# Preprocessing for Controlled Query Evaluation in Complete First-Order Databases

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Lena Wiese

Dortmund

2009

technische universität
dortmund

Tag der mündlichen Prüfung: 19.08.2009

Dekan: Prof. Dr. Peter Buchholz

1. Gutachter: Prof. Dr. Joachim Biskup
2. Gutachterin: Prof. Dr. Gabriele Kern-Isberner

Under the rose (sub rosa). In strict confidence. Cupid gave Harpocrates (the god of silence) a rose, to bribe him not to betray the amours of Venus. Hence the flower became the emblem of silence. It was for this reason sculptured on the ceilings of banquet-rooms, to remind the guests that what was spoken sub vino was not to be uttered sub divo.
– *E. Cobham Brewer, Dictionary of Phrase and Fable*

## Abstract

This dissertation investigates a mechanism for confidentiality preservation in first-order logic databases. The logical basis is given by the inference control framework of Controlled Query Evaluation (CQE). Beyond traditional access control, CQE incorporates an explicit representation of a user's knowledge and his ability to reason with information; it hence prevents disclosure of confidential information that would occur due to inferences drawn by the user.

This thesis pioneers a new approach in the CQE context: An unprotected database instance is transformed into an "inference-proof" instance that does not reveal confidential information; the inference-proof instance formally guarantees confidentiality with respect to a representation of user knowledge and a specification of confidential information. Hence, inference-proofness ensures that all user queries can truthfully be answered by the database; no sequence of responses enables the user to infer confidential information. Due to this concept, query evaluation on the inference-proof instance does not incur any performance degradation. As a second design goal, the availability requirement to maintain as much as possible of the correct information in the input database is accounted for by minimization of a distortion distance.

The transformation modifies the input instance to provide the user with a consistent view of the data. The algorithm relies on query evaluation on the database to efficiently identify those tuples that are to be added or deleted. Due to undecidability of the general first-order case, appropriate fragments are analyzed. The formalization is started with universal formulas (for which a restriction to "allowed" formulas is chosen); it moves on to existential formulas and then finishes up with tuple-generating dependencies accompanied by existential and denial formulas. The due proofs of refutation soundness engage a version of Herbrand's theorem with semantic trees.

An effort was made to present a broad background of related work. Last but not least, exposition and analysis of a prototypical implementation prove practicality of the approach.

## Overview

# I. Introduction and Related Work

## Contents



Prefer the single word to the circumlocution.
– *H.W. Fowler, The King's English*

# 1 Inference Control in Databases

Inference in its generality can be seen as a cognitive reasoning process that – starting from particular premises – leads to particular conclusions. Simon Blackburn defines inference in "The Oxford Dictionary of Philosophy" as:

> "The process of moving from (possibly provisional) acceptance of some propositions, to acceptance of others." [Bla96]

While human thought is far too complex to capture its internal workings and represent it by computational processes, mathematical logic has been the foundation of a computational notion of inference and several automated reasoning mechanisms. From a logical point of view, inference can be subdivided into three categories, namely deduction (infer facts from axioms by application of rules), induction (infer general propositions from example instances) and abduction (infer justifications for observed behavior). Induction and abduction both involve some kind of uncertainty in terms of validity of the inferred facts (which makes both of them unsound), whereas deduction has the nice property of giving a well-defined set of conclusions (see also [Bla96]). In this work, inference is seen as a deductive process, namely based on implications in first-order logic.

On top of that, logic also comes into play when talking about the theoretical foundations of databases. Roughly speaking, databases represent information as data tuples which again can be seen as logical facts. Moreover, database constraints (often codified as logical formulas) restrict the set of possible database states to "admissible" or "semantically meaningful" ones that obey these constraints; that is, data tuples are meant to form a database state that is "consistent" with the constraints.
The main purpose of databases is to store data in a way that makes the contained information easily accessible to database users. Specialized query languages facilitate data access for its users. Although a short introduction to database theory is given in this thesis in the appropriate sections, we will not dwell on all the details. [AHV95] is a good starting point for further studies.

In computerized systems where information flow from information sources to information recipients takes place, some kind of information may be confidential in the sense that this confidential information is not meant to be accessed by some of the recipients. Not surprisingly, this is also true when an information source is a database system and an information recipient is a database user. In the database system, information has to be administered in terms of data in the database; confidentiality requirements have to be specified in terms of these data, too. In this abstract setting, we assume that there is on the one hand a database administrator *dbadm* whose task is to maintain the data in the database independent of their confidentiality requirements. On the other hand, a security administrator *secadm*

is charged with devising an appropriate policy to control access of users to confidential information.
In the "traditional" access control (AC) setting, the policy has to be mapped to access rights for certain users on certain data items – either done by hand by the security administrator or with automated support. This general AC setting is illustrated in Figure 1 where the user sends a query sequence $Q$ and retrieves a sequence $A$ of query responses based on "ordinary query evaluation" (denominated $eval^*$) which will be formalized shortly; some responses may be denied by the database as indicated by the access control policy and the ensuing access rights.

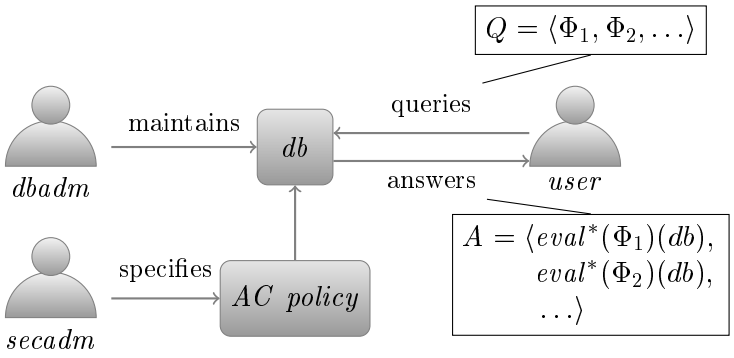


Figure 1: Schematic view of uncontrolled queries and access control

Plain access control on specific data items is in most cases not sufficient to prevent disclosure of confidential information: a user may dispose of additional background knowledge beyond the information revealed purely in the direct information flow. The user's background knowledge might for example consist of:

- common knowledge (information that can be assumed to be generally known)
- domain knowledge (information specific to the topic of interest)
- content knowledge (information about the data stored in the system; for example data constraints that have to be enforced in the system)
- external knowledge (information retrieved from outside of the system)
- system knowledge (information about the functionality of the system; for example internal workings of query evaluation functions or access control mechanisms)
- meta-knowledge (knowledge about knowledge; the user might suspect what the security administrator supposes him to know)

The user can employ his knowledge (combined with the information returned directly by the system) to derive confidential information. Figure 2 depicts this situation for a database instance $db$ that contains some medical facts about a patient

called Mary. The AC policy denies access to the information that a person suffers from aids, hence the response to the user's first query asking for all patients with aids (that is, $Ill(x, \mathsf{Aids})$) is denied. Yet, the second response (all patients being treated with some medicine $A$, that is, $Treat(\mathsf{Mary}, \mathsf{MedA})$) is truthfully returned. Now we assume that the user possesses a set of knowledge called *prior* (because he had this knowledge *a priori* before he started querying *db*) that tells him that every patient taking medicine $A$ is definitely ill with aids. Combining this and the truthful response enables him to derive (with the help of "DB-implication" $\models_{DB}$ which will be defined later on) the confidential information.
Consequentially, in the AC situation if the *secadm* and his AC policy take too

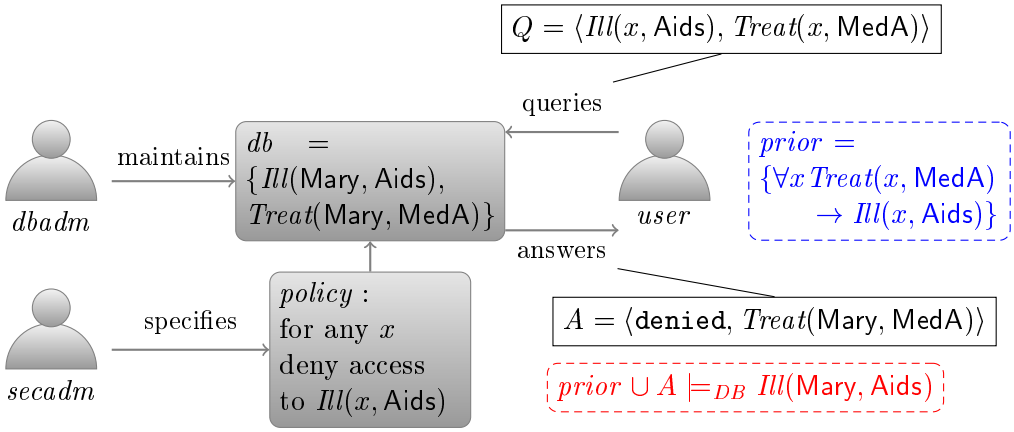


Figure 2: Insecurity of access control

narrow a view of the user knowledge, "harmful" inferences revealing confidential information can come up. The other way round, if the *secadm* gets paranoid, too strict an AC policy may unnecessarily deny access to information that does not lead to such inferences. The research area of **inference control** aims at the design and analysis of automated mechanisms that prevent such harmful inferences while still allowing for maximum availability of non-sensitive information.

As large amounts of data are stored in databases, inference control in databases is of paramount interest. It has long since been investigated in the context of statistical databases; some more details on this area are given below. In general-purpose databases (that is, databases not just released for statistical evaluations), inferences are mostly established based on a specified set of database constraints. This includes approaches for relational and multilevel secure databases; some of these approaches are presented below. Recently, inference control in semi-structured data (for instance, in XML documents) has come up as a new research area. See [YL04, SF04, FBJ06] for some approaches for inference control in tree-based representations of data.

Inference control in databases has been approached with different means which have not been systematically compared yet. It is the author's contention that a categorization of the different parameters of inference control is necessary. We propose the following categorization:

1. Data Model: describes what kind of database is handled
2. Constraint Model: describes the types of constraints on data that should hold in the database
3. User Model: describes assumptions about the user made by the system
4. Inference Model: describes what kind of inferences the user can draw
5. Policy Model: describes the way confidentiality (and sometimes availability) requirements are specified
6. Interaction Model: describes how the user interacts with the system
7. Protection Model: describes by which means confidentiality (and availability) is enforced
8. Execution Model: describes the modus operandi of the inference control mechanism

A schematic view of the categories and the taxonomy of inference control is given in Figure 3. In the following we present related work in the inference control area but structure it based on the above categories. In Section 3 we describe and discuss some of the most prominent approaches to inference control since the 1980s a bit more in detail.

## 2 Related Work

In prior publications on the topic of inference control in databases, different notions of databases and different forms of confidentiality policies are employed; there exists also a great variety of execution techniques.
In this section we want to present some of the existing approaches structured by the categories as defined in the previous section, in order to facilitate a comparison between the disparate existing approaches. We also develop a homogeneous terminology for all aspects surrounding inference control.

### 2.1 Data Model

As previously mentioned, historically, inference control mechanisms differ strongly in the case that a database is solely released for the purpose of statistical evaluation of the data (the "statistical databases"), and the case that the database is meant

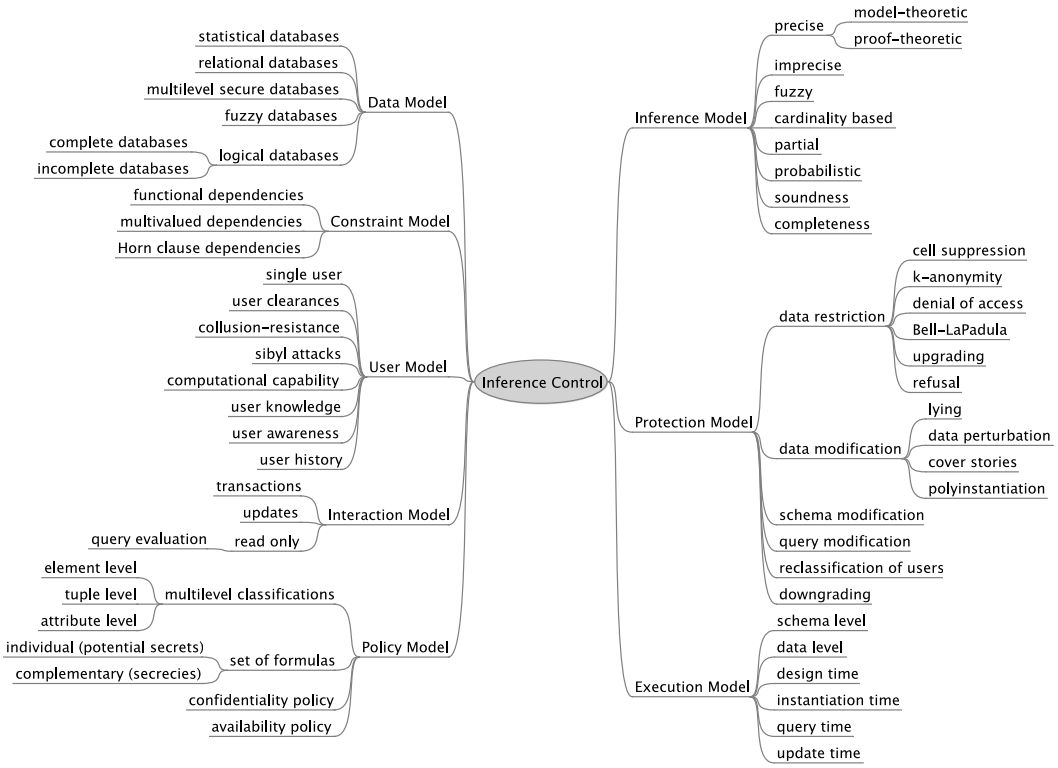


Figure 3: Taxonomy of inference control

for data storage where several iterations of querying or manipulating data items are desirable (the "general purpose databases"). We will relax this distinction a bit and start with a short description of the "relational model" as a kind of base case. Then we describe its extension to the "multilevel secure data model", as well as "statistical databases" as special purpose applications of the relational model. Lastly, a more abstract model is introduced: the "logical databases".

### 2.1.1 Relational Databases

The relational database model has been introduced by Codd ([Cod70]) and forms the basis of most of today's database management systems. The basic notions are those of a database schema, defining several relation schemas and a set of global constraints where a relation schema defines a set of attributes and a set of local constraints. The invariant static definition of the database schema forms a mold for the actual database instance, where each relation instance consists of a set of data tuples obeying all the constraints.

### 2.1.2 Multilevel Secure Databases

In multilevel secure databases, a relational database is accompanied by a lattice of classifications: a classification denotes a level of confidentiality; the lattice defines a partial order on classifications. Classifications are then assigned to "security objects". This can be done in differing granularity: either at the *element level* (that is, each single attribute value of a tuple is considered a security object), at the *tuple level*, or the *attribute level* (for combinations of attributes independent of any attribute values). More simple cases are relation level and database level classifications. [QL92] show that a tuple with element level classification can be replaced by a set of tuples with tuple level classification – at the cost of having more redundancy in the database. Pernul [Per94] notes that "[...] careful labeling is necessary because otherwise it could lead to inconsistent or incomplete label assignments". A host of work attends to the many-faceted problems of inference control in MLS databases; we will look at some of them more closely in the following subsections.

### 2.1.3 Statistical Databases

A statistical database consists of relational tables that are released to the public (for example by a census bureau) as a basis for statistical analyses. Such a database is meant to execute aggregate queries on it (like summing values, counting rows etc.) yielding the desired statistical values as results. The main focus of inference control in statistical databases is that no sensitive information about an individual entity can be derived. In other words: if the data contain personalized information about individuals, attributing particular attribute values to an individual should be impossible. This has already been a research topic for decades (see for example

[DS83]) but is still of high interest today (see for example [DF02] and the Protection Model below).

### 2.1.4 Logical Databases

From a general point of view, a database instance can be seen as an interpretation for some of the syntactical elements of a logical language $\mathscr{L}$ that forms the basis of the system. As such, the instance should be a "model" of the database constraints – that is, the interpretation satisfies all the constraints based on appropriate notions of models and formula satisfaction. In this model-theoretic case, one can assume that the interpretation (and thus the database) is "complete": for every closed formula a definite evaluation (either *true* or *false*) can be given.
Even more generally, a database instance can represent not just one but a set of models: The only restriction is that the instance (plus the constraints) form a consistent set of formulas; the logical closure of the formulas (according to some consequence operator) contains all the true propositions. In this case, the database is "incomplete" as the evaluation of some closed formulas may be indefinite. With an incomplete database, the instance can also be called a "knowledge base" or "belief base". These terms are often used in artificial intelligence and related areas; see for example [Win90] for a comparison.
Essentially, this logical point of view can subsume the above special cases when defining an appropriate base logic and axiomatically specifying the necessary constraints and restrictions. For example, [SW92] give a belief-based semantics for MLS databases. The whole database is represented by a Kripke structure – that is, a set of partial databases (one partial database for each classification) and a binary relation between them whenever the classification of one database dominates the classification of the other.

## 2.2 Constraint Model

Database constraints are crucial for inference control; they are the primary knowledge that enables users to draw inferences – of course under the assumption that the user is fully aware of these constraints. Put differently, database constraints can be seen as the set of rules that a user employs to deduce information. Restricting the class of constraints influences tractability and performance of inference control. That is why some authors just consider functional dependencies or multivalued dependencies (for example [HS96] and [SÖ91]). [DH96, DH94] extend their initial procedure for functional dependencies to transitive associations in general. In [BFJ00] Horn clause constraints are permitted.
Generally, the inference control approaches assume that the database constraints are well-defined in the sense that they are satisfiable and hence there is at least one database instance that satisfies them.

## 2.3 User Model

As already illustrated in Section 1, a database user may have certain background knowledge. When trying to model a real-world user, we have to admit that we can merely produce an approximation of human knowledge. That is, the representation of the user's knowledge in an inference control system (for example as a set of logical sentences) is most likely neither expressive nor comprehensive enough to incorporate the user's "real" knowledge. Moreover, influencing or controlling the user's knowledge acquisition from other information sources (apart from the to-be-controlled database) is probably impossible if not undesirable anyway. That is why inference control in databases necessarily has to assume a closed system where a user's knowledge (background knowledge and knowledge gained by querying and manipulating the database) is known inside the system and cannot be changed from outside of the system. Mostly, it is also assumed that the user's knowledge can only be incremented; that is, technically the user cannot be forced to forget some part of his knowledge. Jajodia and Meadows [JM95a] describe these facts as follows:

> "An inference of sensitive data from non-sensitive data can only be represented within a database if the non-sensitive data itself is stored in the database. We have no way of controlling what data is learned outside of the database, and our abilities to predict it will be limited. Thus even the best model can give us only an approximate idea of how safe a database is from illegal inferences. This fact should always be kept in mind when dealing with inference problems."

Consequently, in all inference control approaches, some fundamental open questions remain with respect to the User Model: In how far is it possible to realistically model human knowledge and reasoning? How can the system avoid or notice knowledge acquisition from other sources external to the system? How can the system avoid or at least detect collaboration of users? Yet, we argue that all kinds of "traditional" access control mechanisms suffer from these difficulties, too, when designing a secure system; even standard access control policies have to make assumptions about user knowledge and collaborations to be effective – or they simply ignore theses issues.

For statistical databases, inference control is mostly executed without an explicit user model. The data in the statistical database are modified to achieve indistinguishability for individual entries – independently of any assumed user knowledge. Obviously, a user with statistical knowledge about some of the individual entries might be able to deduce facts about other individual entries from the released data. In MLS databases, the typical scenario is a multi-user one: a "clearance level" is assigned to each user. The clearance level – loosely speaking – determines which data the user is allowed to read or write (see the Policy Model for more details). [CC94] introduce a finer grained user model that gives the same user different clearances for different attributes (for example, access to *Secret* values for one attribute and to *Unclassified* values for another).

Some of the approaches for the MLS model and most of the approaches for relational and logical databases model user knowledge in their inference control systems. In some cases a user history is maintained, that is augmented with the database's responses to the user's queries. This is for example the case for the "data-dependent mode" in [BFJ00]. In other cases, a user's a priori knowledge is taken into account: In [HTS94, HS96] while the data model is relational, the authors model a user's a priori knowledge in a relation with fuzzy values.
Typically a single-user scenario is assumed; this single user however could also represent a group of users that are assumed to collaborate. In most cases the database is not exposed to updates. Some approaches go further than that as will be amplified in the following two subsections.

### 2.3.1 Collusion and Sybil Attacks

Obviously, the single-user scenario but also the MLS multi-user scenario are not realistic in general. In a real-life setting, the strict mapping of one database user identity to one existing person can easily be subverted: for example, users have the opportunity to collaborate and share information they received from the database (this is known as an *inter-user collusion attack*), or a person can generate multiple database user identities, use all of them to query the database and accumulate all results (this is known as a *sybil attack*).
An approach (using cryptographic primitives) to counter such attacks is the one by Staddon [Sta03]: She applies results from the area of secure group communication to inference control. Even this approach relies crucially on how realistic the User Model is regarding the extent in which users collaborate (see Section 3 for more information).

