mathcal{CL}(\Sigma)$ whose properties we investigate before proceeding to the main result of this chapter: a representation theorem for e_C.

Definition 6.3.6 $R_{ec} = \mathcal{C}(\Sigma) \cup \mathcal{CL}(\Sigma)$.

Proposition 6.3.7 $R_{ec} \subseteq Closure(r)$ *for every* $r \in repairs(\Sigma, IC)$.

Proof: That $\mathcal{C}(\Sigma) \subseteq Closure(r)$ is given by Lemma 6.3.1 since $r \subseteq Closure(r)$. That $\mathcal{CL}(\Sigma) \subseteq Closure(r)$ for all r is given by the definition of $\mathcal{CL}(\Sigma)$. □

Proposition 6.3.8 R_{ec} *is satisfiable.*

Proof: By Proposition 6.3.7 and Corollary 3.4.27, which states that $Closure(r)$ is satisfiable. □

Theorem 6.3.9 (Representation Theorem) $e_C \models K\alpha$ *iff* $\mathcal{M}(R_{ec}) \models K\alpha$, *for KFOPCE* α.

Proof: ($\Leftarrow$) When α is atomic, the result follows by Theorem 6.3.3. Assume, then, that $e_C \models K\alpha_1$ and $e_C \models K\alpha_2$ whenever $\mathcal{M}(R_{ec}) \models K\alpha_1$ and $e_C \models K\alpha_2$.

When α is of the form $\alpha_1 \wedge \alpha_2$ then the result follows trivially since $R_{ec} \subseteq$

$Closure(r)$ for all r by Proposition 6.3.7. If α is of the form $\neg\alpha_1$, then since we know that $R_{ec} \subseteq Closure(r)$ for all repairs r, then $Closure(r) \approx \neg\alpha_1$ iff $e_C \models K\neg\alpha_1$ by Theorem 5.3.1. When α is of the form $\exists x\alpha_1$, then $\mathcal{M}(R_{ec}) \models K\exists x\alpha_1$ iff $\mathcal{M}(R_{ec}), w \models \alpha_1|_p^x$ for all $w \in \mathcal{M}(R_{ec})$ and some parameter p. By induction, $e_C \models K\alpha_1|_p^x$ iff $e_C \models K\exists x\alpha_1$ where x is distinct from all the free variables of α_1. When α is of the form $K\alpha_1$, then $\mathcal{M}(R_{ec}) \models KK\alpha_1$ iff $\mathcal{M}(R_{ec}) \models K\alpha_1$ by Theorem 3.2.8 and that $\mathcal{M}(R_{ec})$ is non-empty since it is satisfiable (Proposition 6.3.8). By induction the result follows trivially.

($\Rightarrow$) We prove this direction by contradiction. So assume that $e_c \models K\alpha$ but $\mathcal{M}(R_{ec}) \not\models K\alpha$ for $KFOPCE$ α. This means that $\mathcal{M}(R_{ec}) \models \neg K\alpha$ which means $\mathcal{M}(R_{ec}) \models K\neg K\alpha$ by Theorem 3.2.7 (i.e. using the negative introspection axiom). By ($\Leftarrow$), above, this means that $e_C \models K\neg K\alpha$ which implies $e_C \models \neg K\alpha$ since e_C is non-empty by Proposition 6.1.1 and Corollary 3.2.9 (i.e. accuracy of subjective knowledge). Then we have $e_C \models \neg K\alpha$ iff $e_C \not\models K\alpha$, contradiction. □

Corollary 6.3.10 *$e_C \models K\alpha$ iff $R_{ec} \approx \alpha$.*

Corollary 6.3.11 *$CInstances(\Sigma, IC, \alpha) = Instances(\alpha, R_{ec})$.*

Theorem 6.3.9 allows us to use $\approx$ to query R_{ec} as if we were querying e_C. The only difficulty is that R_{ec} is potentially infinite because of $\mathcal{CL}(\Sigma)$. Furthermore, $KFOPCE$ is not *compact* since every finite subset of $\{\exists x P(x), \neg P(p_1), \neg P(p_2), \ldots\}$ is satisfiable, but the entire set is not [30].

Nonetheless, we have shown that we can capture e_C with a set of *objective* sentences. Whether there is any other way of *finitely representing* e_C is a direction for future research.