### 2.3.2 Updates

Quite intricate issues arise if the database is subject to updates by the database administrator. If a user history is maintained, the updated database may contradict the now outdated user history; truthful responses to upcoming queries may lead to an inconsistent user history. One way to avoid such inconsistency is that the database actively informs the user about changes (with respect to his knowledge accumulated before the update). However, one could also argue that the database should be more reticent and give updated information only if the user poses a query resulting in a response that would leave the user history in an inconsistent state. Alternatively, one could allow an inconsistent user history and use a paraconsistent logic to administer and reason with the user history. The situation becomes even more problematic if not only the data but also the database schema is allowed to change.
So far, most of the inference control approach ignore the update problem. One not totally satisfactory solution is the "Dynamic Disclosure Monitor" ($D^2$Mon; see [FTE01]): The authors annotate entries in a user's history with updated informa-

tion. See Section 3 for a discussion of the shortcomings. As a totally different approach [MSS88] assume that the database user himself is allowed to update the data via transactions and the user's inferences are based on the denial to execute some transactions.

## 2.4 Interaction Model

User interaction is another crucial point for inference control because it influences the kind of inferences a user can make. Most approaches just consider read-only databases, yet some assume that users also can execute updates on the database (see for example [JS91, MSS88]).

## 2.5 Policy Model

The main concern of inference control for general purpose, MLS and logical databases is *confidentiality of secret data*. In some articles, the specification which data is secret is simply called "security policy". We will use the term "confidentiality policy" instead as it connotes more attention to the fact that confidentiality is the primary security goal of inference control. Obviously, the definition of a what is a secret is closely connected with the user model. That is, a confidentiality policy links a user to a set of permissions or prohibitions to know some data in terms of query evaluation in the database.
For statistical databases the security goal is quite different: the user can compute statistics on the data but should not be able to learn individual entries as this is considered sensitive information. These security requirements are not specified in an explicit confidentiality policy but in terms of statistical properties: Denning ([Den82], page 339) calls perfect secrecy in a statistical database the fact that no sensitive statistic is disclosed but names it an impracticable and unreasonable objective for statistical databases. She instead gives a definition of protection (based on confidence intervals) and a definition of security (based on non-sensitive data). She concludes that inference control mechanisms for statistical databases must be imprecise. See also the Protection Model (Section 2.7) for further details.

Depending on the Data Model, confidentiality policies come in different flavors: For an MLS database, [DdVLS02, DdVLS99] specify confidentiality requirements with so called "lower bound constraints"; they express the lower bounds of classification for attributes in an MLS database. Although the authors do not explicitly mention it, such a policy could be specified for any user's clearance level. [BFJ00] assume a single ("universal") relation. Basically, their confidentiality policy consists of classifications for queries; that is, it specifies to which queries the user (for a given user identification) is not allowed to know the responses. From this policy, in their "data-dependent" mode (where the database entries are taken into account), they compute the set of all database entries (in this case, partial tuples) that the user is not allowed to access.

For logical databases, [SdJvdR83] specify the confidentiality policy as a set of closed formulas. They introduce the semantics of "secrecies" for a policy: the user is not allowed to learn whether a policy entry is *true* or *false* in the database instance. [BKS95] introduce a different concept of a secret: the user is not allowed to learn that a policy entry is *true* in the database instance.
Note that an access decision according to a fixed confidentiality policy at one point of time is a binary decision: either the user is allowed to retrieve the present query response, or he is not.

*Availability of correct data* is always a secondary goal for inference control: not responding to any user queries whatsoever would definitely ensure confidentiality but the database would not be of any use for the user at all.
Often availability (sometimes also called "visibility") is only an implicit requirement. Yet, some approaches involve explicit availability policies. For instance, [DdVLS02, DdVLS99] use "upper bound constraints" to express that an attribute should not be classified higher than the classification permitted by the constraint.
For statistical databases, availability also goes under the name of "precision" and can be viewed as ensuring that the computed statistics on the modified database are as close as possible to the actual statistics on the original database. Denning ([Den82], page 339) mentions the conflict between availability of correct data and confidentiality of secret information:

> "Whereas secrecy is required for privacy, precision is required for freedom of information."

## 2.6 Inference Model

As already described above in the Policy Model, in statistical databases inferences are based on a combination of several statistics of the database entries.
In all other data models, inferences occur whenever the user applies the plain representation of his knowledge (as for example the user history) to deduce data beyond this representation. A *harmful* inference is one with which the user can deduce secret data (as specified by the confidentiality policy).
Some approaches present a restricted set of inference rules that the user can base his inferences on (see for example [YL98a, YL98b]). Analogously, the restriction of database constraints to certain constraint classes (like functional and multivalued dependencies or Horn clause constraints; cf. the Constraint Model) effectively bounds the assumed inference capabilities of the user. Such restrictions lead to inference systems that are sound, meaning that all computed inferences can actually be drawn by the user; but such systems might be incomplete in the sense that the user is able to draw more inferences than the ones modeled in the system.
There are also approaches that base inferences on a probability distribution and engage an information-theoretic notion of confidentiality ([MS07, GH08]; see also Section 3).

## 2.7 Protection Model

In general, there are two basic protection mechanisms that achieve confidentiality by reducing the availability of correct of data in the database: *data restriction* (that is, restriction of access to some data items) and *data modification* (that is, modification of some data items); yet, both come in different flavors depending on the underlying data model. We amplify the description of these core techniques in the following subsections. Both mechanisms only affect the database and leave the remaining input and the confidentiality requirements unchanged; we do not consider other techniques like schema modification, query modification, reclassification of users or downgrading.

### 2.7.1 Data Restriction

For statistical databases, protection has to be conceived on the basis of statistical evaluation techniques. For instance, whenever the size of a query response is under a certain threshold a query has to be denied because identification of individual tuples is possible. As an example, Denning ([Den82], page 337) states the "$n$-respondent, $k$%-dominance rule", which defines query results as sensitive whenever they contain $n$ or less than $n$ tuples but cover more than $k$ percent of an aggregated value (for example, summed total of salaries). "Cell suppression" removes entries from the database which also effectively restricts access to the data.

For relational, MLS and logical databases data restriction is enforced in terms of denial of access. Sicherman, de Jonge and van de Riet ([SdJvdR83]) pioneer "refusal" in logical databases. In MLS databases, data restriction is mostly based on the Bell-LaPadua access policy stating that a user with a particular clearance level is allowed access to data items of a certain classification level and below but is denied access to data items with a higher classification level (in the case of a read-only database).
[DdVLS02, DdVLS99] ensure confidentiality by upgrading attributes; that is, some attributes retrieve a higher classification than in the original database. Combined with the Bell-LaPadua policy this effectively denies access to data items that potentially cause harmful inferences. They use a graph structure (a "constraint graph") to compute the upgraded classifications.
[BFJ00] use a chase algorithm (that is, an application of their Horn clause constraints on the user history) to determine whether the present response discloses confidential data; if so, the present query is denied.

### 2.7.2 Data Modification and Cover Stories

For statistical databases, there are strategies that modify the database entries such that the individual tuples do not contain correct information anymore but the overall statistics are not affected; the terms "perturbation" and "addition of noise" are often used as synonyms for these modification strategies (see also [Den82],

page 371). Similar strategies have been developed under the term "$k$-anonymity" ([Swe02, MGKV06, LLV07]). The work on inference control in online analytical processing (OLAP; [WLWJ03]) systems is also related to this.

While data modification is quite common for statistical databases, for relational, MLS and logical databases the situation is more complex: their purpose is to return individual data items; data modification may however affect the *integrity* of such data items and consequentially reduce the *reliability* of database responses in general. Yet, for MLS databases, addition of incorrect data items may sometimes be necessary for sake of consistency. *Cover stories* in MLS databases ensure a consistent database view to a low-level user without revealing high-level information; this is basically achieved by adding additional harmless tuples that cover up for confidential tuples. [CC94] argue that without cover stories (that is, just using refusal to answer as data restriction), the existence of sensitive information can be disclosed.
One way to add and manage cover stories is to have a polyinstantiated database ([SJ90, SJ92]). For such databases, [Den87] coins the notion of an "apparent primary key", which can occur multiply in a polyinstantiated database as the actual primary key now is only unique with respect to classification levels. Using this concept, a database instance with element or tuple level classification is defined to be polyinstantiated if it contains tuples with identical apparent primary key values. Databases with relation level or attribute level classification are not affected: "Polyinstantiation does not arise when access classes are assigned to relations or individual attributes." (see [JM95b]). In polyinstantiated databases, a new notion of "polyinstantiation integrity" has to be defined that ensures that users with different clearances have a view of the database instance that is consistent with the database constraints. Its correct definition was discussed for a long time by several researchers; [QL92] describe some of the different definitions and their effects. [JS91] define insert, delete and update operations for polyinstantiated databases.
[CC94] emphasize the need for consistency of level-wise views; they propose an algorithm that restores a consistent view for each level by "merging" data items of a level and all lower levels based on some heuristics. Their first assumption is that high-level data items are more reliable than low-level data items. Then they introduce "topics" that group together semantically related values. Most notably, they draw a connection between polyinstantiation and modified database entries:

> "In a situation where data are polyinstantiated, the high users try to lie to the low users in order to cause them to believe something which is incorrect."

Cuppens and Gabillon ([CG01]) describe cover stories as "lies introduced in the multilevel database in order to protect some existing higher classified data". They do not use polyinstantiation but instead explicitly declare which data comprise a cover story. They state that this technique avoids semantic ambiguities inherent to polyinstantiation; with polyinstantiation, it may for instance not be possible to

distinguish the cover story from the correct data.
Last but not least, the belief-base approach of [SW92] presents differing views of the world according to a user's clearance. Their partial databases can thus also be seen as containing cover stories for users of insufficient clearance.

Summing up, we can say that the great interest in cover stories shows that there is indeed a need for data modification, and data restriction in some situations does not suffice to achieve confidentiality.

### 2.8 Execution Model

Mode and time of execution are distinguishing characteristics of the different inference control methods. The most distinctive feature is whether inference control is executed only in a (what we call it) *content-independent* way or in a *content-dependent* way. The terms "data level" (see [YL98a]) or "data dependent" (see [BFJ00]) are equivalent to our term "content-dependent". It is commonly agreed that content-dependent inference control increases availability (compared to content-independent inference control): inferences can be controlled in a finer grained way because the outcome depends on the content of the database instance and not just on the database schema.
Moreover some approaches make a coarse distinction between "online" and "offline" inference control. Others employ the term dynamic inference control for inference control at query time (see for example [Sta03]) or when updates are allowed (see [Qia94]). To avoid confusion, we will use the following unambiguous terms to denote the point of time when inference control takes place: *design time*, *instantiation time*, *query time*, or *update time*.

## 3 A Selection of Prior Work

This section organizes some of the inference control approaches in a more coherent fashion. The reader not interested in the minor details of these approaches is encouraged to skip this section with a clear conscience.

For statistical databases some successful applications have been developed for example in the Datafly project ([Swe97]). Inference control for statistical databases has to be put into practice when statistics are released to the public.
As one example for inference control in statistical databases, the approach of Chang and Moskowitz [CM00] is described in more detail. They consider the two techniques "blocking" and "aggregation" in a statistical database. They identify combinations of attributes whose values are "similar" to the values of so-called target attributes; similar attribute values are those that lead to a disclosure of sensitive data (in the worst case there is a bijection between the similar values and the sensitive target values). Similar values are then blocked (that is, replaced by a question mark) or

aggregated (that is, a number of more specific values are summarized by a more general value). As a measure of data quality, a probabilistic "quality index" is defined as well as a "ratio of reduction". Data modifications are executed as long as the reduction ratio is not reached. Assuming that the original data already have some error probability, the reduction ratio acts as a threshold for the decrease of data quality in the modified database instance. As already mentioned in Section 2.7, a huge amount of other statistical techniques exist. A good collection of these techniques can be found in [DF02, DFT04, DFF06].

The approaches for relational databases are quite dispersed. Mazumdar et al. ([MSS88]) consider a database environment with a set of specified transactions. They analyze whether a user can draw conclusions by observing success or failure of a transaction; causes of a failure can be the transaction's not satisfying integrity constraints or preconditions. They assume a single user being aware of the constraints, preconditions and the transaction's program. Secret information is specified as a set of open formulas; a revelation of a secret happens if the user learns a valid instantiation of the free variables. The transactions are analyzed at compile time with the help of a theorem prover who imitates the reasoning of the user; yet, this method restricts capabilities of the user by modeling him as a sound but incomplete deductive system. In particular, the authors examine iterated executions of transactions and partial inferences. They formally define a partial order on secrets defined by logical implication: a "stronger" secret allows a user to deduce other secrets.
Delugach et al. [DH96, DH94] approach the inference problem in a semantic way: they employ conceptual graphs to visualize "facets" of a database schema (and possibly some instances). Different "facets" denote semantic relationships between entities or activities. While they designed a tool to identify inference paths from unclassified to classified or sensitive information based on a lossless-join algorithm, user interaction of a so called inference analyst is crucial to build the conceptual graph and weigh inference paths according to their severity.
Hale et al. ([HTS94, HS96]) emphasize the existence of fuzzy inference in precise and fuzzy relational databases based on functional dependencies. Fuzzy inference also covers classical precise and imprecise inference: Precise inference means the disclosure of singleton sets, imprecise inference is the disclosure of a set of values and fuzzy inference is the disclosure of fuzzy sets of values. The authors propose to add common sense knowledge as a fuzzy "catalytic relation" to a precise database. They combine the fuzzy and precise notions in an abstract relational database model. This also includes a notion of consistency between precise, imprecise and fuzzy values and a notion of redundancy of fuzzy values; these are based on partitions: attribute domains are partitioned by equivalence relations giving a form of semantically indistinguishable elements.
Yip and Levitt ([YL98a, YL98b]) design a content-based inference detection system for a single (universal) relation. They present several inference rules – which taken together are sound but not complete – that a user can apply to a set of query

results; in this sense this inference detection system assumes a user with restricted reasoning capabilities. Query results are *multisets* of return tuples, which leads to the situation that the user can also draw cardinality-based inferences. The main concern is the detection of return tuples which can be combined and thus disclose additional information. It remains however unclear how the confidentiality policy is specified; the authors just give the combination of two attributes as an example. The inference rules can either be applied online (although the authors state that this involves significant runtime overhead) or offline (unfortunately the authors do not explain how this can be achieved). They also give experimental results for randomly generated tables and queries. They observe the percentage of disclosed data and performance of the system for varying parameters (for example, number of attributes and tuples or distribution of data values).

MLS databases have attracted some attention in military environments where clearance levels of users can be established quite easily. The SeaView project (see [LDS+90]) is one of the most comprehensive inference control approaches for MLS databases. SeaView stores single-level relations and combines (reconstructs) them to multilevel relations as logical views. In the same context, Qian [Qia94] defines the restriction of an MLS Database (with tuple level classification) to a range of classifications. Integrity constraints can have assigned ranges of classifications. She considers "static" inference channels (in databases without updates) and "dynamic" inference channels (in databases with updates).
For the DISSECT system, Stickel ([Sti94]) describes a procedure that eliminates inference channels in an MLS database. He assumes a total order of classifications given; in the basic setting classification is possible only at the attribute level. A given inference channel as well as the classification lattice are translated into a set of Boolean constraints. A model for these constraints is searched for with the SAT solving procedure of Davis, Logeman, Loveland and Putnam (DPLL) (see [DLL62, DP60]). Because the classification lattice is explicitly encoded in the problem, only minimal models of the constraints assure an optimal upgrading of objects: an object's classification is not higher than necessary in a minimal model. Additionally, the author describes how to assign costs to pairs of an object and a classification. These costs define the "feasibility and desirability" of assigning the object the given classification. The DPLL procedure can than be adopted to find a model with minimal costs. Extensions to tuple level classification and to partially ordered classification lattices are sketched.
Cuppens and Gabillon ([CG01]) define a more elaborate form of cover stories for MLS databases. In the authors' data model, a multilevel database consists of (ground) facts each of which is classified at a certain level; integrity constraints (written as closed formulas) have classification levels, too. Cover stories can either be facts or integrity constraints and are stored separately from the data in the database. As already mentioned in Section 2.7.2 this allows for distinguishing cover stories from correct data. The authors define a "view" of the database at a certain classification level $l$ to consist of all facts and constraints classified at level

$l$ or below. They define a notion of consistency of databases and views. Security (i.e., confidentiality in this case) of a view then means that the view itself and all views at a lower level are consistent. They also handle updates of a database with cover stories: an update consists of a transaction at a certain "transaction level" which is equal to or below the user's clearance level. A transaction may only insert, update or delete data at its transaction level. It can only be committed if the resulting database view at the transaction level is secure. Transactions can render the database inconsistent for views at levels above the transaction level; in such a case the security property has to be restored either automatically by the database system or manually by a security administrator. The automatic restoration procedure deletes a fact from the database if there is a duplicate entry at a lower classification level; it deletes a cover story if it does not match a fact in the database. Additions of cover stories may also be necessary; new cover stories can be computed with the help of minimal inconsistent sets. In some situations however automatic determination of cover stories is impossible and semantic guidance by the security administrator is essential.
Dawson et al. ([DdVLS02, DdVLS99]) avoid inference problems in an MLS database by computing minimal classifications for attributes. They assume a lattice of classifications given. Confidentiality requirements are specified by a set of lower bound constraints, determining the classifications the attributes should at least be assigned. Explicit availability requirements are stated in a set of upper bound constraints, denoting the classifications the attributes should at most be assigned. This way, a subject's prior knowledge can also be specified. The constraints are visualized in a constraint graph and classifications are established by back-propagation from leaves in the constraint graph. For the simplest case (an acyclic graph with only "simple constraints") there is a unique minimal solution, whereas for an acyclic graph with "complex constraints" there may be multiple minimal solutions whose order of appearance depends on the evaluation order of the constraints. A cyclic graph with only simple constraints can simply be resolved by assigning all attributes in a cycle the same classification. The most difficult case is a cyclic graph with complex constraints. The authors devise a stepwise "forward-lowering" algorithm to accomplish a minimal solution. The set of lower bound constraints is always consistent: all lower bound constraints are satisfied when all attributes are classified with the top element of the lattice. Yet inconsistency can occur due to upper bound constraints. The authors propose to ignore the upper bound constraints in such cases, effectively giving confidentiality precedence over availability.
The "Disclosure Monitor" (DiMon) of Brodsky et al. ([BFJ00]) ensures confidentiality in a read-only database while maximizing availability by extending the mandatory access control (MAC) mechanism. Their underlying system consists of a universal relation, a set of Horn-clause constraints, classifications for users as well as tuples or queries (i.e., attribute combinations), and a history of user queries. The authors present algorithms that compute a sound, complete and compact "disclosure cover" that represents the inferences that can be drawn by the user; this can

either be based on a concrete instance and a user history containing pairs of queries and the corresponding results (they call it "data-dependent mode") or on a history just containing the queries (the "data-independent mode"). They compute the disclosure by employing a chase of the Horn-clause constraints on the given user history and the current query (accompanied by its result in data-dependent mode). Afterward they check whether the chase led to a fact that the user is not allowed to know. The proof of termination of their algorithms is based on the fact that applications of Horn-clause constraints do not introduce new symbols.
With the "Dynamic Disclosure Monitor" ($D^2$Mon; [FTE01]) the authors extend the static DiMon with an update consolidator that expands a user's history file with values added to (or deleted from) the underlying MLS database. They claim that their approach improves availability, because inferences based on "outdated" information leading to invalid conclusions are considered harmless; this includes that wrong responses can be returned to the user. In the user's history file, results once given are kept and eventually annotated ("stamped") with the updated values; that is, when checking for harmful inferences, the user history also includes the update information of which the user is (currently) not aware. The authors mention a second module for controlling statistical inference without giving details. In [TFE05] the authors give an improved algorithm for applying ("chasing") a set of constraints on the history file. Yet this approach suffers from some unspecified issues: The authors do not define a user model in terms of system awareness of the user (Does the user know which information is classified? Does the user know the workings of the algorithm?) neither do they consider the user's pondering over a refused query or a query result that would actually enable the user to infer a secret. In their example, an "outdated" tuple containing salary and name (though classified) can be inferred if the salary changed in the meantime. Assuming that the user knows the MLS classification (that is, the confidentiality policy), we consider it reasonable that the user might have the ability to reflect on the given results and conclude that at least one of his inference premises must have been invalidated in the current database.

Sicherman, de Jonge and van de Riet ([SdJvdR83]) are among the first to consider inference control in logical databases. They introduce secrets, queries and user knowledge as arbitrary well-formed (closed) formulas in first-order logic and a complete but possibly infinite database instance as an arbitrary structure in a logic. They emphasize that this makes their system a deterministic system (in contrast to other systems dealing with probabilities). Secrets are defined to be sentences that hold in the database. They also assume that information in the database is absolutely correct and does not change over time. And they presume that there is only a single user in a single "session". As a main point they also consider that the user has knowledge about the system's refusal mechanism. As a consequence of this, inferences based on the mechanism are also considered and thus negations of query results are checked for their harmfulness. They also define a secret to be "safely concealed" whenever there is an "alternative system" that gives identical

responses (as the original system) but the secret does not hold such that "the user cannot deduce which system he is talking to". The authors introduce the notion of "secrecies", and take into account the user's awareness of secrecies. In several theorems they establish conditions for safety of answers. In their conclusion they name several relaxations of their system settings, as for example allowing "wrong answers", reducing the user awareness or allowing changes in the database. They also raise the question of feasibility of such a system.
[BKS95] examine privacy concerns in propositional databases with the help of modal logic. They define modalities to express the secret information and the beliefs of the user. They also define axioms that model the reasoning behavior of the user. They introduce lying as a data modification technique. Then they extensively study lying as well as refusal in both complete and incomplete databases with a "logic of interaction".