Example 6.3.12 *(Running Example 1)* Define:

$$\begin{aligned} \Sigma &= \{ssn(jane, 123), ssn(jane, 456), ssn(james, 234)\}, \\ IC &= \{\forall x, y, z(Kssn(x, y) \wedge Kssn(x, z)) \supset Ky = z)\}. \end{aligned}$$

Therefore, $\mathcal{C}(\Sigma)$ and $\mathcal{CL}(\Sigma)$ are given as:

$$\begin{aligned} \mathcal{C}(\Sigma) &= \{ssn(james, 234)\}, \\ \mathcal{CL}(\Sigma) &= \{\neg ssn(tarzan, 000), \neg ssn(jane, 234), \neg ssn(jane, 000), \ldots\}. \end{aligned}$$

From the definition of $\mathcal{CL}(\Sigma)$ we know that $ssn(jane, 123), ssn(jane, 456) \notin \mathcal{CL}(\Sigma)$. Therefore,

$$R_{ec} = \{ssn(james, 234), \neg ssn(tarzan, 000), \neg ssn(jane, 234), \ldots\}.$$

We have:

$$\begin{aligned}
\mathcal{M}(R_{ec}) &\models KIC, \\
\mathcal{M}(R_{ec}) &\models K\exists x\ ssn(jane, x),\ but \\
\mathcal{M}(R_{ec}) &\not\models Kssn(jane, 123) \wedge Kssn(jane, 456),\ and \\
\mathcal{M}(R_{ec}) &\not\models K\exists x Kssn(jane, x), \\
\mathcal{M}(R_{ec}) &\models \neg K\neg ssn(jane, 123)\ and \\
\mathcal{M}(R_{ec}) &\models \neg K\neg ssn(jane, 456)\ so \\
\mathcal{M}(R_{ec}) &\not\models \neg K\neg ssn(tarzan, 000)\ since \\
\mathcal{M}(R_{ec}) &\models K\neg ssn(tarzan, 000).
\end{aligned}$$

We also know that:

$$Instances(\neg K\neg ssn(jane, x), R_{ec}) = \{\langle 123\rangle, \langle 456\rangle\}.$$

Example 6.3.13 *(Running Example 2)* Let Σ, IC be the following:

$$\begin{aligned}
\Sigma &= \{P(a), Q(b), Q(c)\}, \\
IC &= \{\forall x(KP(x) \supset KQ(x))\}.
\end{aligned}$$

Therefore:

$$\begin{aligned}
\mathcal{C}(\Sigma) &= \{Q(b), Q(c)\}, \\
\mathcal{CL}(\Sigma) &= \{\neg P(b), \neg P(c), \neg Q(d), \neg Q(e), \ldots\}
\end{aligned}$$

$R_{ec} = \{Q(b), Q(c), \neg P(b), \neg P(c), \neg Q(d), \neg Q(e), \ldots\}$. Then:

$$\begin{aligned}
\mathcal{M}(R_{ec}) &\models KIC, \\
\mathcal{M}(R_{ec}) &\models KQ(b) \wedge KQ(c), \\
\mathcal{M}(R_{ec}) &\models \neg K\neg Q(a) \wedge \neg K\neg P(a),\ but \\
\mathcal{M}(R_{ec}) &\not\models KP(a).
\end{aligned}$$

We also know that:

$$\begin{aligned} Instances(Q(x), R_{ec}) &= \{\langle b\rangle, \langle c\rangle\}, \\ Instances(\neg K\neg Q(x), R_{ec}) &= \{\langle a\rangle, \langle b\rangle, \langle c\rangle\}, \\ Instances(\neg K\neg R(x), R_{ec}) &= \emptyset, \\ Instances(IC_{free}, R_{ec}) &= \emptyset. \end{aligned}$$

6.4 Summary

In this chapter, we completely characterized the $KFOPCE$ sentences that are known in e_C by first defining the *consistent kernel* of Σ. To prove that atomic sentences known in e_C came from this consistent kernel, we formally investigated how repairs are constructed from a set of $KFOPCE$ dependencies. We then extended the consistent kernel sentences by constructing R_{ec} which we then proved to completely characterize the known $KFOPCE$ sentences in e_C. In addition to R_{ec}, we showed that the dependencies are known in e_C and that when $\Sigma \not\approx IC$, query answering proceeds under the closed-world assumption, which is equivalent to first-order entailment. Nonetheless, we explored the relationship between the epistemic state e_C, which is a collection of possible worlds, and a symbolic knowledge base R_{ec}, which is a collection of sentences about the world; in the former case, what is known is what is true in all these worlds whereas in the latter case, what is known is what is entailed from the sentences [30].