There are some techniques that embed inference control into a broader context. An approach (using cryptographic primitives) to counter inter-user collusion or sybil attacks is the one by Staddon [Sta03]: She applies results from the area secure group communication to query-time inference control. On the one hand, each user is provided with a set of cryptographic keys (which might be identical for all users or at least some users have some keys in common if they are assumed to collaborate). On the other hand – after all inference channels in a database are appropriately identified – all objects in such an inference channel are assigned tokens encrypted with (a subset of) the users' keys. At query time, whenever a user queries an object in an inference channel, he sends along the object's token encrypted with one of his keys. From all other objects in the inference channel, their tokens encrypted with exactly this query key are revoked, signaling that this key has already been used to query an object in this inference channel. All further queries with the revoked key are then denied. The number of keys that are used to encrypt tokens in an inference channel is strictly less than number of objects in the inference channel, such that the users are never able to "complete" the inference channel – that is, to query all objects in the channel.
Woodruff and Staddon [WS04] combine collusion-resistant inference control and private information retrieval and thus not only ensure confidentiality of data but also privacy of a user's query: they devise a "private inference control" (PIC) scheme to prevent the database server from knowing the query results that it returns to the querying user.
Chen and Chu ([CC06]) propose a centralized probabilistic inference detection system: they consider knowledge acquisition from different sources but assume that the user's knowledge gain can be fully modeled in the centralized "knowledge acquisition module". They model semantic dependencies between data with a graph structure and specify sensitive nodes in the graph. They assign conditional probabilities to attribute values based on the dependencies and map the graph to a Bayesian network. In the Bayesian network the probability of a harmful inference can be calculated at runtime with respect to the user's query history. If a threshold

is exceeded, the current query is denied.

Last but not least, there are some information-theoretic approaches that – although not running under the name inference control – can be used to detect inferences before some data ("views") are released to the public. Those views are defined over a relational database in [MS07] and over an "information system formalism" in [GH08] (which contains the relational databases as a special case). The common feature of these approaches is that they assume that the user has fixed a probability distribution over all possible database instances; then their notions of confidentiality (called "security", "privacy" or "safety") require that the a priori probability of a response to a "secret query" be equal to its a posteriori probability (after releasing the views). [MS07] incorporate a priori knowledge of the user which is added as a condition while computing the probabilities of the secret query. [GH08] assume that the database schema is exposed to changes ("evolves"); the a priori probability is computed on the original schema and the a posteriori probability on the evolved schema. Most notably, both approaches later on identify conditions under which the assumption of a probability distribution is unnecessary. [MS07] reduce their probabilistic framework to checking containment of "critical tuples" in the views; they concede that, "[t]his result translates the probabilistic definition of query-view security into a purely logical statement, which does not involve probabilities. This is important, because it allows us to reason about query-view security by using traditional techniques from database theory and finite model theory". On the other hand, [GH08] get rid of the probabilities by checking whether the system returns the same responses before and after the views are released. Moreover, in both approaches availability is not considered.

## 4 Controlled Query Evaluation

In spite of the differing ambitious approaches for inference control, no generic representation has been found so far. Such a uniform representation would preferably incorporate and combine all prior approaches. Indeed, due to its generality, this representation should have a logical basis.

We now introduce the extensive work on inference control called Controlled Query Evaluation (CQE; see [BW08b, BEL08, BW08a, Wei08, BL07, BBWW07, BW07, BB07, BW06, BB04a, BB04b, BW04, BB01, Bis00]). Controlled Query Evaluation is a logic-based framework for inference control and covers a broad range of parameters. In our opinion, this predestines CQE to be the general representation for inference control we are looking for.
It is fundamental for CQE to choose an appropriate logic. So far, propositional logic and a fragment of function-free first-order logic with equality have been investigated. Figure 4 gives a schematic view of a query-time ("censor-based") CQE system, whose components are detailed in the following.

We assume here that installation, initialization and maintenance of the CQE system is a task to be distributed between three different kinds of administrators (we will use these administrator types here though they are not explicitly introduced in the references cited above). The database administrator *dbadm* has the task of maintaining the database; plus, it falls into his responsibility to choose the logic and determine a language $\mathscr{L}$ of the logic that specifies a syntax for all necessary input to the system. The security administrator *secadm* declares the confidentiality policy. He can do this essentially independent of the database as long as he sticks to the syntax as dictated by the language $\mathscr{L}$. As a third expert we introduce the user administrator *useradm*: He is the one who models the a priori knowledge of the database users in the system, again using the language $\mathscr{L}$. Originally, CQE was de-

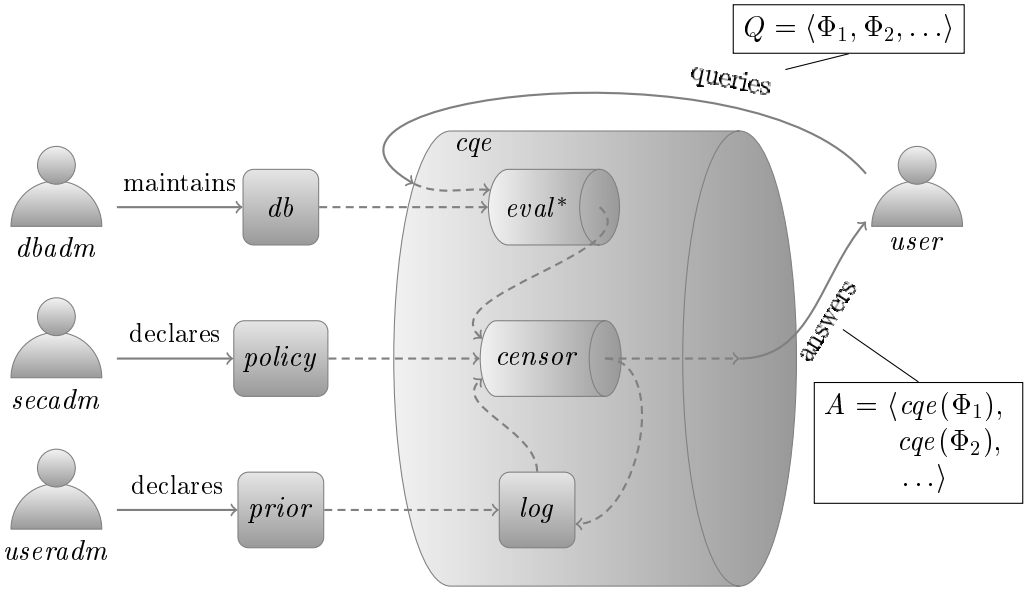


Figure 4: Schematic view of censor-based CQE

signed as a query-time inference control system as visualized in Figure 4. It is based on the ordinary query evaluation function *eval** that will be defined shortly. A *censor* module checks at query time whether unwanted inferences can occur based on a user history called *log*. The censor module essentially decides whether distortion of the answer is necessary or not. The censor may be accompanied by a "modificator" module, that distorts the answer based on the censor's decision; the modificator is left out of Figure 4 for simplicity. We develop the elements and variants of CQE in detail in the following paragraphs.

## 4.1 Data Model

The CQE data model is of the form of a logical database. A database instance (denoted as *db*) is defined as a finite and consistent set of closed formulas in the

chosen logic. The data model is furthermore subdivided into the following two cases (as were already introduced in Section 2.1.4):

- complete database: an instance *db* of a complete database is a finite set of ground atoms in the underlying logical language $\mathscr{L}$. Each ground atom in the instance is evaluated to the truth value *true*; it represents a true fact in the modeled world. By employing a closed world assumption, all other ground atoms have the value *false*. More complex formulas can be evaluated iteratively from the values of the ground atoms. In this way, the database corresponds to an interpretation with a finite positive part in a classical two-valued logic. Accordingly, to every user query the database can return a definite response as defined in the Interaction Model.
  Complete propositional databases are investigated in [Bis00, BB01, BB04b, BB04a, BW06]. The complete logic-based setting basically coincides with the relational data model whenever predicate logic is used as a specification language and we furthermore assume that the set of predicate symbols and all relations are finite. Query-time CQE for relational databases is studied in [BB07, BEL08, BL07]. In [BEL08, BL07] schemas for the considered relations are explicitly stated – including functional dependencies as constraints. In the other approach ([BB07]), the database schema (called *DS*) is left implicit and its syntactical elements can be deduced from the database instance *db*; in this case and in the propositional approaches, the database constraints are incorporated into the user knowledge *prior*.

- incomplete database: an instance *db* of an incomplete database is a consistent set of arbitrary sentences in the underlying logical language. In particular, no closed world assumption is made and an incomplete *db* may for example contain disjunctive and existential formulas, which makes some query responses undefined (see [BW08a, BW04, Wei08]).

## 4.2 Constraint Model

In CQE, the assumption is made that the user is aware of the database constraints. That is why database constraints form part of the user knowledge *prior*. More generally, the knowledge of the user as a whole is seen as a set of constraints for the CQE system: all database responses have to be consistent with the user knowledge (see also the User Model).
While for the propositional approaches no syntactical restrictions of the constraints are necessary, in the relational approaches, the system is confronted with the issue of undecidability. That is why for some specialized applications (see [BL07, BEL08]), database constraints are restricted to functional dependencies and database relations are required to be in object normal form (ONF). The restrictions for [BB07] are explained in the Execution Model below.

## 4.3 User Model

CQE is run in single-user mode (cf. Section 2.3); collusion resistance and sybil attacks have not been examined so far but consideration along the lines of [Sta03] might be possible. The user is modeled in the system as a set *prior* of closed formulas of $\mathscr{L}$. For query-time CQE, a user history called "user log" or simply *log* is maintained in the system and incremented with each query response; that is, $log_i$ contains all entries in $log_{i-1}$ plus the responses to the $i$th query $\Phi_i$ in the query sequence $Q$ (cf. Figure 4) and $log_0$ is equal to the a priori knowledge *prior*. For incomplete databases a transition to modal logic was made in order to account for secrets with an *undefined* value (see [BW08a, BW04, Wei08]). With this user model, CQE encounters the same limitations as do other inference control systems with regard to a correct and comprehensive representation of the user knowledge; this was already discussed in Section 2.3.

Beyond the specification of the user knowledge, several awareness assumptions can be made in the system settings. For example, CQE can account for a user that knows the specification of the confidentiality policy (the "known policy" case) or not (the "unknown policy" case). The user also has system knowledge in so far that he knows which protection mechanisms the CQE system employs.
Moreover, the user is assumed to be "sophisticated" in the sense that he is able to reason rationally based on his knowledge and his awareness of the system settings. In order to achieve a strong attacker model, it is also assumed that the user is not bounded by computational restrictions (whereas the CQE system as the defender clearly is). An alternative user model of a "plain" user who does not exploit all his reasoning capabilities but instead believes anything the database responds is briefly considered in [Wei08]. Plain users can pose a threat in the strictly logical CQE setting and could be examined further.

## 4.4 Interaction Model

From the user's point of view, CQE offers read-only databases; that is, user interaction consists of the evaluation of queries with respect to the current database instance. Its basis is ordinary uncontrolled query evaluation (denoted $eval^*$ and defined in Definition 7.4). In order to control the query responses, ordinary query evaluation is wrapped in a controlled query evaluation function (for brevity denoted *cqe* in Figure 4). For complete relational databases it was shown how controlled evaluation of open queries can be achieved by repeated invocations of controlled evaluation for closed queries; this approach is based on a fixed enumeration of an infinite set of constant symbols and an additional completeness test.
In some cases the query language was restricted (for example, to contain only existentially quantified ground atoms in [BL07], or to the positive existential relational calculus in [BB07]).

## 4.5 Policy Model

In CQE, confidentiality requirements are always explicitly stated in a set *policy* of formulas in the underlying logic. Such an explicit confidentiality policy is declared by *secadm* independently (that is, without considering the database instance). A confidentiality policy in CQE can have one of two different semantics that were briefly introduced in Section 2.5. We amplify their description as follows:

- **Secrecies:** A confidentiality policy of secrecies is a set of formulas denoted *secr*. Any truth valuation of a *secr* entry is sensitive and has to be hidden from the user. In complete databases this means that the user is neither allowed to assume that the secrecy is *true*, nor that the secrecy is *false*.

- **Potential secrets:** A confidentiality policy of potential secrets is a set of formulas denoted *pot_sec*. For a *pot_sec* entry only its evaluation to *true* is a sensitive fact. That is, if a potential secret is *true* in the database, the user must not notice this; instead, he may assume that it is *false* (if a complete database is considered).

For complete propositional databases, both policy models have been studied extensively (see [BB04a, BB04b]). In incomplete databases, modal potential secrets are handled in order to account for the protection of the value *undefined* (see [Wei08]). In special cases, the specification language of confidentiality policies has to be restricted; for example, in [BL07] only atoms are allowed as potential secrets.

As other inference control approaches (cf. Section 2.5), CQE addresses the implicit need for availability of correct data in the query responses. In query-time censor-based CQE this highly depends on the actual query sequence of the user. Due to the dynamics of the user's queries, availability cannot be globally captured without knowing the query sequence in advance. Instead, there is a local "last-minute distortion strategy": based on the query and knowledge of a user the system returns correct responses as long as possible; this strategy however could lead to more distortions than necessary in the long run.

If availability of some database entries is more important than availability of other entries, the *dbadm* can additionally declare an explicit availability policy *avail* as a set of formulas. In the simplest case, *avail* and *policy* are non-contradictory. Then, all *avail* entries can be handled as "must-know" facts: they are added to *prior* and thus an affirmative response is given if the user queries them.
More flexibility can be gained when alternating a set of explicit confidentiality and availability requirements in a policy hierarchy (see [BBWW07]). In this manner, the *secadm* can specify fine-grained preferences.

## 4.6 Inference Model

Inferences in the CQE framework are precise. Inferences are in particular based on the user's knowledge as modeled by the appropriate current user history $log_i$. The user commands over all knowledge that he can deduce from his current user image $log_i$ by means of logical consequence. In CQE, logical consequence $Cn$ is based on model-theoretic implication in the given logic. A *harmful* inference occurs, if the consequential closure of the user image contains a secret: $\Psi \in Cn(log_i)$ for any $\Psi \in pot\_sec$. Apart from these *log*-dependent inferences, other inferences occur due to the user's system knowledge; for example, knowledge about the confidentiality policy and the protection mechanisms. To account for all those kinds of inferences, for query-time CQE "confidentiality-preserving censors" are defined (see for example [BB04a]). These definitions demand the existence of an alternative database instance based on the sentences in the user history $log_i$, some sentences of the confidentiality policy *policy* and a finite prefix of the query sequence.

## 4.7 Protection Model

CQE implements the two basic mechanisms data restriction on the one hand and data modification on the other hand (cf. Section 2.7):

- *Refusal* is the means of restriction: to a sensitive closed query the database does not return the correct answer (as retrieved by ordinary query evaluation) but instead a special value (denoted `mum` or *refuse*) is returned to indicate that the system refuses to answer.
- *Lying* is the means of data modification: to a sensitive closed query the database returns the negation of the correct answer in a complete database; additionally, in an incomplete database, the value *undefined* may occur as a lie.

As already mentioned, protection for open queries is based on these primitives for closed queries (see [BB07]). If refusal is the only protection mechanism available, it is called "uniform refusal". Analogously, "uniform lying" describes the setting with lying as the only protection mechanisms. The "combined lying and refusal" method uses both mechanisms.

For the different protection mechanisms, *preconditions* have also been identified (and maintained as invariants) that ensure that no $log_i$ leads to harmful inferences. Remarkably, as a precondition for uniform lying and potential secrets the disjunction of all potential secret must not be inferable a priori by the user. This is also the reason why secrecies can only be protected by means of refusal: for lying there always has to be a non-secret alternative left that can be used as a lie.
Whereas for uniform refusal the precondition is simpler, so-called metainferences pose a threat for a confidentiality of potential secrets: if only the sensitive truth

value of a query response is refused, but the insensitive truth value is always returned, the user can conclude that the refusal is equivalent to the sensitive truth value. That is why refusal has to be extended to also cover the insensitive truth value.

### 4.8 Execution Model

In general, CQE implements content-dependent inference control. Yet, CQE can also be used to execute content-independent inference control on schema level. Indeed, for uniform refusal and known policies it has been found out that the refusal decision is independent of the actual evaluation of the query in the database. In [BL07] conditions for a reduction of CQE to access control have been formalized.

When CQE is executed at query-time, a *censor* module has to simulate the inferences of the user. Indeed, (a fragment of) the underlying logic has to be chosen in such a way that implication in it is decidable (if not efficiently computable); that is, implication problems on the current $log_i$ and the present query $\Phi_{i+1}$ (or the corresponding response $eval^*(\Phi_{i+1})(db)$) have to be solved for every $i$. For example, in [BB07] decidability results for the Bernays-Schönfinkel class (first-order logic without function symbols with formulas in prenex normal form and the prefix $\exists^*\forall^*$) are used to show decidability of implication between universally and existentially quantified formulas.

## 5 Contributions and Outline of this Thesis

We believe that the strict logical framework of CQE is able to subsume other inference control approaches based on relational and logical databases – maybe even MLS and statistical databases when adapted accordingly. Probability based approaches can possibly be emulated in CQE with appropriate notions of interpretations and logical implication. One particular advantage of CQE is that it forms a clear and semantically sound basis for both complete and incomplete databases as well as confidentiality policies and user knowledge declarations.

The topic of this thesis is a novel constituent of Controlled Query Evaluation. In contrast to the existing variants of Controlled Query Evaluation – namely, the query-time censor with all its subvariants and the reduction of CQE to traditional access control – this thesis covers a distinct characteristic of the CQE framework: briefly, the goal is to develop and analyze an efficient algorithm that, given the input constraints (that is, the potential secrets and the user's a priori knowledge), preprocesses the original database instance into an "inference-proof" instance which preserves confidentiality of the specified secrets. The user can interact with the inference-proof instance with ordinary uncontrolled query evaluation and thus he does not suffer from performance loss at runtime. Moreover, a main advantage

of this setting is that preservation of availability can effectively be measured in terms of "distortion minimality". While the algorithm (called *pre*CQE) is designed for predicate logic and hence faces the problem of undecidability, restrictions to fragments of the logic and their effects on termination, soundness and completeness are studied comprehensively.

*pre*CQE extends and improves upon related work in several points. We summarize the achievements in brief. The *pre*CQE algorithm

- comprises a unique combination of model generation and distance minimization in an infinite domain.
- handles quantifiers immediately without a need to expand them into ground conjunctions or disjunctions.
- puts only minor restrictions on the syntax of constraint formulas in comparison to other approaches, where restriction to CNF, TGDs or even simpler syntax is required.
- incorporates both addition and deletion of tuples as modification primitives and is hence more general as for example the chase procedure.
- is optimized for complete databases and can thus take advantage of an efficient query evaluation function.
- outputs a complete database instance and does not switch to an incomplete instance; other approaches introduce null values and hence eschew concrete instantiations of variables.
- is proved to be sound and complete; the proof of refutation soundness applies semantic trees to non-clausal formulas in an innovative manner.
- is highly extensible (for example to other distance measures or other policy models).

The particular parts of this thesis make the following contributions:

- Part I
  - At the outset, Section 1 introduced the basic notions of inferences and inference control in databases. It illustrated the substantial shortcomings of access control and the importance of accounting for background knowledge of a user in inference control systems; it set forth a taxonomic categorization of the parameters of inference control systems.
  - Section 2 used this taxonomy to broadly garner information about previous work in the inference control context. This also provided a review of the basic notions for the reader.

- A more detailed view of some inference control systems was procured in Section 3 to put the disparate approaches in perspective.
- Section 4 served the purpose of establishing the background of Controlled Query Evaluation (CQE). The basic terminology in the CQE context was introduced.

- Part II
  - Section 7 sets terminology for the upcoming parts of this thesis. In its subsections, central notions like "DB-interpretation", "DB-satisfiability", "DB-implication", "query evaluation", "inference-proofness" and "distortion minimality" are defined. While the former are borrowed from previous work on CQE, the latter two (inference-proofness and distortion minimality) are novel definitions for confidentiality and availability in the context of this thesis.
  - Section 8 is concerned with finding conditions under which DB-satisfiability of the universal fragment (of pure predicate logic) can be decided. This involves the use of an inductive definition of "allowed formulas" – a well-known subclass of safe formulas that (when used as database queries) ensure finite responses. For the purposes of this section it suffices to consider allowed *universal* formulas; for this class of formulas, we establish the following results by elaborating the proofs based on the structure of the formulas:

    1. the conjunctive normal form of an allowed universal formula is an allowed formula
    2. some simple properties of the conjunctive normal form can be used to identify allowed universal formulas
    3. the negation of an allowed universal formula is an allowed formula

    These results are a step toward the main contribution of this section: for allowed universal formulas, consideration of active domain constants (constants in the input instance and the constraints) is sufficient to achieve inference-proofness. Lastly, we show that also distortion minimality can be ensured with the active domain semantics.
  - Section 9 presents the *pre*CQE algorithm that is devised to search for an inference-proof and distortion minimal solution instance in a "Branch and Bound" approach. In the search process, the algorithm proceeds by marking tuples (possibly in an auxiliary column of the data tables). Appropriate model operators and evaluation functions for marked database instances are defined. It makes abundant use of the evaluation functions to identify constraints that are currently violated and determine their ground instances; for open formulas only evaluation for the positive part

is needed which is ensured to be finite due to the allowed property. Hence the preparation in the previous section now nicely allows for disregarding structural properties of formulas in the algorithm. The *pre*CQE algorithm improves upon previous work by scrutinizing both the possibility to delete as well as to add tuples to the database instance while at the same time minimizing the amount of distortion. It is by far more efficient than finding a solution instance merely by complete enumeration of all active domain tuples. Termination as well as satisfiability soundness and refutation completeness of the algorithm can be established quite straightforwardly by a thorough analysis of the pseudocode. The proof of refutation soundness and satisfiability completeness requires more intricate arguments; to show it we resort to Herbrand's theorem and semantic trees. Lastly, distortion minimality of the solution instance is derived.

- Part III
  - Section 10 shifts the attention to the existential fragment. We argue that Skolemization is not appropriate for our goals of inference-proofness and distortion minimality. Instead, we single out the following two important aspects: an existential formula has the finite model property, and constants outside of the active domain are generic from the point of view of inference-proofness and distortion minimality. Hence we conclude that a solution instance can be found only by considering the active domain and a finite ("invented") set outside of it; yet several alternative instantiations of variables have to be pursued to guarantee distortion minimality. Proofs of termination, soundness and completeness move along very much like in the universal case; still the proof of refutation soundness requires particular attention on how finite invention is used for existentially quantified variables.
  - Section 12 regards a more general type of formulas: tuple-generating dependencies (TGDs) that combine universal and existential quantification. While greatly improving expressiveness they have the disadvantage that difficulties in avoiding infinity axioms are exacerbated. We apply a well-known result for weakly acyclic TGDs to our settings to ensure the applicability of finite invention. Additionally, denial and purely existential formulas may occur. The resulting version of the *pre*CQE algorithm requires a more elaborate presentation than the previous versions and the proofs are also a bit more involved.
- Part IV
  - Section 14 presents a sizable set of extensions that can be pursued further whenever additional requirements arise.

  - Section 15 establishes the connection to related research areas that do not focus on confidentiality but turned out to be helpful resources for *pre*CQE.

- Part V
  - Section 16 amplifies the special case of propositional logic.
  - Section 17 determines how the *pre*CQE problem can be encoded in several versions of the propositional satisfiability problem.
  - Section 18 provides a brief description of a prototypical implementation for propositional logic in which current SAT solving technology is applied.
  - Section 19 documents two sets of test cases. Interestingly, the transformation of the input format into the SAT solver format caused a significant overhead in many cases whereas the performance level of the SAT solver was absolutely favorable.

## 6 Contributions to Published Work

A predecessor of the *pre*CQE algorithm was described in the conference paper [BW06] and then extended to include an availability policy in the invited journal article [BW08b]. The published articles are my original work and the theoretical exposition contained therein laid the foundation for the prototypical implementation that will be covered in Part V of this thesis. The articles are co-authored by my advisor Joachim Biskup. His contribution comprised joint exploration of potential approaches, ongoing discussions, proof-reading and general advisory.
Nevertheless, we point out that this previous work differs from the *pre*CQE approach in this thesis in several points: first and foremost, the algorithms in the published articles are purely designed for propositional clause logic; the published algorithms and *pre*CQE do not even coincide in the propositional case because with *pre*CQE non-clausal input can be handled and only violated constraints are processed. Second, in the articles no formal proofs are given; in particular, in the published propositional case no decidability issues arise and no soundness and completeness results are established. Hence the work contained in this thesis significantly extends and improves upon the algorithms in [BW06] and [BW08b].

# II. Preprocessing for Complete Databases

## Contents



Science is operated according to the judicial system. A theory is assumed to be true if there is enough evidence to prove it 'beyond all reasonable doubt'. On the other hand, mathematics does not rely on evidence from fallible experimentation, but it is built on infallible logic.
– *Simon Singh, Fermat's Enigma*

# 7 Preprocessing for CQE in Complete Databases

In the following sections, we develop an algorithm that constructs a so-called "inference-proof" database instance from a given specification of a problem for Controlled Query Evaluation. The algorithm is called *pre*CQE because it "preprocesses" an instance in the sense that it is executed before a user starts querying the database.

We adopt the terminology and notation of the general CQE framework as defined in Section 4. From a general point of view, the original database $db$, the user's a priori knowledge *prior* and the specification of the confidential information *pot_sec* (in the case of potential secrets), form the input of the transformation routine *pre*CQE that computes a database instance $db'$. The user can issue any sequence of queries to $db'$; these queries are evaluated on $db'$ with ordinary uncontrolled query evaluation (as defined in Definition 7.4). This is possible because *pre*CQE ensures that the answers do not disclose any information that would enable the user to infer confidential information. This property of $db'$ is exactly what we call "inference-proofness". The general thought of this is depicted in Figure 5; a formal definition will be given in Section 7.10.

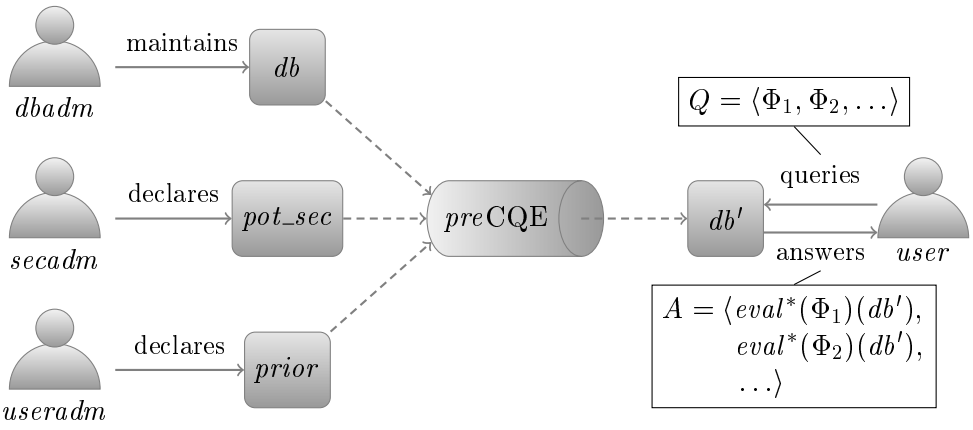


Figure 5: Schematic view of *pre*CQE settings

Before moving on to a detailed description of the system settings, some more words on notational conventions in this thesis are due: As already mentioned previously, the administrators express the input (that is, $db$, *pot_sec* and *prior*) as finite sets of formulas of a logical language $\mathscr{L}$. *pre*CQE is designed for "pure" predicate logic (that is, first-order logic without function symbols except constant symbols); we do not consider a special equality predicate (which would have an infinite extension) or similar arithmetic or "built-in" predicates, either. An incorporation of those would be no obstacle, though; see Section 14.4 for a comment. More formally,

the language $\mathscr{L}$ includes a finite set $\mathcal{P}$ of predicate symbols and an infinite (albeit recursive) set *dom* of constant symbols. $\mathscr{L}$ further includes an infinite set $\mathcal{V}$ of variables, the quantifiers $\exists$ and $\forall$ and the connectives $\neg$ (negation), $\vee$ (disjunction) and $\wedge$ (conjunction); sometimes material implication $\rightarrow$ is used as an abbreviation (for a negation and a disjunction).
In general, $P$ and $Q$ represent arbitrary predicate symbols from $\mathcal{P}$. The function $arity(\cdot)$ returns the arity of a predicate symbol. A single constant from *dom* is denoted $a$, a vector of constants $\vec{a}$; for disambiguation, indices are used for sets of different constants, for example $\{a_1, \ldots, a_n\}$. Similarly, $x$, $y$ and $z$ are variables from $\mathcal{V}$ and $\vec{x}$ is a vector of variables. In some contexts, $\vec{a}$ is used as the set of constants occurring in it, and $\vec{x}$ is used as the set of variables occurring in it. Yet, to have intuitive examples for the predicate and constant symbols in the examples, more figurative denominators like the predicate symbol *Ill* and the constant symbol Mary are chosen. Moreover, for the examples it is also assumed that a position of a predicate symbol only accepts constants from a more specific infinite subset of *dom* – the *sort* of the position – to avoid creation of nonsensical atoms in the examples. The use of sorts is not incorporated in the general theoretic expositions to keep presentation simple and readable; however, sorts can be included without any difficulties.
We write $\Phi$ or $\Psi$ and later on $\Theta$ (on occasion with indices) to denote arbitrary formulas of the language $\mathscr{L}$; these may sometimes be parametrized with a vector of their free variables – for example, $\Phi(\vec{x})$ or $\Phi(x_1, \ldots, x_n)$ – to emphasize that a formula is open. To specifically denote a ground formula (a formula without quantifiers and variables) we write $\phi$ or $\psi$. Moreover, $\Gamma$ and $\gamma$ stand for an open atomic formula (that is, a formula consisting of just one predicate) and a ground atom, respectively. To denote a literal (that is, an atom that might be negated) we write $\Lambda$, respectively $\lambda$ for a ground literal; if the atom inside a literal is seeked, we access it by $|\Lambda|$ and $|\lambda|$. The conjugate of a literal is denoted $\overline{\Lambda}$ or $\overline{\lambda}$. On occasion, $\sigma$ denotes a substitution – in particular, a one-to-one mapping that assigns variables a constant from *dom*.
In all formulas, we assume that variables are standardized apart: no variable is bound by more than one quantifier and no variable occurs bound and free at the same time. We assume that variables are also standardized apart between the formulas; that is, formulas do not have variables in common.
Frequently, we will use "normal form" representations of a formula $\Phi$. More precisely, the following normal forms are needed:

- PNF: the prenex normal form $pnf(\Phi)$, where only quantifiers and variables are allowed to appear in the "prenex" of the formula; the remaining "matrix" of the formula is quantifier-free
- NNF: the negation normal form $nnf(\Phi)$, where negation symbols are only allowed immediately in front of atoms but nowhere else in the formula
- PLNF: the prenex literal normal form (borrowed from Definition 4.2 of [GT91])

$plnf(\Phi)$ which is a combination of PNF and NNF: it has a prenex of quantifiers and variables and a quantifier-free matrix where negation symbols occur immediately in front of atoms.

- CNF: the conjunctive normal form $cnf(\Phi)$, which is a formula in PLNF where additionally the matrix consists of a conjunction of disjunctions of literals; the disjunctions of atoms are often referred to as "clauses".

- DNF: the disjunctive normal form $dnf(\Phi)$, which also is a formula in PLNF but this time with a matrix of a disjunction of conjunctions of literals.

Note that all these normal form representations are equivalent to the original formula $\Phi$. See also the equivalence-preserving rewrite rules in [AHV95] (Figure 5.1 in Chapter 5).
For sets, the operator $card(\cdot)$ returns the cardinality of a set. For sake of brevity, singleton sets are usually written without braces. Other notation is introduced as needed and defined in its context. Based on these notational conventions, we now formalize the basic system settings and the input elements for *pre*CQE with the help of the taxonomy defined in Section 1.

## 7.1 Data Model

*pre*CQE specifically handles the case of complete databases such that the *dbadm* specifies a database schema $DS = \langle \mathcal{P} \rangle$: the schema consists of the set of relation symbols $\mathcal{P}$ and is based on the set *dom* as attribute values. In the complete case, we let $DS$ define $\mathscr{L}$: the set of predicate symbols is $\mathcal{P}$ and the set of constant symbols is *dom*. An analogous definition of a language based on a database schema can be found in [ABK00].

After having defined the database schema $DS$ and thus $\mathcal{P}$, *dom* and $\mathscr{L}$, the administrator *dbadm* has to create a database instance according to the schema. We consider a complete database such that the "input instance" *db* is a finite set of ground atoms of the language $\mathscr{L}$ and a closed world assumption is made to account for the negative part of the database. The database instance represents a special form of interpretation which is called a "DB-interpretation"; this notion is borrowed from [BB07] (Definition 1) – albeit with slight notational changes. It is based on a Herbrand-like interpretation for the constant symbols: we identify the infinite universe (of discourse) $\mathcal{U}$ of semantic individuals of an interpretation $I$ with the set *dom* of constant symbols and let every constant symbol be interpreted by itself; that is, the interpretation involves a function $j$ that maps each constant symbol from *dom* to itself. Furthermore, a DB-interpretation demands that a predicate symbol corresponds to a finite relation; that is, the interpretation involves a function $i$ that maps each predicate symbol of arity $n$ to a finite relation over $dom^n$.

**Definition 7.1 (DB-interpretation (cf. [BB07], Definition 1))**
*A logical structure* $I = \langle \mathcal{U}, i, j \rangle$ *is a* DB-interpretation *for* $\mathscr{L}$ *iff*

1. $\mathcal{U} = dom$
2. $i(P) \subset_{finite} (dom)^{arity(P)}$ *for every* $P$ *in* $\mathcal{P}$
3. $j(a) = a$ *for every* $a$ *in* $dom$

To denote specifically the interpretation that is "induced" by a database instance $db$ we write $I^{db}$. The database instance $db$ represents $I^{db}$ in the following sense: As we do not have function symbols and the constant interpretation is fixed, the only thing that is left to be defined by $db$ is the interpretation for the predicate symbols of $\mathcal{P}$. More precisely, for each ground atom $P(\vec{a})$, $\vec{a} \in i(P)$ if and only if $P(\vec{a}) \in db$. Based on such a DB-interpretation $I$ we can now indeed define satisfaction of a formula of $\mathscr{L}$ under $I$. Starting with ground atoms, we say that $I$ *satisfies* a ground atom $P(\vec{a})$ if and only if $\vec{a} \in i(P)$; alternatively, we say that $P(\vec{a})$ is *satisfied* or *true* in $I$, or that $I$ is a *model* of $P(\vec{a})$; we use the standard model operator $\models$ to denote satisfaction: $I \models P(\vec{a})$. We ignore variable assignments of free variables, as we do not need open formulas in the following. This notion of satisfaction is inductively extended from the ground atoms to arbitrary closed formulas based on the structure of the formulas.
With these definitions of interpretation and satisfaction, we have established the logical foundation for complete databases (to each query a database returns either *true* or *false*) with instances that are finitely representable by a set of ground atoms as their finite positive part. We finally require that the database is static and thus not exposed to updates.

## 7.2 Constraint Model

As explained previously, database constraints are not specified in the schema, but in the user's a priori knowledge *prior*. We require *prior* to be a set of closed formulas (of the language $\mathscr{L}$). More generally in a CQE system, *prior* can be seen as a set of constraints that database instances have to satisfy. We assume that already the input instance $db$ obeys these constraints in *prior*. Thus, we demand that the database interpretation induced by $db$ be a model of *prior* (that is, $I^{db} \models prior$) and thus assigns all formulas in *prior* the truth value *true*.
Based on our Data Model we must make sure that *prior* is indeed satisfiable by a DB-interpretation. We will thus assess sets of formulas in terms of DB-satisfiability:

**Definition 7.2 (DB-satisfiability)**
*A closed formula* $\Phi$ *is* DB-satisfiable *iff there is a DB-interpretation* $I$ *such that* $I \models \Phi$.
*A set* $S$ *of closed formulas is* DB-satisfiable *iff there is a DB-interpretation* $I$ *such that for all formulas* $\Phi \in S$, $I \models \Phi$.

In the following parts of this thesis we are in particular interested in finding syntactical restrictions for a set of constraint formulas (including *prior*) to be DB-satisfiable.

## 7.3 Inference Model

In this thesis, we consider precise inference in first-order logic. We base inference on DB-implication as defined in [BB07] as follows:

---

**Definition 7.3 (DB-implication (cf. [BB07], Definition 2))**
*For a set $S$ of closed formulas of $\mathscr{L}$ and a a closed formula $\Phi$, $S$ implies $\Phi$ by* DB-implication *(written as $S \models_{DB} \Phi$) iff for all DB-interpretations $I$ such that $I \models S$ also $I \models \Phi$.*

---

That is, though finitely represented by *prior* the user's a priori knowledge comprises the closure of *prior* under $\models_{DB}$. Upon querying the database, the a priori knowledge is augmented with query responses; the closure of *prior* and all query responses under $\models_{DB}$ is the knowledge the user disposes of after a number of queries. Note that the response to an open query always consists of an infinite set as both the negative and the positive part of the response are returned explicitly; this is formalized in the Interaction Model.

As a matter of fact, the user is aware of other facts that cannot explicitly be represented in *prior*: he has system knowledge (confer to the description of knowledge types in Section 1) that extends his inference capabilities. These additional inferences and their treatment in *pre*CQE are explained in the User Model.

## 7.4 User Model

In this thesis, as opposed to censor-based CQE, only the user's a priori knowledge *prior* has to be represented; there is no need for a log file that records the history of responses to the user's queries. We make the following assumptions regarding the user's knowledge:

(a) **[Consistent knowledge]** The user knowledge has to be consistent set of formulas because from inconsistent knowledge the user is able to deduce any proposition in the strict logical settings CQE bases on. This implies that *prior* as well as all database responses have to form a consistent set of knowledge.

(b) **[Closed system]** The user's a priori knowledge is fully represented in *prior*. Further, while querying the database, he does not retrieve information outside of the inference-proof database instance $db'$ (at least no information relevant to the protection of the secrets). As discussed previously in Section 2.3, this assumption is usual in inference control systems.

(c) [**Monotonic knowledge**] The user's knowledge is incremental in the sense that he just adds (non-contradictory) information to his knowledge. Notably, he never forgets information once added to his knowledge.

(d) [**A priori protection**] The a priori knowledge does not imply a secret; for otherwise if the user inferred a secret already from *prior*, we could not protect it. That is,

$$\text{for all } \Psi \in \mathit{pot\_sec} : \mathit{prior} \not\models_{DB} \Psi$$

(e) [**Disjunctive closure of secrets**] For the uniform lying mechanism, we have an even stricter precondition (that actually implies precondition (d)): The user must not know (a priori) that the disjunction of the potential secrets is true.

$$\mathit{prior} \not\models_{DB} \mathit{pot\_sec\_disj} \quad (\text{where } \mathit{pot\_sec\_disj} := \bigvee_{\Psi \in \mathit{pot\_sec}} \Psi) \qquad (1)$$

This is already a precondition for the censor-based CQE with uniform lying and there kept as an invariant of the user log. As a justification for this precondition, assume that we have a confidentiality policy $\mathit{pot\_sec} = \{\Phi, \Psi\}$ (for formulas $\Phi$ and $\Psi$ that are both *true* according to *db*) and $\mathit{prior} = \{\Phi \vee \Psi\}$; to the query $\Phi$ the CQE system would return the lie $\neg\Phi$, but this would enable the user to conclude that $\Psi$ was true (although he is not allowed to know this); thus, without requiring Equation 1, the system could run into a situation where even a lie reveals a secret. In *pre*CQE, we need this precondition in order to be able to find an inference-proof instance (with only lying) that can give the user responses that are consistent with *prior*.

(f) [**Sophisticated user**] The user is sophisticated in the sense that he is a rational reasoner and computationally unrestricted in his inference process. Apart from knowledge of the Constraint Model we also assume some user knowledge that is not explicitly represented in *prior*. To wit, the user is aware of the system settings: He knows the Data Model (a complete database without updates), he knows the Policy Model (the specification of the potential secrets without knowing their actual truth values in *db*), and he knows the Protection Model (uniform lying).

## 7.5 Interaction Model

The intention of an inference-proof database instance is that the user (as modeled by *prior*) is allowed to retrieve uncontrolled responses to any query. That is why he can interact with the database with the ordinary query evaluation $\mathit{eval}^*$. This means the *pre*CQE does not incur any query-time performance degradations at all; all computations are shifted into the preprocessing step. Ordinary query evaluation is defined in [BB07] for complete database instances as follows (again, we slightly

altered notation):

**Definition 7.4 (Ordinary query evaluation for complete *db* ([BB07]))**
*An open formula $\Phi$ is evaluated in a complete database instance db according to the following function that returns a set of formulas in $\mathscr{L}$ (where $\wp$ is the power set operator):*

$$eval^*(\Phi(\vec{x})) : DS \to \wp\{\ \Psi \mid \Psi \text{ is a closed formula of } \mathscr{L}\}$$

*with*

$$\begin{aligned} eval^*(\Phi(\vec{x}))(db) &:= \{\Phi(\vec{a}) \mid \vec{a} \subset dom \text{ and } I^{db} \models \Phi(\vec{a})\} \\ &\cup \{\neg\Phi(\vec{a}) \mid \vec{a} \subset dom \text{ and } I^{db} \not\models \Phi(\vec{a})\} \end{aligned}$$

*For a closed formula $\Phi$, ordinary query evaluation reduces to a singleton set (for which curly braces are skipped):*

$$eval^*(\Phi)(db) := \begin{cases} \Phi & \text{if } I^{db} \models \Phi \\ \neg\Phi & \text{else} \end{cases}$$

## 7.6 Policy Model

Due to the restriction of the Protection Model to uniform lying, only a policy semantics of potential secrets is of interest – we already mentioned in Section 4.7 that in the case of secrecies, *any* truth value has to be protected which can only be achieved by refusal. Hence, for *pre*CQE the *dbadm* specifies a set called *pot_sec* of closed formulas. *pre*CQE protects the value of a secret only if it actually evaluates to *true* in the input instance *db*; which means, in the case of a complete database, the user is instead allowed to know the value *false* for a secret.
So far, we do not pose any restrictions on *pot_sec*, apart from the fact that *pot_sec* must not contain (or imply) tautologies: we cannot prevent a user from knowing things that are valid in every model. Furthermore, there is delicate connection between the potential secrets and the user's a priori knowledge: we assume that the user cannot infer the disjunction of all secrets already from *prior*. This condition has been formalized and justified in the User Model.
In the course of this thesis, the potential secrets will be transformed into a set of constraints. We will apply to them the same syntactic conditions for DB-satisfiability that were already mentioned in the Constraint Model for *prior*.