Chapter 7

Conclusion

If we envision a knowledge-based system obtaining its information from multiple sources—be it from sensors, from other systems, or from humans entering data at a terminal—then it is a possibility that these sources have conflicting descriptions of the world, culminating in a situation where our knowledge-based system has to decide what to believe. Furthermore, the system has no reason to doubt the correctness of the pieces of information it receives, or, alternatively, it has no reason to prefer any particular piece of information. It is not that these sources have different vocabularies that cause the problems that we consider, it is that these sources have different opinions as to what is true—"inconsistency threatens" [11]. This inconsistency arises from the *unified* perspective the system gives to these sources of information. There are myriad practical settings of which this unified view is the synthesis, such as data warehousing, multidatabases and agent communication, but the problems are similar: what does our knowledge-based system choose to believe?

This dilemma frames the techniques employed in this thesis, which looks at integrity constraint violations as a fragment of the incomplete knowledge the system currently has of the world. So we do not want to consider an agent's knowledge inconsistent until it receives more information. This is accomplished by leaving the database in its original state, and modeling this incompleteness using an *epistemic state* in which consistent information is *known*. This idea has roots in Arenas et al.'s [4] notion of a *consistent query answer* as one that holds in all *repairs* of a database. With the epistemic query language $\mathcal{KL}$ and

the subset $KFOPCE$ that we consider, we formalize consistent query answers in our system by treating each of the repairs as a set possible worlds. The logic $KFOPCE$ then allows us to quantify over these worlds using formulas with K operators as queries. The result is a knowledge-level semantics of consistent query answers, which is information that is known across all possible states of affairs, an incompleteness effected by integrity constraint violations.

While other approaches attempt to *clean* the data, the proposed approach is supported by the view that to establish which data is in fact inconsistent, some querying will have to occur [16]. The benefit of this approach is that information loss is minimized since application programmers do not have the necessary domain knowledge to isolate relevant information. Furthermore, we saw that without the closed-world assumption, information that caused constraint violations are indistinguishable from completely absent information. This feature extends the work of Arenas et al. [4] and provides further justification for their approach of consistent query answering. Ultimately, a certain amount of texture is added to the information because decision makers can then distinguish consistent and inconsistent information by querying the knowledge-based system about what it knows.

In the remainder of this chapter, we present highlights of the techniques described above and directions for future research.

7.1 Highlights and Discussion

The Logic $\mathcal{KL}$

The $KFOPCE$ fragment of $\mathcal{KL}$ is used as an *epistemic query language* for knowledge-based systems [30]. That is, we can ask what a knowledge base knows. We appropriate this language for querying databases, and show that it can be adapted to the special circumstances of databases. In particular, by defining and investigating the *admissible queries* of Reiter [45], we ensure that answers to these queries are finite. This is not enough since queries such as $\forall y R(x, y)$ are necessarily finite but depend on the values over which the universal quantifier ranges. This is impractical since the answers to queries depend on information not accessible to the user writing queries. Thus, in addition to being *safe*,

queries need to be *domain independent.* Admissible queries in $KFOPCE$ are indeed domain independent and provide us with an alternative proof of safety. Finally, since databases are assumed to satisfy the *closed-world assumption*, we explore the effect closure constraints have on our query operator $\approx$. The result is that closed-world query evaluation is equivalent to first-order entailment, so we lose the distinctions provided by the K operator. However, we define a class of $FOPCE$ formulas, which we call AC formulas, in order to query the database directly as opposed to its closure, which is a possibly infinite set.