Implicitly, not only confidentiality of the potential secrets has to be guaranteed but also availability of correct data has to be ensured as much as possible. In *pre*CQE this is made precise with a so called "distortion distance" (whose exact definition will be deferred until Section 7.10) and its minimization. This distortion distance makes a crucial difference to censor-based CQE as described in Section 4.5: while

in censor-based CQE availability is captured solely by a heuristics, in *pre*CQE the distortion distance measures availability in concrete figures.
An explicit specification of availability requirements in a separate availability policy *avail* is presented as an extension: its syntax and semantics are defined in Section 14.8.

## 7.7 Protection Model

In contrast to approaches that concentrate on data restriction (upgrading and refusal / denial of access), we design an algorithm for data modification; we adopt the method of uniform lying from the prior work in censor-based CQE. In Section 7.10 we will ultimately define the notion of an "inference-proof" as well as "distortion-minimal" database instance, which corresponds to the definition of "confidentiality-preserving censor" of censor-based CQE. Intuitively, an inference-proof database instance can be seen as an instance that contains cover stories (as described above in Section 2.7.2 for MLS databases) for a single user and his associated clearance level. The novel approach is that we design a fully automated algorithm to obtain the solution database instance and provide the due correctness proofs. [CG01] present an (incomplete) set of rules that are meant to establish a consistent view of a database instance containing cover stories; yet, they neither analyze the complexity of their algorithm nor give a proof of correctness.

## 7.8 Execution Model

*pre*CQE is content-dependent in the sense that it processes the data of an input database instance: the entries of the input instance $db$ are taken into account to find the solution instance $db'$. As long as no updates are considered in the Data Model, *pre*CQE is executed at instantiation-time – directly after a database instance has been created.

## 7.9 Introductory Example

It may be worthwhile to give an example for the CQE settings before continuing the theoretical exposition. We consider a database with medical data consisting of two relations: the one called "Ill" relates a person with a diagnosis, the one called "Treat" relates a person with a medical treatment.

*Example 7.5:* The database administrator specifies the language $\mathscr{L}$ as having the finite set of predicate symbols

$$\mathcal{P} = \{Ill(Name, Diagnosis), Treat(Name, Treatment)\},$$

(recall that $\mathcal{P}$ coincides with the database schema $DS$), as well as the infinite set of constants

$$dom = \{\mathsf{Pete}, \mathsf{Mary}, \mathsf{Lisa}, \mathsf{Paul}, \mathsf{Aids}, \mathsf{Flu}, \mathsf{Cancer}, \mathsf{Myopia}, \mathsf{MedA}, \mathsf{MedB}, \mathsf{MedC}, \ldots\}$$

In this example, we assume the predicates to be sorted and have the following infinite sort subsets of $dom$: $Name = \{\mathsf{Pete}, \mathsf{Mary}, \mathsf{Lisa}, \mathsf{Paul}, \ldots\}$, $Diagnosis = \{\mathsf{Aids}, \mathsf{Flu}, \mathsf{Cancer}, \mathsf{Myopia}, \ldots\}$ and $Treatment = \{\mathsf{MedA}, \mathsf{MedB}, \mathsf{MedC}, \ldots\}$.
As the original database instance $db$, we have

| *Ill* | *Name* | *Diagnosis* |
|---|---|---|
| | Pete | Aids |
| | Mary | Cancer |

| *Treat* | *Name* | *Treatment* |
|---|---|---|
| | Pete | MedA |
| | Mary | MedB |

Ordinary query evaluation on this database instance would for example give the following results:

$$\begin{aligned}
eval^*(\mathit{Treat}(\mathsf{Pete}, \mathsf{MedC}))(db) &= \neg \mathit{Treat}(\mathsf{Pete}, \mathsf{MedC}) \\
eval^*(\exists x\, \mathit{Treat}(x, \mathsf{MedA}))(db) &= \exists x\, \mathit{Treat}(x, \mathsf{MedA}) \\
eval^*(\mathit{Ill}(x, y))(db) &= \{\mathit{Ill}(\mathsf{Pete}, \mathsf{Aids}), \neg \mathit{Ill}(\mathsf{Pete}, \mathsf{Cancer}), \\
&\quad \neg \mathit{Ill}(\mathsf{Pete}, \mathsf{Flu}), \ldots, \\
&\quad \neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids}), \mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer}), \\
&\quad \neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Flu}), \ldots, \\
&\quad \neg \mathit{Ill}(\mathsf{Lisa}, \mathsf{Aids}), \ldots, \\
&\quad \ldots \ \}
\end{aligned}$$

The user's a priori knowledge is declared in $prior$ by the user administrator $useradm$; we assume in this example, that if the user knows the treatment of a patient, then he can narrow down the set of possible diagnoses (these formulas could also represent global integrity constraints of the database):

$$\begin{aligned}
prior &= \{\forall x(\mathit{Treat}(x, \mathsf{MedA}) \rightarrow \mathit{Ill}(x, \mathsf{Aids}) \vee \mathit{Ill}(x, \mathsf{Cancer})), \\
&\quad \forall x(\mathit{Treat}(x, \mathsf{MedB}) \rightarrow \mathit{Ill}(x, \mathsf{Cancer}) \vee \mathit{Ill}(x, \mathsf{Flu}))\}
\end{aligned}$$

The security administrator $secadm$ specifies the potential secrets; in our example, the modeled user should *not* be able to infer that there is a patient with the diagnosis Cancer or the diagnosis Aids.

$$pot\_sec = \{\exists x \mathit{Ill}(x, \mathsf{Aids}), \exists x \mathit{Ill}(x, \mathsf{Cancer})\}$$

Unfortunately, some ordinary query responses enable the user (as modeled by $prior$) to infer a potential secret; for instance:

$$\begin{aligned}
eval^*(\mathit{Ill}(\mathsf{Pete}, \mathsf{Aids}))(db) &\models_{DB} \exists x \mathit{Ill}(x, \mathsf{Aids}) \\
prior \cup eval^*(\exists x(\mathit{Treat}(x, \mathsf{MedB}) \wedge \neg \mathit{Ill}(x, \mathsf{Flu})))(db) &\models_{DB} \exists x \mathit{Ill}(x, \mathsf{Cancer}) \quad \diamondsuit
\end{aligned}$$

## 7.10 Inference-Proofness and Distortion-Minimality

In this section we set forth all the terminology necessary to analyze and avoid the problem of harmful inferences due to ordinary query evaluation. As a main contribution, "inference-proofness" for a database instance with respect to given specifications of a user's a priori knowledge *prior* and potential secrets *pot_sec* is introduced. It serves the purpose of setting terms for the security goal of confidentiality on the strict logical background of CQE. First of all, an inference-proof instance $db'$ has to be consistent with the user's knowledge: from database responses that are inconsistent with *prior*, the user can infer any proposition whatsoever – including all potential secrets. Consistency involves that the interpretation $I^{db'}$ induced by $db'$ is a model of *prior*. We also demand that $I^{db'}$ is not a model of any of the potential secrets. This gives rise to the following definition within the CQE settings relevant to this part of the thesis, particularly the complete database case.

---

**Definition 7.6 (Inference-proofness for complete database instance)**
*Given a set prior and a set pot_sec, a complete database instance $db'$ is called* inference-proof *(with respect to prior and pot_sec) if and only if*

*(i)* $I^{db'} \models prior$

*(ii)* $I^{db'} \not\models \Psi$ *for every* $\Psi \in pot\_sec$

---

We now show that this formal definition coincides with the intention that ordinary query evaluation can be used to respond to the queries of a user without enabling the user to infer secrets. That is, assuming a complete inference-proof database instance $db'$ and the specification of potential secrets *pot_sec* known to a user with a priori knowledge *prior*, no finite response sequence reveals secret information.

---

**Theorem 7.7 (Non-inferability of secrets)**
*For an inference-proof database instance $db'$ and a finite sequence of queries $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$, the following is true:*

$$prior \cup \bigcup_{i=1\ldots n} eval^*(\Phi_i)(db') \not\models_{DB} \Psi, \quad \text{for every } \Psi \in pot\_sec$$

---

*Proof.* From inference-proofness we know that $I^{db'} \models prior$ and from the definition of ordinary evaluation we know that $I^{db'} \models eval^*(\Phi_i)(db')$ (for $i = 1 \ldots n$), such that the union of *prior* and the query responses $eval^*(\Phi_i)(db')$ is a consistent set of formulas with model $I^{db'}$

$$I^{db'} \models prior \cup \bigcup_{i=1\ldots n} eval^*(\Phi_i)(db').$$

Second, inference-proofness gives us $I^{db'} \not\models \Psi$ (for every $\Psi \in pot\_sec$). That is, $I^{db'}$ is a counterexample for the DB-implication ($\models_{DB}$) of any potential secret: while being a model ($\models$) of the left-hand side of the DB-implication, it is not a model of the right-hand side. □

Recall from Section 4.5 that under the assumption of a "known policy" the user is aware of the specification of *pot_sec*. In this case, for complete databases we are left only with the option to evaluate potential secrets to *false* in the inference-proof database:

**Corollary 7.8 (Negated potential secrets for known policy)**
*For a complete database and known pot_sec, Item (ii) in Definition 7.6 is equivalent to*

$$I^{db'} \models \neg\Psi \text{ for every } \Psi \in pot\_sec$$

This is justified by the fact that to a query $\Psi$ (for $\Psi \in pot\_sec$), $db'$ definitely has to respond with $\neg\Psi$ – independent of whether this a lie or a truthful response. In the following we will thus often need the set of negations of potential secrets:

**Definition 7.9 (Set of negated potential secrets)**
*For a set pot_sec of potential secrets the* negated potential secrets *are defined as*

$$Neg(pot\_sec) := \{\neg\Psi | \Psi \in pot\_sec\}$$

Summarizing, for the case of a complete database and a known policy of potential secrets, an inference-proof instance $db'$ is one that induces a model of $prior \cup Neg(pot\_sec)$:

$$I^{db'} \models prior \cup Neg(pot\_sec)$$

From now on we call the set $prior \cup Neg(pot\_sec)$ "constraint set" – written $C$ for short – and speak of its elements as "constraints"; that is:

**Definition 7.10 (Constraint set)**
*For a set prior and a set pot_sec, the* constraint set *is*

$$C := prior \cup Neg(pot\_sec)$$

In the following theorem we utilize the precondition for the a priori knowledge (as given in Item (e) of the User Model in Section 7.4) to show that the corresponding constraint set is DB-satisfiable.

**Theorem 7.11 (Satisfiability of constraint set)**
*Assuming $prior \not\models_{DB} pot\_sec\_disj$, the constraint set $C$ is DB-satisfiable.*

*Proof.* The assumption ensures that *pot_sec_disj* is not a tautology (otherwise it would be implied by *prior*) such that $Neg(pot\_sec)$ is indeed satisfiable. *prior* itself is satisfiable, too, as we require the user knowledge to be consistent. More precisely, applying the definition of DB-implication (Definition 7.3) in contraposition, there is a DB-interpretation $I$ such that $I \models prior$ and $I \not\models pot\_sec\_disj$. But then, for all $\Psi \in pot\_sec$ also $I \not\models \Psi$ holds and thus (for complete $db$) $I \models \neg\Psi$ holds. In other words, $I \models Neg(pot\_sec)$. This ensures that indeed $I \models C$. □

While inference-proofness is intended to ensure confidentiality, "distortion minimality" responds to the issue of availability. We define the "distortion distance" of a database instance $db'$ with respect to the original input instance $db$ as the number of ground atoms that have a different evaluation in $db'$ than in $db$. In essence, we calculate the cardinality of the symmetric difference of the two instances. The symmetric difference is the standard cardinality-based distance, which is widely used in belief revision and related fields; there it is often called "Dalal's distance" (see [Win90]).

---

**Definition 7.12 (Distortion distance)**
*The* distortion distance *of a database instance $db'$ with respect to the input instance $db$ is*

$$db\_dist(db') := card(db \oplus db').$$

---

As we are interested in minimizing the differences between the input instance and the inference-proof solution instance, we also need the notion of "distortion minimality":

---

**Definition 7.13 (Distortion minimality)**
*An inference-proof database instance $db'$ is* distortion-minimal, *iff there is no other inference-proof database instance $db''$ such that*

$$db\_dist(db') > db\_dist(db'').$$

---

We continue Example 7.5 to illustrate the notions of inference-proofness and distortion minimality.

*Example 7.14:* As a preparation for the constraint set $C$, $Neg(pot\_sec)$ is produced:

$$Neg(pot\_sec) \quad = \quad \{\forall x \neg Ill(x, \mathsf{Aids}), \forall x \neg Ill(x, \mathsf{Cancer})\}$$

*prior* is identical to the one in Example 7.5. Now the task is to find a satisfying DB-interpretation for the constraint set $C = prior \cup Neg(pot\_sec)$; in other words, we are looking for a model of $C$ that can be represented by an inference-proof database instance. To start with, we can see that the precondition of Theorem 7.11 is satisfied in this example, that is (after standardizing variables apart in $pot\_sec\_disj$) it holds that $prior \not\models_{DB} \exists x(Ill(x, \mathsf{Aids})) \vee \exists y(Ill(y, \mathsf{Cancer}))$. From Theorem 7.11 we know that $C$ is DB-satisfiable. Indeed we notice that the empty database instance $db'_1 = \emptyset$ is a solution candidate: $I^{db'_1}$ is a model of *prior*, but it's not a model of any of the potential secrets. We can calculate that its distortion distance is $db\_dist(db'_1) = 4$. A second inference-proof instance is the following $db'_2$:

| *Ill* | *Name* | *Diagnosis* |
|---|---|---|
| | Mary | Flu |

| *Treat* | *Name* | *Treatment* |
|---|---|---|
| | Mary | MedB |

$db'_2$ also has distortion distance $db\_dist(db'_2) = 4$. We can see that both instances are distortion-minimal because other inference-proof instances have greater distances. ♢

The definitions of this and the previous section form the basis for the forthcoming examination of the topic. In the following sections, we define different syntactical restrictions for the constraint set $C$ and study their effect on the decidability and the complexity of the problem. Yet, in general we assume formulas to be in PLNF which does not reduce expressiveness and can easily be obtained by pushing negation symbols inward and moving quantifiers to the front of the formula.

## 8 Active Domain Semantics For the Universal Fragment

Given that the satisfiability problem for general first-order logic and also full predicate logic is undecidable (see for example the book on the classical decision problem [BGG01]), it is appropriate to narrow our sights to specific fragments of predicate logic. In this section, we restrict the formulas under consideration in the following manner: we require that the formulas be closed and additionally be in PLNF (as defined in Section 7) where we further assume that the prenex contains only universal quantifiers; we call such formulas "universal formulas":

---

**Definition 8.1 (Universal formulas)**
*A formula* $\Phi = \forall\vec{x}\Psi(\vec{x})$ *is a* universal formula *iff it is a closed formula in PLNF with universal prenex* $\forall\vec{x}$ *and* $\vec{x}$ *consists of all variables occurring free in the matrix* $\Psi(\vec{x})$.

---

We will also extensively use the notion of the "active domain" of a set of formulas and thus introduce it already here:

---

**Definition 8.2 (Active domain)**
*The* active domain *of a set* $S$ *of formulas is the set of all constant symbols that occur in some formula of* $S$*:*

$$adom(S) := \{\ a \mid a\ \in dom\ \text{ and } a \text{ is a constant in a formula of } S\}$$

---

We are specifically interested in the active domain of the original database instance $db$ and the active domain of the constraint set $C$: we simply write $adom$ for $adom(db \cup C)$ if $db$ and $C$ are clear from the context.

When looking for constraints that are DB-satisfiable we see that a restriction to universal formulas is not enough: If arbitrary universal formulas are allowed in $C$, it may be the case that the constraints are not satisfiable by a database instance with a finite positive part. For example, a constraint $\forall xP(x)$ can only be satisfied by positing $P(a)$ for all the infinitely many values $a$ of the infinite domain $dom$; thus, infinitely many database entries would be necessary to satisfy it. We must

agree on some restrictions for the constraints to ensure finiteness of the positive part.
In database theory, the problem of finiteness of query answers has long been investigated; it lead to the definition of "safe" queries, that is, queries that have a finite response based on a fixed domain. On the other hand, "domain-independent" queries are those that evaluate to the same responses for a given database instance even in different domains; that is, evaluation of a domain-independent query depends only on the instance and not on the domain. The reader may readily note that for database instances, domain-independent queries are also safe, because their evaluation can effectively be restricted to the finite active domain of the database instance. As both safety and domain independence of queries are undecidable, syntactic restrictions for queries that define a decidable subclass of safe or domain-independent queries were seeked. Several characterizations of differing complexity have been proposed; see [AHV95] (Chapter 5.3) for an overview.

We pose analogous syntactic restrictions on the formulas in the constraint set $C$. To be able to decide efficiently whether a given formula complies with the restrictions (and also to keep proofs short and readable), we aim at characterizing such restrictions based on the CNF representation of a formula. We resort to the definitions of van Gelder and Topor (see [GT91]) for their "allowed" formulas, though in this section we will only consider universal formulas and thus we do not allow existential quantification and equality for now. Yet, to begin with, we state the general definition of allowed formulas based on the *gen*-relation (as an abbreviation for "generated") between variables and formulas. Intuitively, the allowed property ensures that each subformula that has to be evaluated returns a finite result; in other words, variables that could be bound to infinitely many values when evaluating one subformula are bound to only finitely many values when evaluating another subformula and additionally this second subformula can be evaluated first. More precisely, van Gelder and Topor state that for a variable $x$ and a formula $\Phi(x)$ (where $x$ occurs freely) if $gen(x, \Phi(x))$ holds, $x$ will be bound to constants that appear in the formula $\Phi$ directly or to constants that occur in the database relations whose relation symbols appear in $\Phi$ (see [GT91], page 244): that is, when the formula $\Phi$ is evaluated, $x$ ranges over a subset of $adom$.
Van Gelder and Topor define the *gen*-relation by stating a set of rules and taking the closure of these rules after a finite number of applications. To facilitate readability, we changed the inductive statements of their definition from the rule notation to a definitional notation (with "iffs" and one "if") – which is similar to the inductive definitions in [Dem92].

---

**Definition 8.3 (Allowed formulas / *gen*-relation ([GT91]))**
*An arbitrary formula* $\Phi$ *is* allowed *(or has the* allowed property*) iff*

- *for every* $x$ *that is free in* $\Phi$, $gen(x, \Phi)$ *holds*
- *for every subformula* $\exists x \Psi$ *of* $\Phi$, $gen(x, \Psi)$ *holds*

- *for every subformula* $\forall x\Psi$ *of* $\Phi$, $gen(x, \neg\Psi)$ *holds*

*The relation gen for an atom* $\Phi_1$ *is defined by:*

$$gen(x, \Phi_1) \text{ holds} \quad \text{iff} \quad x \text{ is free in } \Phi_1 \tag{2}$$

*This definition is inductively extended as follows:*

$$\begin{aligned}
gen(x, \neg\Phi_1) \text{ does not hold} \quad &\text{if} \quad \Phi_1 \text{ is an atom} && (3)\\
gen(x, \neg\Phi_1) \text{ holds} \quad &\text{iff} \quad gen(x, pushnot(\neg\Phi_1)) \text{ holds} \\
&\quad \text{and } \Phi_1 \text{ is not an atom} && (4)\\
gen(x, \exists y\Phi_1) \text{ holds} \quad &\text{iff} \quad x \text{ and } y \text{ are distinct and } gen(x, \Phi_1) \text{ holds} && (5)\\
gen(x, \forall y\Phi_1) \text{ holds} \quad &\text{iff} \quad x \text{ and } y \text{ are distinct and } gen(x, \Phi_1) \text{ holds} && (6)\\
gen(x, \Phi_1 \vee \Phi_2) \text{ holds} \quad &\text{iff} \quad \text{both } gen(x, \Phi_1) \text{ and } gen(x, \Phi_2) \text{ hold} && (7)\\
gen(x, \Phi_1 \wedge \Phi_2) \text{ holds} \quad &\text{iff} \quad gen(x, \Phi_1) \text{ or } gen(x, \Phi_2) \text{ holds} && (8)
\end{aligned}$$

*where again the function pushnot is defined for the following formulas:*

$$\begin{aligned}
pushnot(\neg(\Phi_1 \wedge \Phi_2)) &= (\neg\Phi_1) \vee (\neg\Phi_2)\\
pushnot(\neg(\Phi_1 \vee \Phi_2)) &= (\neg\Phi_1) \wedge (\neg\Phi_2)\\
pushnot(\neg\exists x\Phi_1) &= \forall x\neg\Phi_1\\
pushnot(\neg\forall x\Phi_1) &= \exists x\neg\Phi_1\\
pushnot(\neg\neg\Phi_1) &= \Phi_1
\end{aligned}$$

---

The reader will readily note that ground formulas (that is, formulas without variables) always have the allowed property by this definition. Now we apply this general definition to our special case of universal formulas. When restricted to universal formulas, Definition 8.3 reduces to the following Definition 8.4; note that all quantified variables are distinct and thus their quantifiers can be skipped due to Rules 4 and 5 and the *pushnot* definition for $\neg\forall$:

---

**Definition 8.4 (Allowed universal formulas)**
*Let* $\Phi = \forall\vec{x}\Psi(\vec{x})$ *be a universal formula.* $\Phi$ *is an* allowed universal *formula iff for every* $x \in \vec{x}$, $gen(x, \neg\Psi(\vec{x}))$ *holds.*

---

Consider $\Phi = \forall x(Treat(x, \mathsf{MedA}) \rightarrow Ill(x, \mathsf{Aids}))$ as a tiny example formula: it is an allowed universal formula because, when replacing the connective $\rightarrow$ (with a $\neg$ and an $\vee$ symbol), we have $\Phi = \forall x(\neg Treat(x, \mathsf{MedA}) \vee Ill(x, \mathsf{Aids}))$. We can now verify that for $x$ the *gen*-relation $gen(x, \neg(\neg Treat(x, \mathsf{MedA}) \vee Ill(x, \mathsf{Aids})))$ indeed holds; after two applications of *pushnot*, we are left with $gen(x, Treat(x, \mathsf{MedA}) \wedge \neg Ill(x, \mathsf{Aids}))$ and this holds because $gen(x, Treat(x, \mathsf{MedA}))$ holds: the variable $x$ occurs freely in the ground atom $Treat(x, \mathsf{MedA})$.

While van Gelder and Topor show that the "evaluable" property (originally by Demolombe [Dem92]) of a formula can be efficiently verified with the help of its CNF representation (cf. [GT91], Theorem 7.2), there is no such characterization for allowed formulas in general. The problem with arbitrary allowed formulas is that moving the quantifiers into the prenex (that is transforming the formula into PLNF) can destroy the allowed property; for example, a formula like $\Phi \vee \exists x\, \Psi(x)$ (where $x$ does not occur in $\Phi$) is allowed but $\exists x\, (\Phi \vee \Psi(x))$ is not.
In the following we show that for our allowed *universal* formulas (that is, allowed formulas that are already in PLNF with universal prenex) such a CNF-based verification for the allowed property is possible. While van Gelder and Topor concentrate on the evaluable property and the corresponding *con*(strained)-relation, we extend their results (cf. [GT91], Theorem 6.6) to our case of allowed universal formula and elaborate the proofs for the *gen*-relation. As a preliminary step, we show that the CNF representation of an allowed universal formula is allowed and vice versa. Note that this is not the case for arbitrary allowed or evaluable formulas.

**Lemma 8.5 (Allowed property of CNF)**
*A universal formula $\Phi = \forall \vec{x}\, \Psi(\vec{x})$ is allowed iff its CNF representation $cnf(\Phi)$ is allowed.*

*Proof.* The CNF representation of $\Phi$ is the CNF representation of $\Psi$ and the unaltered prenex: $cnf(\Phi) = \forall \vec{x}\, cnf(\Psi(\vec{x}))$ by definition of CNF formulas. Now, when transforming $\Psi$ into CNF, only the distributive law of "pushing ors" is necessary:

$$\Psi_1 \vee (\Psi_2 \wedge \Psi_3) \quad \equiv \quad (\Psi_1 \vee \Psi_2) \wedge (\Psi_1 \vee \Psi_3)$$

We have to show that $gen(x, \neg\Psi(\vec{x}))$ holds if and only if $gen(x, \neg cnf(\Psi(\vec{x})))$ holds (for every $x \in \vec{x}$); that is, in essence we have to show that the application of the *pushnot*-function as well as the application of the distributive law do not interfere with the allowed property.
First, when starting with the left side of the distributive law, we see that:

$$\begin{aligned}
& gen(x, \neg(\Psi_1 \vee (\Psi_2 \wedge \Psi_3))) \text{ holds} \\
\iff\ & gen(x, pushnot(\neg(\Psi_1 \vee (\Psi_2 \wedge \Psi_3)))) \text{ holds} \\
\iff\ & gen(x, \neg\Psi_1 \wedge \neg(\Psi_2 \wedge \Psi_3)) \text{ holds} \\
\iff\ & gen(x, \neg\Psi_1) \text{ holds or } gen(x, \neg(\Psi_2 \wedge \Psi_3)) \text{ holds} \\
\iff\ & gen(x, \neg\Psi_1) \text{ holds or } gen(x, pushnot(\neg(\Psi_2 \wedge \Psi_3))) \text{ holds} \\
\iff\ & gen(x, \neg\Psi_1) \text{ holds or } gen(x, \neg\Psi_2 \vee \neg\Psi_3) \text{ holds} \\
\iff\ & gen(x, \neg\Psi_1) \text{ holds or both } gen(x, \neg\Psi_2) \text{ and } gen(x, \neg\Psi_3) \text{ hold}
\end{aligned}$$

Second, when starting from the right side, we have:

$$\begin{aligned}
& gen(x, \neg((\Psi_1 \vee \Psi_2) \wedge (\Psi_1 \vee \Psi_3))) \text{ holds} \\
\iff\ & gen(x, pushnot(\neg((\Psi_1 \vee \Psi_2) \wedge (\Psi_1 \vee \Psi_3)))) \text{ holds}
\end{aligned}$$

$\iff$ $gen(x, \neg(\Psi_1 \vee \Psi_2) \vee \neg(\Psi_1 \vee \Psi_3))$ holds
$\iff$ both $gen(x, \neg(\Psi_1 \vee \Psi_2))$ and $gen(x, \neg(\Psi_1 \vee \Psi_3))$ hold
$\iff$ both $gen(x, pushnot(\neg(\Psi_1 \vee \Psi_2)))$ and $gen(x, pushnot(\neg(\Psi_1 \vee \Psi_3)))$ hold
$\iff$ both $gen(x, \neg\Psi_1 \wedge \neg\Psi_2)$ and $gen(x, \neg\Psi_1 \wedge \neg\Psi_3)$ hold
$\iff$ at least one of ($gen(x, \neg\Psi_1)$ and $gen(x, \neg\Psi_2)$) holds as well as
at least one of ($gen(x, \neg\Psi_1)$ and $gen(x, \neg\Psi_3)$) holds
$\iff$ $gen(x, \neg\Psi_1)$ holds or both $gen(x, \neg\Psi_2)$ and $gen(x, \neg\Psi_3)$ hold

Thus, we reach at the same conclusion as in the first part from the left side. From these equivalences the claim of the lemma follows. □

We now move on to our main result for the recognition of allowed universal formulas: the allowed property of a universal formula can be verified by means of syntactical characterization of its CNF representation. This result – the following Theorem 8.6 – will be of great help for the forthcoming considerations of inference-proof database instances.

**Theorem 8.6 (CNF-based verification of allowed property)**
*A universal formula $\Phi = \forall\vec{x}\,\Psi(\vec{x})$ is allowed iff every variable $x \in \vec{x}$ occurs in every conjunct of $cnf(\Phi)$ in a negative literal ($x$ may or may not occur in other negative or positive literals in a conjunct).*

*Proof.* We start with the "left-to-right" direction. From Lemma 8.5 we know that $\forall\vec{x}\,\Psi(\vec{x})$ is allowed if and only if $\forall\vec{x}\,cnf(\Psi(\vec{x}))$ is allowed. From the allowed property we know that $gen(x, \neg cnf(\Psi(\vec{x})))$ has to hold for any $x \in \vec{x}$. We thus take a closer look at the structure of the CNF representation. We know that $cnf(\Psi(\vec{x}))$ consists of conjuncts $\Pi_i$:

$$cnf(\Psi(\vec{x})) = \Pi_1 \wedge \ldots \wedge \Pi_m$$

and each conjunct $\Pi_i$ consists of a disjunction of literals $\Lambda_{ij}$:

$$\Pi_i = \Lambda_{i1} \vee \ldots \vee \Lambda_{in_i}.$$

Now, a transformation into NNF can be achieved by pushing the negation symbols inwards; $nnf(\neg cnf(\Psi(\vec{x})))$ consists of disjuncts $\Delta_i$:

$$nnf(\neg cnf(\Psi(\vec{x}))) = \Delta_1 \vee \ldots \vee \Delta_m$$

where each $\Delta_i$ is a conjunction of the conjugates of the literals $\Lambda_{ij}$:

$$\Delta_i = \overline{\Lambda_{i1}} \wedge \ldots \wedge \overline{\Lambda_{in_i}}.$$

We now have to make sure that pushing negation does not influence the *gen*-property. This is basically an application of the definitions:

$$gen(x, \neg(\Psi_1 \wedge \Psi_2)) \text{ holds} \iff gen(x, pushnot(\neg(\Psi_1 \wedge \Psi_2))) \text{ holds}$$

$$
\begin{aligned}
& \iff gen(x, \neg\Psi_1 \vee \neg\Psi_2) \text{ holds} \\
& \iff \text{both } gen(x, \neg\Psi_1) \text{ and } gen(x, \neg\Psi_2) \text{ hold} \\
gen(x, \neg(\Psi_1 \vee \Psi_2)) \text{ holds} & \iff gen(x, pushnot(\neg(\Psi_1 \vee \Psi_2))) \text{ holds} \\
& \iff gen(x, \neg\Psi_1 \wedge \neg\Psi_2) \text{ holds} \\
& \iff gen(x, \neg\Psi_1) \text{ or } gen(x, \neg\Psi_2) \text{ holds} \\
gen(x, \neg\neg\Psi) \text{ holds} & \iff gen(x, pushnot(\neg\neg\Psi)) \text{ holds} \\
& \iff gen(x, \Psi) \text{ holds}
\end{aligned}
$$

From these equivalences we know that $gen(x, nnf(\neg cnf(\Psi(\vec{x}))))$ holds if and only if for every $\Delta_i$, $gen(x, \Delta_i)$ holds. And this is equivalent to the fact that (for every $\Delta_i$) there is a $j_i$ such that for the literal $\overline{\Lambda_{ij_i}}$ indeed $gen(x, \overline{\Lambda_{ij_i}})$ holds. This again means that $\overline{\Lambda_{ij_i}}$ has to be an atom where $x$ occurs freely. From this we conclude $\Lambda_{ij_i}$ has to be a negative literal and such a literal has to exist for every conjunct $\Pi_i$ in the CNF representation. As all these deductions are based on equivalences, the opposite direction follows, too. □

Our tiny example formula $\Phi = \forall x(Treat(x, \mathsf{MedA}) \rightarrow Ill(x, \mathsf{Aids}))$ is already in CNF – to wit, $cnf(\Phi) = \forall x(\neg Treat(x, \mathsf{MedA}) \vee Ill(x, \mathsf{Aids}))$. In the single conjunct of its matrix, $\neg Treat(x, \mathsf{MedA}) \vee Ill(x, \mathsf{Aids})$, $x$ occurs not only in the positive literal $Ill(x, \mathsf{Aids})$ but also in the negative literal $\neg Treat(x, \mathsf{MedA})$ which makes it allowed.

Note that Theorem 8.6 provides an elegant mechanism to recognize allowed universal formulas: instead of checking the *gen*-relation inductively for every subformula, it is possible to recognize allowed universal formulas by their CNF representation. Yet, the *pre*CQE algorithm (as will be presented in Section 9) does not depend on CNF input; on the contrary, it accepts allowed universal formulas of arbitrary structure. We consider this a great advantage of *pre*CQE over other CNF-dependent first-order procedures; further comments on this will be given in Section 15.
Internally, *pre*CQE transforms an allowed universal input formula $\Phi$ by

1. negating the formula: $\neg\Phi$
2. pushing the negation inwards such that the formula is in PLNF: $plnf(\neg\Phi)$
3. dropping the prenex of the formula (including all the quantifiers) such that only its matrix is left where all variables now occur free: $droppprenex(plnf(\neg\Phi))$ (we implicitly introduce the function *droppprenex* here that effectively returns the matrix of its input formula)

While the technical details will be filled in in Section 9, we anticipate here a novel theoretical result for negations of allowed universal formulas that are transformed into PLNF and freed of their prenex – that is, quantifier-free formulas of the form $droppprenex(plnf(\neg\Phi))$.

---

**Lemma 8.7 (Negations of allowed universal formulas)**
*If $\Phi$ is an allowed universal formula, $droppprenex(plnf(\neg\Phi))$ is an allowed formula.*

---

*Proof.* As $\Phi$ is allowed universal of the form $\Phi = \forall \vec{x}\, \Psi(\vec{x})$, the PLNF of its negation has the form $plnf(\neg\Phi) = \exists \vec{x}\, nnf(\neg\Psi(\vec{x}))$. We drop the prenex such that $droppprenex(plnf(\neg\Phi)) = nnf(\neg\Psi(\vec{x}))$. Yet, we can easily see that $nnf(\neg\Phi)$ is an allowed formula (by Definition 8.3): Due to $\Phi$ being allowed universal, we can be sure that $gen(x, \neg\Psi(\vec{x}))$ holds. What is left to show is that pushing the negation inwards does preserve the allowed property. That is, we have to show that $gen(x, nnf(\neg\Psi(\vec{x})))$ holds for every $x \in \vec{x}$; as already shown in Theorem 8.6, this is the case because the *gen*-relation is actually defined (Definition 8.3) by pushing $\neg$ inwards and thus the allowed property carries over to $nnf(\neg\Psi(\vec{x}))$. □

Lastly, we briefly motivate why the treatment of allowed universal formulas and their negations is sensible:

*Remark 8.8 (Evaluation properties of allowed formulas):* The transformed formula $droppprenex(plnf(\neg\Phi))$ inherits the nice properties of allowed formulas; to wit, when an allowed formula is evaluated in a database instance,

- all results (including intermediate results) will be finite such that the evaluation can effectively be computed; van Gelder and Topor show this by devising a procedure that transforms allowed formulas into a "relational algebra normal form" RANF.
- all free variables are mapped to a subset of *adom* such that the evaluation result consists of a finite set of ground formulas with *adom* constants. ♢

The reader may observe, that our properties are stricter than the properties for range-restricted formulas (see [Nic82] or page 102 of [AHV95]) or evaluable formulas (see [GT91]): in range-restricted and evaluable formulas, quantified variables are allowed to be absent in some conjuncts of the CNF-formula, while in our definition they have to occur in every conjunct owed to the fact that quantified variables become free after an application of *droppprenex*. Yet, this obvious disadvantage is compensated by our treatment of a constraint *set* (instead of analyzing a single query alone).

We now want to show that for a constraint set $C$ that contains only allowed universal formulas, effectively the "active domain semantics" (see [AHV95]) can be employed. We allege that we only have to consider values from *adom* when looking for an inference-proof instance $db'$. We denote by $db'_{|adom}$ the set of ground atoms such that $db'_{|adom} \subseteq db'$ where all ground atoms of $db'$ that contain constants not in the active domain *adom* are removed. More formally:

---

**Definition 8.9 (Restriction to active domain)**
*The* restriction to *adom of a database instance* $db'$ *is*

$$db'_{|adom} := \{\gamma \mid \gamma \in db' \quad \textit{and all constants in } \gamma \textit{ are in adom } \}$$

---

The main result of this section in terms of inference-proofness is presented next: The restriction to *adom* of an inference-proof database instance is also an inference-proof database instance.

**Theorem 8.10 (Active domain semantics for allowed constraints)**
*For a set $C = prior \cup Neg(pot\_sec)$ of allowed universal formulas, and an inference-proof database instance $db'$, $db'_{|adom}$ is an inference-proof database instance.*

*Proof.* Due to $db'_{|adom} \subseteq db'$, $db'_{|adom}$ is clearly a database instance with a finite positive part that complies with the database schema.
To show that $db'_{|adom}$ also satisfies Definition 7.6, we first of all prove that if $I^{db'} \models C$ then also $I^{db'_{|adom}} \models C$ by contradiction. Thus, assume that $I^{db'_{|adom}} \not\models C$. This means that there is a constraint $\Phi$ in $C$ that is not satisfied in $db'_{|adom}$; that is, $I^{db'_{|adom}} \not\models \Phi$. $\Phi$ cannot be a ground formula because $db'$ and $db'_{|adom}$ coincide on ground atoms $\gamma$ with only *adom* constants by definition: $I^{db'} \models \gamma$ if and only if $I^{db'_{|adom}} \models \gamma$. Thus, $\Phi$ is an allowed universal formula of the form $\Phi = \forall \vec{x} \Psi(\vec{x})$. This in turn means that there is a tuple $\vec{a}$ of constants for which the ground formula $\Psi(\vec{a})$ is not satisfied in $db'_{|adom}$, that is, $I^{db'_{|adom}} \not\models \Psi(\vec{a})$.

Due to equivalence, also the CNF representation is not satisfied: $I^{db'_{|adom}} \not\models cnf(\Psi(\vec{a}))$, where $cnf(\Psi(\vec{a}))$ consists of the (ground) conjuncts $\pi_i$: $cnf(\Psi(\vec{a})) = \pi_1 \wedge \ldots \wedge \pi_m$ and the conjunct $\pi_i$ consists of literals: $\pi_i = \lambda_{i1} \vee \ldots \vee \lambda_{in_i}$. Thus we know that there is one $\pi_i$ for which $I^{db'_{|adom}} \not\models \pi_i$ and accordingly $I^{db'_{|adom}} \not\models \lambda_{ij}$ for all literals $\lambda_{ij}$ in $\pi_i$. We now differentiate the following cases:

1. $\lambda_{ij}$ contains only *adom* constants
   (a) if $\lambda_{ij}$ is a negative literal, then $|\lambda_{ij}| \in db'_{|adom}$; but because $db'_{|adom} \subseteq db'$ (by definition) then also $|\lambda_{ij}| \in db'$ and $I^{db'} \not\models \lambda_{ij}$
   (b) if $\lambda_{ij}$ is a positive literal (that is, a ground atom), then also $I^{db'} \not\models \lambda_{ij}$; because otherwise $I^{db'} \models \lambda_{ij}$ if and only if $\lambda_{ij} \in db'$ and hence also $\lambda_{ij} \in db'_{|adom}$ but then $I^{db'_{|adom}} \models \lambda_{ij}$ would hold – a contradiction
2. $\lambda_{ij}$ contains a constant $a \in \vec{a}$ which is not in *adom*
   (a) $\lambda_{ij}$ cannot be a negative literal because otherwise $|\lambda_{ij}| \in db'_{|adom}$ but $db'_{|adom}$ does not contain any non-*adom* constants by definition
   (b) if $\lambda_{ij}$ is a positive literal, the allowed property (of $\Phi$) demands that there be a negative literal $\lambda_i(a)$ which contains the non-*adom* constant $a$; yet, the Case 2a) impedes the existence of such a negative literal $\lambda_i(a)$

From these cases we conclude that $I^{db'}$ does not satisfy the conjunct $\pi_i$ which contradicts our assumption that $db'$ is an inference-proof database instance. That

is, we can be sure that $I^{db'_{|adom}} \models C$ and use Corollary 7.8 to conclude that $db'_{|adom}$ is indeed inference-proof. □

We illustrate the notions of the allowed property and the active domain semantics with the help of Example 7.14.

*Example 8.11:* Note that Example 7.14 illustrates the settings for allowed universal constraints: When replacing the material implication "→" in $C$, the constraints are already in CNF:

$$\begin{aligned} C \;=\; \{&\forall x(\neg \mathit{Treat}(x, \mathsf{MedA}) \vee \mathit{Ill}(x, \mathsf{Aids}) \vee \mathit{Ill}(x, \mathsf{Cancer})),\\ &\forall x(\neg \mathit{Treat}(x, \mathsf{MedB}) \vee \mathit{Ill}(x, \mathsf{Cancer}) \vee \mathit{Ill}(x, \mathsf{Flu})),\\ &\forall x \neg \mathit{Ill}(x, \mathsf{Aids}),\\ &\forall x \neg \mathit{Ill}(x, \mathsf{Cancer})\} \end{aligned}$$

They have the allowed property because the variable $x$ appears in every conjunct in a negated literal. We also see that both $db'_1$ and $db'_2$ do not contain any non-*adom* constants. ♢

We conclude from Theorems 8.10 and 7.11 that under the CQE precondition for uniform lying (as already used in Theorem 7.11) for allowed universal constraint formulas there is at least one inference-proof instance in which only *adom* constants occur:

---

**Corollary 8.12 (Inference-proof instances with active domain semantics)** *Given a constraint set $C$ of allowed universal formulas and assuming prior $\not\models_{DB}$ pot_sec_disj, there exists an inference-proof solution instance (with respect to prior and pot_sec) that contains only adom constants.*

---

In Example 7.14 we have already verified that this precondition for the example sets *prior* and *pot_sec* holds.

To be sure that the restriction to the active domain is a good choice, we must now show that the restricted instance $db'_{|adom}$ has at least as good a distortion distance as the unrestricted instance $db'$. This is done in the following theorem.