All this assumes that we have a semantics that allows us to distinguish between some fact about the world and the system's knowledge of the world. $KFOPCE$ furnishes us with this semantics where the truth of a formula $K\alpha$ depends only on the *epistemic state*, which is understood to be a set of possible worlds. Objective formulas are evaluated with respect to a "real" world w, which is not necessarily a member of e; the agent's beliefs need not be consistent with what is actually the case. Because of this, $KFOPCE$ is considered a logic of *belief.*

A knowledge-based system specified in $KFOPCE$ also has complete and accurate knowledge of its beliefs, which means that we cannot tell it a fact about itself which it does not already know to be true or know to be false. So even in the state e_0, where it knows nothing of the objective world, aside from objective validities, it completely and accurately knows about what it believes about the world. The difference is that there may be no known instances that it believes to be true.

From the user's perspective, we can ascertain whether the system has definite knowledge about a property with $\exists x K P(x)$ or ask whether the system knows that some individual has a property without necessarily knowing who that individual is, $K\exists x P(x)$. We can also ask about possibly known facts with $\neg K \neg \alpha$ or whether there is a possibly known yet currently unknown individual with some property, $\exists x[P(x) \wedge \neg K P(x)]$. In the end, in addition to being a general knowledge-base query language, these properties and distinctions of $\mathcal{KL}$ enhances its utility as a database query language, even if we restrict ourselves to admissible queries.

Integrity Constraints in $\mathcal{KL}$

Reiter [44] suggests that integrity constraints should be interpreted as statements about the knowledge a knowledge-based system has of the world rather than statements about the world itself. He provides counterexamples against standard notions of constraint satisfaction and suggests that integrity constraints in epistemic logic evades these problems. Inherent in Reiter's examples of integrity constraints in $KFOPCE$ is the assumption that each is satisfied by e_0. The intuition being that if nothing is known that violates the constraints, then the constraints should be satisfied trivially. We make this assumption explicit in our definition of *allowed integrity constraints.* In addition to being satisfiable by e_0, they are subjective sentences without iterated modalities.

While allowed constraints are sufficiently general, they have no obvious syntactic form that we could use for our application to database inconsistencies. Fortunately, database *dependencies* are a sufficiently general class of constraints used in relational database systems. They specify relationships between data already *known* to be true of the database. This makes their translation to $KFOPCE$ more intuitive. We formulate dependencies in $KFOPCE$ and show that such $KFOPCE$ *dependencies* are admissible.

Consistent Query Answers and e_C

In answer to the question "What should our knowledge-based system believe if it receives conflicting data from multiple sources?", we suggest that it believe consistent data but still consider possible and unknown those data that are inconsistent with the given set of dependencies. Formally, we *repair* a database instance either by inserting or deleting facts so that the database satisfies the constraints. Then, we take the union of the *Closures* of these repairs to construct the epistemic state e_C. If α holds in all repairs of a database, then it is considered known in e_C and is consistent with the integrity constraints; it is a *consistent query answer.* Otherwise, facts are considered either possibly but currently unknown or known to be false.

We distinguish these cases by isolating facts that appear in one repair and facts that do not appear in any repair at all. This is accomplished by taking

the *Closures* of each of the repairs of Σ with respect to a set of $KFOPCE$ dependencies IC. Thus, for tuple-generating dependency (tgd) violations, we can "access" data in Σ that lead to the violation of the tgd. We therefore increase the flexibility of our system in handling such facts, departing from Arenas et al.'s [4] notion.

Considerations of admissibility for consistent query answering reveals that asking about possibly known instances are prohibited. Yet this should not distract us from showing that e_C formalizes Arenas et al.'s [4] notion of consistent query answer, which meta-logically quantifies over the set of repairs. Therefore, whenever $e_C \not\models K\neg\alpha$ and $e_C \not\models K\alpha$ then $\Sigma \not\models_c \alpha$ and $\Sigma \not\models_c \neg\alpha$. But this is equivalent to $e_C \models \neg K\neg\alpha \wedge \neg K\alpha$ for $FOPCE$ α. This leads to our query evaluator *cqa* for admissible formulas with respect to e_C and Arenas et al.'s [4] $\models_c$. We thus extend Reiter's [45] *demo* to the task of consistent query answering.