---

**Theorem 8.13 (Distortion minimization with active domain semantics)** *For a set $C$ of allowed universal constraints, the distortion distance of $db'_{|adom}$ is not greater than the one of $db'$; that is,*

$$db\_dist(db'_{|adom}) \leq db\_dist(db').$$

---

*Proof.* First of all note that $db \setminus db'$ is the same as $db \setminus db'_{|adom}$: both the restricted and the unrestricted instance only differ on ground atoms that are not contained in $db$ because the ground atoms in $db$ only contain values from *adom*. On the other hand, we can see that $(db'_{|adom} \setminus db) \subseteq (db' \setminus db)$ because $db'_{|adom} \subseteq db'$. In conclusion, we have $card(db \oplus db'_{|adom}) \leq card(db \oplus db')$. □

With the active domain semantics we know that we only have to check instantiations of database predicates with constants in *adom* to find an inference-proof database instance. That is, we can establish an upper bound for the distortion distance as the number of all possible ground atoms with *adom* constants.

**Corollary 8.14 (Upper bound of distortion distance)**
*For a set $C$ of allowed universal constraints, the distortion distance for $db'_{|adom}$ (for any inference-proof instance $db'$) can be bounded as follows*

$$db\_dist(db'_{|adom}) \leq \sum_{\substack{P \in \mathcal{P} \\ P \text{ occurs in } C}} card(adom)^{arity(P)}$$

To illustrate the upper bound, we continue the running example.

*Example 8.15:* Continuing Example 8.11, we can now calculate an upper bound value for the distortion distance; we identify as the active domain of $db$ and $C$ the set $\{\mathsf{Pete}, \mathsf{Mary}, \mathsf{Aids}, \mathsf{Cancer}, \mathsf{Flu}, \mathsf{MedA}, \mathsf{MedB}\}$ and thus $card(adom) = 7$. Yet, for the sorted predicate symbols we consider in this example we can restrict the general active domain to the active domains of the sorts; in this case, the upper bound can be calculated as

$$\begin{aligned} db\_dist(db') &\leq card(adom \cap Name) \cdot card(adom \cap Diagnosis) \\ &\quad + card(adom \cap Name) \cdot card(adom \cap Treatment) \\ &= 6 + 4 = 10 \end{aligned}$$

♢

## 9 *pre*CQE for Allowed Universal Constraints

Based on the theoretical background of the previous section, we present an algorithm for the case of allowed universal constraints as defined in Section 8 – recall that this means that the formulas are closed and in prenex literal normal form (PLNF) with only universally quantified variables that additionally have the allowed property; we have already remarked that this especially includes ground formulas in NNF. We find the solution instance $db'$ by searching along branches in a binary "search tree". Some leaf in the search tree is then chosen as an instruction how to transform the original instance $db$ into the solution instance $db'$. The tree is traversed in depth-first search manner: one branch at a time is processed.
Branches are constructed by a "splitting" operation that creates two child nodes. It assigns a ground atom the value *false* in the left child node; in other words, the ground atom is either removed from the database instance (if it was included in $db$) or left out of the instance (if it was not included). In the right child node, the ground atom is assigned *true*; in other words, the ground atom is added to or kept

in the database instance.
Yet, it may occur that there is actually no need for a splitting operation: only one of the two truth values (either *true* or *false*) for a ground atom promises the opportunity to satisfy a constraint. Then no new nodes are created but instead the unique truth value is assigned to the ground atom in the current node. In particular, this strategy applies to "unit constraints" – the ones containing only a single literal; thus, the truth valuations of all the ground atoms affected by a unit constraint are set to the truth value that is predetermined by the constraint. Before starting with the technical details we give an example of a *pre*CQE input in Example 9.1 and show its associated search tree in Figure 6.

*Example 9.1:* We continue Example 8.15. That is, we have the input instance

$$db \;=\; \{\mathit{Ill}(\mathsf{Pete},\mathsf{Aids}),\, \mathit{Ill}(\mathsf{Mary},\mathsf{Cancer}),\, \mathit{Treat}(\mathsf{Pete},\mathsf{MedA}),\, \mathit{Treat}(\mathsf{Mary},\mathsf{MedB})\}$$

and the constraint set

$$\begin{aligned} C \;=\; \{&\forall x(\neg \mathit{Treat}(x,\mathsf{MedA}) \vee \mathit{Ill}(x,\mathsf{Aids}) \vee \mathit{Ill}(x,\mathsf{Cancer})),\\ &\forall x(\neg \mathit{Treat}(x,\mathsf{MedB}) \vee \mathit{Ill}(x,\mathsf{Cancer}) \vee \mathit{Ill}(x,\mathsf{Flu})),\\ &\forall x \neg \mathit{Ill}(x,\mathsf{Aids}),\\ &\forall x \neg \mathit{Ill}(x,\mathsf{Cancer})\} \end{aligned}$$

Figure 6 shows how a solution for this input could be found. At the outset, we satisfy the two unit constraints $\forall x \neg \mathit{Ill}(x,\mathsf{Aids})$ and $\forall x \neg \mathit{Ill}(x,\mathsf{Cancer})$: the removal of the two entries $\mathit{Ill}(\mathsf{Pete},\mathsf{Aids})$ and $\mathit{Ill}(\mathsf{Mary},\mathsf{Cancer})$ in the root node is unequivocal. The first splitting step then corresponds to the decision whether to keep the atom $\mathit{Treat}(\mathsf{Pete},\mathsf{MedA})$ or not. The splitting gives rise to a further splitting step in the left child node $v_1$ where ultimately in node $v_2$ the solution instance $db'_1$ is found.
Node $v_3$ also yields a solution instance – $db'_2$ – in the following manner: After the splitting operation, we furthermore have to add $\mathit{Ill}(\mathsf{Mary},\mathsf{Flu})$ (in order to satisfy the second constraint formula). Yet, in this situation we do not have to split anymore because the other atoms in the formula (in this case, $\mathit{Treat}(\mathsf{Mary},\mathsf{MedB})$ and $\mathit{Ill}(\mathsf{Mary},\mathsf{Cancer})$) have already been treated.
The root's right child node $v_4$ fails (that is, its branch is "pruned"): With the splitting operation we decided to keep the database entry $\mathit{Treat}(x,\mathsf{MedA})$ but then the atom $\mathit{Ill}(\mathsf{Pete},\mathsf{Cancer})$ has to be added (in order to satisfy the first constraint formula). This indeed constitutes an unresolvable conflict with the unit constraint $\forall x \neg \mathit{Ill}(x,\mathsf{Cancer})$.
Note that this procedure results exactly in the solution candidates $db'_1$ and $db'_2$ from Example 7.14. ♢

We now move on to a technical description of the *pre*CQE procedure. *pre*CQE assigns truth values by marking ground atoms in the original database instance: a special value is temporarily appended to the ground atom that designates the

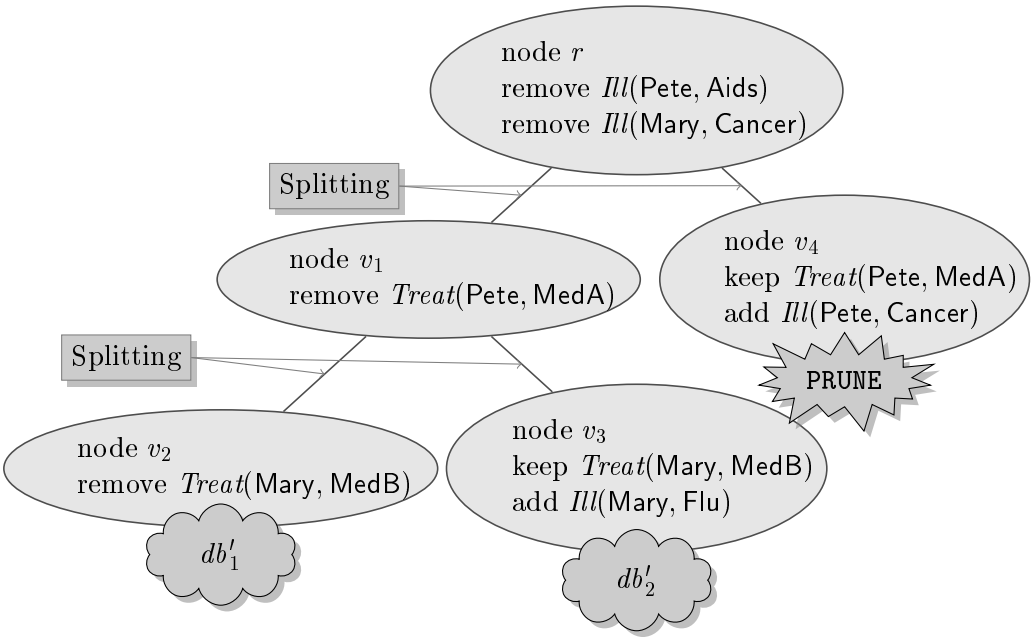


Figure 6: Example for active domain semantics

intended truth valuation for the ground atom in the resulting instance $db'$. In this way, the markers represent an interpretation for the predicate symbols in the language $\mathscr{L}$. In some cases, a ground atom has to be added to the database instance before its marker is set. In what follows, we will speak of a "marked database instance" $db_v$ and mean by it the database instance with marked ground atoms in a node $v$ of the search tree. More precisely, a marked database instance $db_v$ is a *finite* set of ground atoms some of which are marked.
The following markers are used:

1. "keep" (`k`) signifying that the according ground atom of $db$ should be retained in $db'$
2. "add" (`a`) signifying that the according ground atom is not contained in $db$ but should be added to $db'$
3. "remove" (`r`) signifying that the according ground atom of $db$ should not occur in $db'$
4. "leave" (`l`) signifying that the according ground atom is not contained in $db$ and should also be left out of $db'$

As a more general notation, we use the function $marker_v(\gamma)$ to access and set the marker for a ground atom $\gamma$ in a marked database instance $db_v$. We can for example write $marker_v(P(\vec{a})) :=$ `a` to set a marker `a` for a ground atom $P(\vec{a})$; its meaning is that $P(\vec{a})$ was *false* in the input instance $db$ (due to the closed world

assumption for complete databases), but that it has to be *true* in the solution instance $db'$. To denote that no marker for $\gamma$ is set (that is, $\gamma$ is "unmarked"), we write $marker_v(\gamma) \notin \{\mathtt{k}, \mathtt{a}, \mathtt{r}, \mathtt{l}\}$; this applies to both ground atoms $\gamma$ contained in $db_v$ as well as ground atoms $\gamma$ not contained in $db_v$. We extend the *marker*-function to also be applicable to ground literals; that is, for a ground literal $\lambda$, $marker_v(\lambda)$ returns the marker $marker_v(\gamma)$ of the ground atom $\gamma = |\lambda|$.
Markers can be implemented by an additional attribute in each relation (that is, an additional column in the data tables); values for the markers are inserted in the column when descending the search tree. This implementation also facilitates backtracking in the search tree as this now corresponds to a deletion of some marker values. A slightly different notion (using a bit vector) can be found in [EFGL08] (page 10:36) in the context of consistent query answering. Yet, when using a ground atom $\gamma$ in the theoretical exposition we will *not* include the marker as a an additional attribute; instead, we will access the marker only with the help of the *marker*-function. This is crucial when a new model operator is introduced in Definition 9.4.

For an arbitrary formula $\Phi$, we define the function $unmarked_v(\Phi)$ that returns the set of all unmarked *ground literals* in $\Phi$ – that is, all literals that contain an unmarked ground atom; this definition is extended to sets of formulas:

---

**Definition 9.2 (Set of unmarked ground literals)**
*For a formula $\Phi$ and a marked instance $db_v$, the following function returns the set of ground literals that are unmarked with respect to $db_v$:*

$$unmarked_v(\Phi) := \{\lambda \mid \lambda \text{ is ground literal in } \Phi \text{ and } marker_v(\lambda) \notin \{\mathtt{k}, \mathtt{a}, \mathtt{r}, \mathtt{l}\}\}$$

*For a set $S$ of arbitrary formulas*

$$unmarked_v(S) := \bigcup_{\Phi \in S} unmarked_v(\Phi)$$

---

Note that $unmarked_v(\Phi)$ does not contain $|\lambda|$ if $\lambda$ is a negative literal. To denote that a ground atom is unmarked in $db_v$, we can now abbreviatorily write $unmarked_v(\gamma) = \gamma$; to denote that a marker for $\gamma$ is set, we can write $unmarked_v(\gamma) = \emptyset$ or analogously $card(unmarked_v(\gamma)) = 0$.

A marked database instance corresponds to a "normal" unmarked database instance when all the ground atoms that are marked with `r` or `l` are removed; that is, the marked database instance is restricted to the "positive" ground atoms:

---

**Definition 9.3 (Positive restriction of $db_v$)**
*The* positive restriction *of a marked database $db_v$ is:*

$$db_v^{pos} := \{\ \gamma \mid \gamma \in db_v \text{ and } marker_v(\gamma) \notin \{\mathtt{r}, \mathtt{l}\}\ \}$$

---

Upon execution of the algorithm, formulas have to be evaluated according to a marked database instance $db_v$. Analogously to an unmarked database instance, a marked instance can also be seen as a finite representation of an interpretation. We base evaluation for marked instances on a new notion of a model of a formula (see also the description of the model operator in Section 7.1). We thus define the model operator for ground atoms with respect to a marked instance:

**Definition 9.4 (Model operator for marked database instance)**
*The interpretation $I^{db_v}$ induced by a marked database instance $db_v$ is a* model *of a ground atom $\gamma$ (written as $I^{db_v} \models \gamma$) in the following case:*

$$I^{db_v} \models \gamma \quad \text{iff} \quad \gamma \in db_v \text{ and } marker_v(\gamma) \notin \{\mathtt{r}, \mathtt{l}\}$$

*In all other cases, $I^{db_v}$ is not a model of $\gamma$ (written as $I^{db_v} \not\models \gamma$).*

In other words, a ground atom is *true* in $I^{db_v}$ if it is contained in (the positive part of) $db_v$ and either marked with `k` or `a` or unmarked; it is *false* in $I^{db_v}$ if it is either not contained in $db_v$ or marked with `r` or `l`.

Definition 9.4 extends as usual to formulas containing Boolean connectives or quantifiers. This way we get a full-blown model operator $\models$ for marked database instances. Again, we only use this operator with closed formulas and we dispense with a variable assignment for free variables. Note that the induced interpretation for a marked database instance coincides with the one for its positive restriction: $I^{db_v} = I^{db_v^{pos}}$. With the help of the new model operator, we now formally define the evaluation function called $eval_v$ for closed as well as open formulas in a marked database instance; we differentiate the positive and the negative part of a response.

**Definition 9.5 (Evaluation function for marked database instance)**
*A closed formula $\Phi$ is evaluated in a marked database instance $db_v$ according to the following functions – the* positive *and the* negative evaluation *(again, we skip the curly braces for singleton sets):*

$$eval_v^{pos}(\Phi) = \begin{cases} \Phi & \text{if } I^{db_v} \models \Phi \\ \emptyset & \text{else} \end{cases}$$

*and*

$$eval_v^{neg}(\Phi) = \begin{cases} \emptyset & \text{if } I^{db_v} \models \Phi \\ \neg\Phi & \text{else} \end{cases}$$

*In general,*

$$eval_v(\Phi) = \begin{cases} \Phi & \text{if } I^{db_v} \models \Phi \\ \neg\Phi & \text{else} \end{cases}$$

*An open formula $\Phi(\vec{x})$ is evaluated in a marked database instance $db_v$ according to the following functions (for the positive and the negative part of $I^{db_v}$):*

$$eval_v^{pos}(\Phi(\vec{x})) := \{\Phi(\vec{a}) \mid \vec{a} \subset dom \text{ and } I^{db_v} \models \Phi(\vec{a})\}$$

$$eval_v^{neg}(\Phi(\vec{x})) := \{\neg\Phi(\vec{a}) \mid \vec{a} \subset dom \text{ and } I^{db_v} \not\models \Phi(\vec{a})\}$$

*The combined positive and negative part are retrieved with the function:*

$$eval_v(\Phi(\vec{x})) := eval_v^{pos}(\Phi(\vec{x})) \cup eval_v^{neg}(\Phi(\vec{x}))$$

---

Next, we have to identify those constraints in the constraint set $C$ that are "violated" in a marked database instance $db_v$; that is, in a complete database, their negation is satisfied in $db_v$. As all the constraints are closed formulas, we can define them via their evaluation in a marked database instance as follows.

---

**Definition 9.6 (Violated constraints)**
*A constraint $\Phi \in C$ is* violated *in a marked database instance $db_v$ iff its evaluation returns its negation, that is*

$$eval_v(\Phi) = \neg\Phi$$

---

## 9.1 The *pre*CQE Algorithm

Having defined all the necessary terminology, we move on to an algorithmic description of how an inference-proof and distortion-minimal solution instance can be computed. As mentioned earlier, in this section we concentrate on an algorithm for allowed universal formulas. *pre*CQE for allowed universal constraints comprises some procedures operating on marked database instances; they are given in pseudo code in Listings 1 to 5 with numbered lines. *pre*CQE starts with the initialization procedure `INIT` in the root node of the search tree. The `PRUNE` procedure – responsible for backtracking (or backjumping) in the search tree – is not explicitly adduced.

*pre*CQE achieves confidentiality (by means of inference-proofness) and availability (by means of distortion minimality) at the same time. While confidentiality requirements are strict (the solution instance must represent a model of the constraints), availability optimization is achieved by a "Branch and Bound" approach on the distortion distance *db_dist*. Branch and Bound (B&B, for short) is a method for finding solutions to an optimization problem more efficiently than purely by a

1. `INIT`: **Initialization** for root node $r$
   1.1. create root node $r$
   1.2. $db_r$:=$db$;
   1.3. $min\_lies_r := 0$;
   1.4. $C_r := C$;
   1.5. $db_{best} :=$ *undefined*;
   1.6. $min\_lies_{best} := \infty$;
   1.7. `GROUND`($r$);

Listing 1: *pre*CQE – Initialization for root node $r$

2. GROUND($v$): Determine **ground violations** in node $v$
   2.1. $C_v^{vio} := \{\Phi \in C_v \mid eval_v(\Phi) = \neg\Phi\}$;
   2.2. `if` $(C_v^{vio} = \emptyset)$
      2.2.1. $db_{best} := db_v$;
      2.2.2. $min\_lies_{best} := min\_lies_v$;
   2.3. `else`
      2.3.1. `foreach` $\Phi_i \in C_v^{vio}$
         2.3.1.1. $\Phi_i' := plnf(\neg\Phi_i)$;
         2.3.1.2. $\Phi_i'' := dropprenex(\Phi_i')$;
      2.3.2. $V_v := \bigcup_i eval_v^{pos}(\Phi_i'')$;
      2.3.3. SIMP($v$);
      2.3.4. `if` ( there is $\psi \in V_v$ with $card(unmarked_v(\psi)) = 0$ )
         2.3.4.1. PRUNE; //(conflicting markers)
      2.3.5. `else if` ( there is $\psi \in V_v$ with $card(unmarked_v(\psi)) = 1$ )
         2.3.5.1. take unique literal $\lambda \in unmarked_v(\psi)$;
         2.3.5.2. MARK($v$,$\overline{\lambda}$);
         2.3.5.3. GROUND($v$);
      2.3.6. `else`
         2.3.6.1. SPLIT($v$);

Listing 2: *pre*CQE – Computing ground violations

3. SIMP($v$): **Simplification** of violation set $V_v$
   3.1. `repeat` until no more changes occur:
      3.1.1. `foreach` subformula $\psi'$ of a formula $\psi \in V_v$
         3.1.1.1. `if` $(\psi' = \phi \wedge \gamma$ and $marker_v(\gamma) \in \{\mathtt{k},\mathtt{a}\}$
                  `or` $\psi' = \phi \wedge \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{r},\mathtt{l}\}$
                  `or` $\psi' = \phi \vee \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{k},\mathtt{a}\}$
                  `or` $\psi' = \phi \vee \gamma$ and $marker_v(\gamma) \in \{\mathtt{r},\mathtt{l}\})$
            3.1.1.1.1. replace $\psi'$ with $\phi$; //(unit resolution)
         3.1.1.2. `if` $(\psi' = \phi \wedge \gamma$ and $marker_v(\gamma) \in \{\mathtt{r},\mathtt{l}\}$
                  `or` $\psi' = \phi \vee \gamma$ and $marker_v(\gamma) \in \{\mathtt{k},\mathtt{a}\})$
            3.1.1.2.1. replace $\psi'$ with $\gamma$; //(unit subsumption)
         3.1.1.3. `if` $(\psi' = \phi \wedge \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{k},\mathtt{a}\}$
                  `or` $\psi' = \phi \vee \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{r},\mathtt{l}\})$
            3.1.1.3.1. replace $\psi'$ with $\neg\gamma$; //(unit subsumption)

Listing 3: *pre*CQE – Simplifying violation set

4. SPLIT($v$): **Splitting** on a ground atom in node $v$
   4.1. choose $\psi \in V_v$;
   4.2. choose $\lambda \in unmarked_v(\psi)$;
   4.3. generate two child nodes $v_{left}$ and $v_{right}$;
   4.4. $db_{v_{left}} := db_{v_{right}} := db_v$;
   4.5. $C_{v_{left}} := C_{v_{right}} := C_v$;
   4.6. $min\_lies_{v_{left}} := min\_lies_{v_{right}} := min\_lies_v$;
   4.7. MARK($v_{left}$,$\neg|\lambda|$);
   4.8. GROUND($v_{left}$);
   4.9. MARK($v_{right}$,$|\lambda|$);
   4.10. GROUND($v_{right}$);

Listing 4: *pre*CQE – Splitting on a ground atom

5. `MARK(`$v$`,`$\lambda$`)`: **Marking** an unmarked ground atom $\gamma$ in $db_v$
   5.1. $\gamma := |\lambda|$;
   5.2. `if (`$\lambda = \gamma$ `and` $eval_v(\gamma) = \gamma$ `)`
      5.2.1. $marker_v(\gamma)$ :=`k;`
   5.3. `else if (`$\lambda = \gamma$ `and` $eval_v(\gamma) = \neg\gamma$ `)`
      5.3.1. $marker_v(\gamma)$ :=`a;`
      5.3.2. $min\_lies_v$`++;`
   5.4. `else if (`$\lambda = \neg\gamma$ `and` $eval_v(\gamma) = \gamma$ `)`
      5.4.1. $marker_v(\gamma)$ :=`r;`
      5.4.2. $min\_lies_v$`++;`
   5.5. `else if (`$\lambda = \neg\gamma$ `and` $eval_v(\gamma) = \neg\gamma$ `)`
      5.5.1. $marker_v(\gamma)$ :=`l;`
   5.6. `if (`$min\_lies_v \geq min\_lies_{best}$`)` `PRUNE; //`(bad bound)

Listing 5: *pre*CQE – Marking a ground atom

complete search with backtracking. It offers the features "branching", "bounding" and "pruning". These three feature are briefly desribed as follows.