Finally, the analysis of e_C leads to a representation theorem that relates certain properties of repairs to sentences known in e_C using an infinite set of objective sentences R_{ec}. This requires a formal investigation of how repairs are constructed with respect to a set of $KFOPCE$ dependencies and what set of sentences from Σ, and from every repair, come out known in e_C. Whenever $\Sigma \not\approx IC$, we show that e_C preserves the closed-world query answers on Σ with respect to $\not\approx$. And to ensure that we constructed e_C properly, we prove that IC is known, although the *known instances* of IC possibly differs from the information in Σ. In the examples presented in Chapter 5, while Jane has two social security numbers in the original database Σ, neither of these numbers are known to be hers because of the integrity constraint that social security numbers are unique. James, on the other hand, has a unique number, and this information is therefore known in e_C. Such information is formalized as the *consistent kernel* of Σ. Information from Σ is therefore retained since we simply treat consistent information as the only known facts.

7.2 Suggestions for Further Research

Finite Representability of e_C

What we have covered of $\mathcal{KL}$ as an epistemic query language is horribly incomplete; $KFOPCE$ is essentially $\mathcal{KL}$ with no function symbols. Levesque and Lakemeyer's [30] work is a full examination, development and extension of $\mathcal{KL}$. It is proved sound and complete, and the two operations **ASK** and **TELL** are defined. Our $\approx$ is equivalent to **ASK** with the exception that in the definition provided by Levesque and Lakemeyer [30], the models of Σ appear as a parameter. **ASK** is therefore defined over the models of a set of formulas but also over epistemic states that may or may not have a set of representative sentences. Chapter 6 shows that R_{ec} is such a set, even if it is infinite. One avenue of research is to find a *finite* representative set of sentences for e_C.

Work in incomplete databases may provide insights into this problem [1, 2, 33, 35]. In [2], Abiteboul et al. discuss various representations of sets of possible worlds as *conditional tables* that include variables as values of particular tuples. There is an associated set of *global* and *local* conditions that impose restrictions on these variables. As such, the permutation of all values the variables can take that satisfy the conditions represent all the possible scenarios described by this incomplete database. It may be interesting to investigate whether e_C has a representation in such a model.

Updates to e_C

What we have considered thus far is a system that is *static*; no additions to e_C are considered. How are we to incorporate changes to e_C such that it maintains its consistency with respect to a set of constraints? Levesque and Lakemeyer [30] provide an update operation on epistemic states called **TELL** defined as follows:

Definition 7.2.1 ([30]) $\mathbf{TELL}[\alpha, e] = e \cap \{w \mid w \models \alpha\}$.

If we use this definition of **TELL**, then $\mathbf{TELL}[\neg\sigma, e_C] = \emptyset$ where σ is any $KFOPCE$ dependency. This happens because e_C has complete knowledge of subjective sentences, since $e_C \models K\sigma$ by Theorem 6.1.3. Telling it that it does

not have complete knowledge of a subjective sentence results in an inconsistent epistemic state. Thus, a problem is to define a new **TELL** operation, **TELLc**, that allows the system to evolve with the introduction of new facts or integrity constraints.

De Amo et al. [19] consider a repair operation in a paraconsistent setting that allows their system to change with the introduction of new information, while at the same time maintaining consistency. Since their notion of a *repair* is close to the one we have defined, it may be worthwhile to consider their algorithm for updates to extract a semantic specification that we could adopt for **TELLc**. Furthermore, the work of Alchourrón et al. [3] is especially relevant since it provides a principled foundation for *belief revision.* But specific connections with the approach of this thesis have yet to be established although recent work on knowledge-base merging based on belief revision by Delgrande and Schaub [20] suggests connections with database inconsistencies with respect to integrity constraints.

Meyer [36] extends the work of Spohn [48] and considers merging operations on *epistemic states* as opposed to *knowledge bases* of which work in belief revision , so **TELLc** could implement some version of his approach using either an arbitration operation, which tries to accommodate different sources' opinions, or a majority operation that simply takes the opinion of the majority. This technique addresses a problem that seems to be related to the one with which we originally began, but in this thesis we considered *constructing* an epistemic state by simply taking the union of the models of repairs as opposed to performing complex operations on them. If we view $\mathcal{M}(r)$ for repair r as an epistemic state, then the proposed operations on epistemic states could be applied. We suggest this as a topic for further investigation.

A Wider Class of *Admissible* Formulas

Reiter's [45] Prolog-like query evaluator *demo* for $KFOPCE$ has furnished our definitions of admissible queries. This definition stems from Reiter's adoption of the Lloyd-Topor [27] transformation rules for implementing definitional theories in Prolog. The resulting first-order formula has the form of normal Prolog goals, and successful termination of any pure Prolog interpreter ensures that the answer

is true with respect to the definitional theory, due to Clark's Theorem [46].

Furthermore, Reiter's definition was tailored for evaluation of $KFOPCE$ queries by a Prolog interpreter. A side effect of this is that admissible queries are also "safe for negation". Since, it is assumed, implementation of this evaluator is straightforward once a first-order theorem prover is found that respects the semantics of parameters, then it will no doubt inherit Prolog's sensitivity to unbound variables on negated subgoals. It would be interesting to find a superset of the $KFOPCE$ admissible queries whose instances are finite, but are not required to be "safe for negation".

Constraint Violations in $KFOPCE$

The only known extension of Reiter's [44] work on integrity constraints is that of Demolombe and Jones [21] which clarifies the epistemic status of integrity constraints by refining what it means for a constraint to be violated in light of the understanding of integrity constraints put forward by Motro [37]. In this work, Demolombe and Jones provide a formalization of a constraint violation with respect to *validity* and *completeness*, where the violation of the former involves an unsatisfiable instance (i.e. false sentences have to removed) while the latter involves a failure of entailment (i.e. sentences have to be inserted). The formal system they develop is therefore able to check validity and/or completeness of the database's beliefs. Possible connections to this thesis have yet to be explored, but it is suspected that this distinction of constraint violation can lead to a more elegant treatment of the violation of dependencies than that presented in this thesis.

Study of Dependencies in $KFOPCE$

When we considered allowed $KFOPCE$ constraints, we overlooked complex sets of constraints that are *only* satisfied by e_0. For instance:

Example 7.2.2 Let

$$\begin{aligned} \Sigma &= \{R(a), R(b), R(c), Q(a), Q(b)\}, \\ IC &= \{\forall x(KR(x) \supset KQ(x)), \forall x(KQ(x) \supset KR(x)), \forall x(KR(x) \supset \neg KQ(x))\} \end{aligned}$$

The problem is that IC *is* satisfiable by $\{\}$ but it is *only* satisfiable by $\{\}$. Thus, the only repair of Σ is $r = \{\}$, in which case *all* the information is lost. This is surely something we do not want. We avoided this problem by focussing on dependencies. Whether dependencies are non-trivially satisfied is discussed in [52], but an extension of this analysis to $KFOPCE$ dependencies would reinforce $KFOPCE$'s applicability as a logic for reasoning about integrity constraints.

Construction of Repairs and e_C

Further analysis needs to be conducted on the notion of *repair* that we employ for $KFOPCE$ dependencies and constraints. The model-based characterizations of paraconsistent-based approaches, such as those of Arieli et al. [8, 9], Baeza [10] and De Amo [19], are attractive because the repairs can simply be "read" from the multiple-valued models under some notion of preference. Their resulting implementations and algorithms therefore characterize the conditions under which repairs are constructed. We have no equivalent formulation or investigation here.

Gertz [24] provides methods for generating repairs stemming from work on model-based diagnosis. Apart from the specific algorithms used, Gertz defines properties of repair strategies that imposes an ordering with respect to "permissibility". Gertz models the scenario in which a repair is required with a *violating transaction* T, which is a set of insertions or deletions of facts. A *repair transaction* T', or *undo* of T is a transaction in which for every insertion (deletion) in T' of a fact there exists a deletion (insertion) of that same fact in T. Other notions of repair are defined, such as consistent completion, and cardinality-based versions of these. In any case, the point of pursuing repairs more formally is to characterize the knowledge in e_C more elegantly than presented in Chapter 6.

Query Evaluators for Consistent Query Answers

The work of Arenas et al. [4] uses the same semantics of repairs as we have, with respect to first-order dependencies, but provides an effective method for computing consistent query answers without generating any repair. This is expedited with an operator T_ω which rewrites a first-order query and returns the consistent

answers *as if* it had been queried on all the repairs of the database. That is, it is complete with respect to the repair semantics, with some constraints on the form of the integrity constraints.

Although our approach does not need such restrictions on constraints, an interesting proposal is to use the query evaluator that Levesque [29] provides for $\mathcal{KL}$ coupled with T_ω to perform consistent query answering without generating any repairs at all. This would extend T_ω to an epistemic query langauge. In particular, by using the operators $||\cdot||_\Sigma$ and RES of Levesque [29], we can eliminate the K operators from the query and answer it using first-order theorem proving techniques. While this is inefficient [29, 45] and further analysis is required regarding complexity issues of consistent query answering, it suggests an intriguing alternative and fusion of approaches.

Similarly embedding *cqa* in the framework of *inductive logic programming* would allow us to use the system developed by Arieli et al. [9] that generates repairs of a possibly inconsistent database.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley Publishing Company, 1995.

[2] Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. On the representation and querying of sets of possible worlds. In *Proceedings ACM SIGMOD International Conference on Management of Data, San Francisco, California*, pages 34–48, 1987.

[3] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:510–530, 1985.

[4] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.

[5] Ofer Arieli and Arnon Avron. A bilattice-based approach to recover consistent data from inconsistent knowledge-bases. In *Proc. 4th Bar-Ilan Symp. on Foundations of Artificial Intelligence (BISFAI'95)*, pages 14–23. AAAI Press, 1995.

[6] Ofer Arieli and Arnon Avron. The value of four values. *Artificial Intelligence*, 102(1):97–141, 1998.

[7] Ofer Arieli and Arnon Avron. A model-theoretic approach for recovering consistent data from inconsistent knowledge bases. *Journal of Automated Reasoning*, 22(2):263–309, 1999.

[8] Ofer Arieli, Marc Denecker, Bert Van Nuffelen, and Maurice Bruynooghe. Repairing inconsistent databases: A model-theoretic approach and abductive reasoning. In H. Decker and T. Waragai, editors, *Proceedings ICLP'02 Workshop on Paraconsistent Computational Logic*, pages 51–65, 2002.

[9] Ofer Arieli, Marc Denecker, Bert Van Nuffelen, and Maurice Bruynooghe. Coherent integration of databases by abductive logic programming. *Journal of Artificial Intelligence Research*, 21:245–286, 2004.

[10] Pablo Barceló Baeza. Applications of annoted predicated calculus and logic programs to querying inconsistent databases. Master's thesis, Pontificia

Unversid Catolica de Chile, Escuela de Ingenieria, Departamento de Ciencia de la Computacion, 2002.

[11] Nuel D. Belnap. How a computer should think. In Gilbert Ryle, editor, *Contemporary Aspects of Philosophy*, pages 30–56. Oriel Press, 1976.

[12] Nuel D. Belnap. A useful four-valued logic. In J.M.Dunn and G.Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 5–37. D.Reidel Publishing Company, 1977.

[13] S. Benferhat, D. Dubois, and H. Prade. How to infer from inconsistent beliefs without revising? In *Proceedings of the Foureenth International Joint Conferences on Artificial Intelligence*, pages 1449–1455, 1995.

[14] Leopoldo Bertossi, Jan Chomicki, Alvaro Cortes, and Claudio Gutierrez. Consistent answers from integrated data sources. In T. Andreasen, H. Christiansen A. Motro, and H. L. Larsen, editors, *'Flexible Query Answering Systems', Proc. of the 5th International Conference, FQAS 2002.*, pages 71–85. Springer LNAI 2522, 2002.

[15] M.W. Bright, A.R. Hurson, and Simin H. Pakzad. A taxonomy and current issues in multidatabase systems. *IEEE Computer Magazine*, March 1992.

[16] Francois Bry. Query answering in information systems with integrity constraints. In *IICIS*, pages 113–130, 1997.

[17] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[18] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, March 1997.

[19] Sandra de Amo, Walter Alexandre Carnielli, and Joao Marcos. A logical framework for integrating inconsistent information in multiple databases. In *Foundations of Information and Knowledge Systems*, pages 67–84, 2002.

[20] James P. Delgrande and Torsten Schaub. Consistency-based approaches to merging knowledge bases: Preliminary report. In *Proceedings of the* 10^{th} *International Workshop on Non-Monotonic Reasoning*, pages 126–133, Whistler, BC, Canada, June 2004.

[21] Robert Demolombe and Andrew Jones. Integrity constraints revisted. *Journal of the International Interest Group in Pure and Applied Logics (IGPL)*, 4(3):369–383, 1996.

[22] Phan Minh Dung. Integrating data from possibly inconsistent databases. In *Conference on Cooperative Information Systems*, pages 58–65, 1996.

[23] Ronald Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

[24] Michael Gertz. An extensible framework for repairing constraint violations. *Workshop on Foundations of Models and Languages for Data and Objects*, pages 41–56, 1996.

[25] Larry Greenfield. An (informal) taxonomy of data warehouse data errors, http://dwinfocenter.org, 2004.

[26] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[27] J.W.Lloyd and R.W.Topor. Making Prolog more expressive. *Journal of Logic Programming*, 3:225–240, 1984.

[28] Robert Kowalski. Logic for data description. *Logic and Data Bases*, pages 77–103, 1977.

[29] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23:155–212, 1984.

[30] Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases.* The MIT Press, 2000.

[31] Hector J. Levesque and Gerhard Lakemeyer. Situations, si! situation terms, no! In Didier Dubois, Christopher Welty, and Mary-Anne Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference*, pages 516–526, Whistler, BC, Canada, June 2004.

[32] Jinxin Lin and Alberto O. Mendelzon. Merging databases under constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1998.

[33] Witold Lipski. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4(3):262–296, 1979.

[34] Efrem G. Mallach. *Decision Support and Data Warehouse Systems.* Irwin McGraw-Hill, 2000.

[35] Ron van der Meyden. Logical approaches to incomplete information: A survey. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publishers, 1998.

[36] Thomas Meyer. Merging epistemic states. In *Pacific Rim International Conference on Artificial Intelligence*, pages 286–296, 2000.

[37] Amihai Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, 1989.

[38] Amihai Motro. A formal framework for integrating inconsistent answers from multiple information sources. Technical Report ISSE-TR-93-106, Department of Information & Software Systems Engineering, George Mason University, 1993.

[39] Allen Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[40] J.M. Nicolas and H. Gallaire. Data base: Theory vs. interpretation. *Logic and Data Bases*, pages 33–54, 1977.

[41] Guilin Qi, Weiru Liu, and David H. Glass. A split-combination method for merging inconsistent possibilitistic knowledge bases. In Didier Dubois, Christopher Welty, and Mary-Anne Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference*, pages 348–356, Whistler, BC, Canada, June 2004.

[42] Erhard Rahm and Hon Hai Do. Data cleaning: Problems and current approaches. *IEEE Bulletin of the Technical Committee on Data Engineering*, 23(4), December 2000.

[43] Raymond Reiter. Towards a logical reconstruction of relational database theory. *On Conceptual Modelling*, pages 191–233, 1984.

[44] Raymond Reiter. On integrity constraints. *Proceedings of the Second Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 97–111, 1988.

[45] Raymond Reiter. What should a database know? *Journal of Logic Programming*, 14(2):127–153, 1992.

[46] Raymond Reiter. *Knowledge In Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, 2001.

[47] Raymond Reiter. On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic*, pages 433–457, 2001.

[48] Wolfgang Spohn. Ordinal conditional functions: A dynamic theory of epistemic states. In William L. Harper and Brian Skyrms, editors, *Causation in Decision, Belief Change, and Statistics, II*, pages 105–134. Kluwer Academic Publishers, 1988.

[49] Michael Stonebraker. Integrating islands of information. *EAI Journal*, pages 1–5, September/October 1999.

[50] R.W. Topor and E.A. Sonenberg. On domain indepdent databases. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 217–240. Morgan Kaufmann Publishers, Inc., 1988.

[51] Jeffrey D. Ullman. *Principles of Database and Knowledge Base Systems.* Computer Science Press, 1988.

[52] Moshe Y. Vardi. Fundamentals of dependency theory. In Egon Börger, editor, *Trends in Theoretical Computer Science*, pages 171–224. Computer Science Press, 1988.

[53] Ke Wang and W. Zhang. Detecting data inconsistency for multidatabases. *The Ninth International Conference on Parallel and Distributed Computing Systems (PDCS'96), Vol II, Dijon, France*, pages 657–663, September 1996.