- Branching: dividing the problem into adequate subproblems
- Bounding: efficiently computing local lower and upper bounds for subproblems
- Pruning: discarding a subproblem due to a bad bound value

For a minimization problem (as ours), a global upper bound is maintained stating the currently best value. A B&B-algorithm may have a super-polynomial running time; however, execution may be stopped with the assurance that the optimal solution's value is in between the global upper bound and the minimum of the local lower bounds. The branching step of B&B coincides with the splitting operation with which we search for satisfying truth value assignments for ground atoms.

When the algorithm is run, the Branch and Bound search tree is (virtually) constructed. For each node $v$ in the search tree we need the following elements:

- a marked database $db_v$ that might however still be inconsistent with the constraints; in other words, it might be the case that $I^{db_v} \not\models C$. Yet, if $v$ is an unpruned leaf in the search tree, we can be sure that $I^{db_v} \models C$.
- a lower bound for the distortion distance *db_dist* called $min\_lies_v$, defined as the number of ground atoms (of $\mathscr{L}$) with different evaluation in $db$ than in $db_v$:
$$min\_lies_v := \{\ \gamma \mid eval^*(\gamma)(db) \neq eval_v(\gamma)\}$$
  The value of $min\_lies_v$ is monotonically nondecreasing between a node $v$ and its child nodes.
- a local constraint set $C_v$. In fact, for universal constraints (in this first version of the algorithm we present here) these constraint sets are the same in every

node $v$. Yet, when allowing existential quantification in Section 10 (or introducing simplification of constraints), these constraint sets will indeed change from node to node; that is why we use $v$ as an index.

We also have a current global optimum $db_{best}$ that stores the best solution found so far; that is, $db_{best}$ (which is a marked database instance) is a model of $C$: $I^{db_{best}} \models C$. The same is true for its positive restriction $db^{pos}_{best}$ (which is an unmarked database instance): $I^{db^{pos}_{best}} \models C$. In other words, the marked database instance $db_{best}$ corresponds to a leaf in the search tree whose branch is completely explored and all constraints are satisfied. For such optima $db_{best}$ we can be sure that the value $min\_lies_{best}$ is indeed the value of the distortion distance (for which the positive restriction is used): $min\_lies_{best} = db\_dist(db^{pos}_{best})$.

There are two conditions under which exploration of the current branch is stopped (that is, the branch is pruned) and backtracking to an alternative branch is started; the two conditions are:

1. [**Conflict**] A conflicting truth value assignment has been found; that is, there is a violated constraint but in at least one of its violating ground instantiations all literals are already marked. The current instance $db_v$ cannot be adjusted to satisfy the violated constraint. Hence, in this case the confidentiality requirements are violated.
2. [**Bad distortion distance**] In the current instance $db_v$ there are already more lies than (or as many as) in the current global optimum. The current branch will never result in an instance with a better distortion distance. Hence, in this case the availability requirements are met better (or equally well) with the currently best solution. This condition is checked directly after the local lower bound was recalculated (see Line 5.6.)

We now describe each procedure in detail referring to the line numbering of Listings 1 to 5:

`INIT` **(Listing 1)** *pre*CQE starts with some initialization in the root node of the search tree. The marked database instance (called $db_r$) is identical to the input instance with all entries unmarked (Line 1.2.); therefore, the local lower bound for the distortion distance is 0 (Line 1.3.). The local constraint set $C_r$ (Line 1.4.) is initialized with $C$. The global optimum is undefined (Line 1.5.) and initially the global upper bound for the distortion distance is $\infty$ (Line 1.6.). Then, the treatment of the constraints is started with a call to `GROUND` (Line 1.7.).

`GROUND` **(Listing 2)** When starting the `GROUND` procedure, all violated constraints are determined (Line 2.1.). If none is detected, a new global optimum is found (Line 2.2.1.) and the global upper bound for the distortion distance is set to the optimum's distance value (Line 2.2.2.).

If there are violated constraints, their satisfaction is achieved as follows. The constraints are negated and transformed into their PLNF representation (Line 2.3.1.1.). Then, for all these formulas, the prenex (including all quantifiers) is dropped (see Line 2.3.1.2.). The resulting formulas are evaluated in the marked database instance $db_v$ (Line 2.3.2.) such that we have the set $V_v$ of "ground violations": all database entries that impede the satisfaction of the constraints. The violation formulas are simplified with the `SIMP` operation.
It may happen that an unresolvable conflict is found (Line 2.3.4.): the markers for a ground instantiation of a constraint are already set, but still the corresponding constraint is not satisfied. In this case, the branch is pruned (Line 2.3.4.1.).
Next, we try to set markers (of unmarked ground atoms in the violations) in such a way that the constraints are satisfied. If there is a violation with only one ground literal unmarked (see Line 2.3.5.1.), we take advantage of the fact that the violations are transformed into PLNF and thus negations only appear directly in front of atoms. When trying to satisfy the constraint formula, our only chance is by assigning the violating ground literal $\lambda$ the truth value *false* (or conversely, $\overline{\lambda}$ the truth value *true*; see Line 2.3.5.2.). Note that the distortion distance for each marked instance is checked and adjusted in the `MARK` procedure (see Lines 5.3.2. and 5.4.2. and the description below). Then, a recursive call to `GROUND` is executed in which the effect of the new assignment on the constraints (including the constraint for which the marker was set) is checked.
When none of the above cases holds, we start the expansion of the search with a call to the `SPLIT` procedure (Line 2.3.6.1.).

`SIMP` **(Listing 3)** If markers for literals have been set, the ground formulas in the violation set $V_v$ can be simplified: a subformula that contains a marked literal can be replaced by a simpler formula. These replacements yield an equivalent formula in accordance with the partial interpretation that is represented by the markers. More precisely, the `SIMP` procedure differentiates the following cases:

- In a subformula $\phi \wedge \lambda$ or $\phi \vee \lambda$ (see Line 3.1.1.1.), the marked literal $\lambda$ does not influence the solution anymore, but still markers can be set in the subformula $\phi$; then, $\lambda$ is removed and only $\phi$ is kept.
- In a subformula $\phi \wedge \lambda$ or $\phi \vee \lambda$ (see Lines 3.1.1.2. and 3.1.1.3.), the marked literal $\lambda$ determines the truth valuation of the whole subformula $\psi'$, and thus markers in the subformula $\phi$ do not influence the solution; then, $\phi$ is removed and only $\lambda$ is kept.

Note that commutativity of the Boolean connectives $\wedge$ and $\vee$ is assumed; that is, the procedure not only applies to $\phi \wedge \gamma$ but also to $\gamma \wedge \phi$ and analogously for the other cases. This kind of truth value simplification is also used in a slightly different form in [GT91] (Definition 8.2). Our version here helps us exclude those unmarked ground literals that are "don't care" in the current partial interpretation $I^{db_v}$ and thus can retain their truth value without the need of setting any markers for them.

As a small example consider $db = \{\, \mathit{Treat}(\mathsf{Mary}, \mathsf{MedA})\}$ and

$$C = \{\neg \mathit{Treat}(\mathsf{Mary}, \mathsf{MedA}),\ \mathit{Treat}(\mathsf{Mary}, \mathsf{MedA}) \wedge \neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids})\}$$

Assume that $\mathit{marker}_v(\mathit{Treat}(\mathsf{Mary}, \mathsf{MedA})) = \mathtt{r}$. Then, the violation set is in fact $V_v = \{\neg \mathit{Treat}(\mathsf{Mary}, \mathsf{MedA}) \vee \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids})\}$ where the literal $\neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids})$ is unmarked. Yet, the simplified violation set is $V_v = \{\neg \mathit{Treat}(\mathsf{Mary}, \mathsf{MedA})\}$ (by application of Line 3.1.1.3.) where no literal is unmarked; now we can immediately prune (see Line 2.3.4.1.) and no marker for $\neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids})$ has to be set.
Note that this ground simplification is only possible on the set $V_v$ (which contains only ground formulas) but not on the non-ground constraint set $C$.

`SPLIT` **(Listing 4)** If there are only violations with more than one unmarked literal, one literal of one of the violations is chosen as the splitting literal (Line 4.2.) and two child nodes are generated and initialized with the current node's values (Lines 4.3. to 4.6.). Then, first the left branch is explored where the splitting literal's ground atom is assigned *false* (Line 4.7.); we continue with the `GROUND` procedure in the left branch (Line 4.8.). Afterward, the complementary right branch is treated with the ground atom assigned *true* (Lines 4.9. and 4.10.).

`MARK` **(Listing 5)** The `MARK` procedure takes as input a yet unmarked literal and retrieves the ground atom from it (Line 5.1.). Then the marker of the ground atom in the current instance $db_v$ is adjusted to satisfy the input literal. This is done according to the four types of markers that can occur (see Lines 5.2.1., 5.3.1., 5.4.1. and 5.5.1.). Whenever the marker changes the evaluation of the ground atom (that is, swaps its truth value as is the case with the markers `a` in Line 5.3. and `r` in Line 5.4.), the local lower bound of the distortion distance is incremented (see Lines 5.3.2. and 5.4.2.). Finally, the current distortion distance is compared with the global optimum value and the branch is pruned if necessary (see Line 5.6.).

Now we can have a second look at Example 9.1 and see how it is processed by *pre*CQE; Table 1 shows the markers in the different nodes as pictured in Figure 6.

*Example 9.7:* In the `INIT` procedure, we determine the constraint set of the root node:

$$\begin{aligned} C_r \;=\; \{&\forall x(\neg \mathit{Treat}(x, \mathsf{MedA}) \vee \mathit{Ill}(x, \mathsf{Aids}) \vee \mathit{Ill}(x, \mathsf{Cancer})),\\ &\forall x(\neg \mathit{Treat}(x, \mathsf{MedB}) \vee \mathit{Ill}(x, \mathsf{Cancer}) \vee \mathit{Ill}(x, \mathsf{Flu})),\\ &\forall x \neg \mathit{Ill}(x, \mathsf{Aids}),\\ &\forall x \neg \mathit{Ill}(x, \mathsf{Cancer})\} \end{aligned}$$

When `GROUND` is started in root node $r$, we find that both unit constraints are violated, that is $C_r^{vio} = \{\forall x \neg \mathit{Ill}(x, \mathsf{Aids}), \forall x \neg \mathit{Ill}(x, \mathsf{Cancer})\}$ and the violations are

| *Ill* | *Name* | *Diagnosis* | $r$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|---|---|---|
| | Pete | Aids | r | r | r | r | r |
| | Mary | Cancer | r | r | r | r | r |
| | Mary | Flu | | | | a | |
| | Pete | Cancer | | | | | a |

| *Treat* | *Name* | *Treatment* | $r$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|---|---|---|
| | Pete | MedA | | r | r | r | k |
| | Mary | MedB | | | r | k | |

Table 1: Markers set by *pre*CQE

computed as

$$\begin{aligned} V_r &= eval_r^{pos}(\mathit{Ill}(x, \mathsf{Aids})) \cup eval_r^{pos}(\mathit{Ill}(x, \mathsf{Cancer})) \\ &= \{\mathit{Ill}(\mathsf{Pete}, \mathsf{Aids}), \mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer})\} \end{aligned}$$

Both of these ground atoms are marked with `r` by executing `MARK`$(r, \neg\mathit{Ill}(\mathsf{Pete}, \mathsf{Aids}))$ and `MARK`$(r, \neg\mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer}))$ in two subsequent calls of `GROUND`. Afterward we already have $min\_lies_r = 2$.
Then the next recursion of `GROUND` is started. We find that both non-unit constraints are violated and the violations are evaluated as

$$\begin{aligned} V_r &= eval_v^{pos}(\mathit{Treat}(x, \mathsf{MedA}) \wedge \neg\mathit{Ill}(x, \mathsf{Aids}) \wedge \neg\mathit{Ill}(x, \mathsf{Cancer})) \\ &\cup\ eval_v^{pos}(\mathit{Treat}(x, \mathsf{MedB}) \wedge \neg\mathit{Ill}(x, \mathsf{Cancer}) \wedge \neg\mathit{Ill}(x, \mathsf{Flu})) \\ &= \{\mathit{Treat}(\mathsf{Pete}, \mathsf{MedA}) \wedge \neg\mathit{Ill}(\mathsf{Pete}, \mathsf{Aids}) \wedge \neg\mathit{Ill}(\mathsf{Pete}, \mathsf{Cancer}), \\ &\quad \mathit{Treat}(\mathsf{Mary}, \mathsf{MedB}) \wedge \neg\mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer}) \wedge \neg\mathit{Ill}(\mathsf{Mary}, \mathsf{Flu})\} \end{aligned}$$

We see that a call to `SPLIT` is necessary as both violations have two unmarked literals each. We choose the first formula in $V_r$ as $\psi$ according to Line 4.1. and decide to split on the unmarked literal $\mathit{Treat}(\mathsf{Pete}, \mathsf{MedA})$ (see Line 4.2.). In the root's left child node $v_1$, this ground atom is marked with `r` (see Line 4.7.) which leads to $min\_lies_{v_1} = 3$. In the following `GROUND` operation, we find the violations

$$V_{v_1} = \{\mathit{Treat}(\mathsf{Mary}, \mathsf{MedB}) \wedge \neg\mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer}) \wedge \neg\mathit{Ill}(\mathsf{Mary}, \mathsf{Flu})\}$$

We decide to split on $\mathit{Treat}(\mathsf{Mary}, \mathsf{MedB})$: in the left child node $v_2$, it is marked with `r`. Afterward, no violated constraints are left such that the marked database of $v_2$ is the currently best solution candidate ($db_{best} := db_{v_2}$) with a distance value $db\_dist_{best} = 4$.
The right child node of $v_1$ is called $v_3$; here, $\mathit{Treat}(\mathsf{Mary}, \mathsf{MedB})$ is marked with `k`. Another `GROUND` operation follows. Again we find the same constraint violated in $v_3$, but $\neg\mathit{Ill}(\mathsf{Mary}, \mathsf{Flu})$ is the only unmarked literal. The violation can only be

avoided by marking the ground atom with `a` (see Line 2.3.5.2.). This actually gives an equally good solution as $v_2$ (with distance 4), but it is pruned (see Line 5.6.) as we prefer the first solution found.
We continue the search in the root's right child node $v_4$. $\mathit{Treat}(\mathsf{Pete}, \mathsf{MedA})$ is marked with `k` (see Line 4.9.), such that the lower bound remains unchanged after the splitting: $min\_lies_{v_4} = 2$. However, in the subsequent `GROUND` operation we find

$$V_{v_4} \quad = \quad \{\, \mathit{Treat}(\mathsf{Pete}, \mathsf{MedA}) \wedge \neg \mathit{Ill}(\mathsf{Pete}, \mathsf{Aids}) \wedge \neg \mathit{Ill}(\mathsf{Pete}, \mathsf{Cancer}) \}$$

Of the violation only $\neg \mathit{Ill}(\mathsf{Pete}, \mathsf{Cancer})$ is unmarked and its ground atom has to be marked with `a` (see Line 2.3.5.2.). In the next recursion, we find the unit constraint $\neg \mathit{Ill}(\mathsf{Pete}, \mathsf{Cancer})$ violated (see Line 2.3.4.); that is why the branch is pruned. ♢

## 9.2 Termination, Soundness and Completeness of *pre*CQE

We now examine the general properties in terms of termination, soundness and completeness of the *pre*CQE procedure as described in the previous section. The constraint set $C$ is restricted to be a set of allowed universal constraints owed to the fact that for arbitrary constraints termination cannot be ensured. We start with the observation that the violation sets $V_v$ are finite.

---

**Lemma 9.8 (Finite violation sets with *adom* constants)**
*For a set $C$ of allowed universal constraints, the set $V_v$ of ground violations (according to Line 2.3.2.) is always finite for any node $v$ and contains only ground formulas with constants from adom.*

---

*Proof.* The constraint set $C$ is a finite set of closed formulas; thus, also $C_v$ is. In the `GROUND` procedure, the set of violated constraints $C_v^{vio}$ is determined, which is a subset of $C_v$ and thus finite.
Then, each of these violated constraints is negated and put into PLNF and the prenex of the formula is dropped (if the formula is a non-ground formula) such that only its matrix is left and all variables now occur freely. At this point, we can resort to Lemma 8.7 and Remark 8.8 to assure ourselves that the evaluation returns a finite set of ground formulas; and, what is more, the formulas contain only constants from *adom*. □

The pivotal result of this part is that for allowed universal constraints termination of *pre*CQE can be ensured:

---

**Theorem 9.9 (Termination of *pre*CQE)**
*For a set $C$ of allowed universal constraints, preCQE terminates in a finite amount of time.*

---

*Proof.* The first argument is that all sets of formulas that are computed in a *pre*CQE run are finite. We easily see that this is the case for $C_v^{vio}$ in Line 2.1. (as it is a subset of the local constraint set $C_v$ which is exactly the finite constraint set

$C$). This implies that there are only finitely many iterations of the foreach loop that processes each formula of $C_v^{vio}$ in Line 2.3.1. Moreover, each violation set $V_v$ (see Line 2.3.2.) is finite: we have shown its finiteness for allowed constraints in Lemma 9.8; we already commented on its computability in Remark 8.8. But then indeed we can be sure that

- the foreach loop in `SIMP` (Line 3.1.1.) needs only a finite number of iterations
- the if-conditions in Lines 2.3.4. and 2.3.5. can be checked in finite time
- the selection of the splitting formula (see Line 4.1.) is based on the finite set $V_v$

Moreover, as the set $V_v$ consists of formulas of finite length, the repetition of the simplification of formulas of $V_v$ (see Line 3.1.) only takes finite time because, whenever an application is possible, the length of one of the formulas decreases (see Lines 3.1.1.1.1., 3.1.1.2.1. and 3.1.1.3.1.). All other steps in the five procedures consist of finite instructions.
Next, we have to show that the number of (recursive) calls of the procedures is finite. We observe that `GROUND` and `SPLIT` are the only procedures involved in the recursion: `GROUND` calls itself in Line 2.3.5.3. or calls `SPLIT`; `SPLIT` recurs to `GROUND` twice (Lines 4.8. and 4.10.). The recursion is only stopped in the following three cases (which correspond to leaves in the search tree):

1. a (new) optimum is found (Line 2.2.)
2. the first pruning condition arises: the current partial interpretation contains a conflict (see Line 2.3.4.1.)
3. the second pruning condition arises: a better solution than the current partial interpretation has already been found (see Line 5.6.)

Each recursive call to `GROUND` is preceded by a `MARK` operation (see Lines 2.3.5.2., 4.7. and 4.9.). We can apply Lemma 9.8 to note that the `MARK` operation is only executed on ground literals with *adom* constants that are taken from ground formulas in $V_v$. We have already in Corollary 8.14 used the fact that there are at most $k$ different ground atoms with *adom* constants where

$$k := \sum_{\substack{P \in \mathcal{P} \\ P \text{ occurs in } C}} card(adom)^{arity(P)}$$

This implies that there are at most $2^k$ literals containing such ground atoms. More precisely, `MARK` is only applied to unmarked literals. Yet, with each `MARK` operation the overall number of unmarked ground atoms (with *adom* constants) decreases and so does the number of unmarked literals. From this it follows that the recursion depth is in fact bounded by $k$. This corresponds to the fact that the length of each branch in the tree is bounded by $k$.
Moreover, the search tree is binary (due to splitting) and there are $2^k$ ways to assign

each of these $k$ ground atoms either the truth value *true* or the truth value *false*. This is why there are at most $2^k$ branches in the search tree. □

Based on this termination result, we have to make sure that the algorithm is sound and complete. To this end, we first of all observe that $db_{best}$ is either a marked database instance or $db_{best}$ is *undefined*; second we differentiate the cases that $C$ is a satisfiable constraint set and the case that $C$ is unsatisfiable. Due to this, we investigate the algorithm's *satisfiability soundness* and *satisfiability completeness* – the algorithm finds a marked database instance if and only if there exists an inference-proof solution. Owed to the fact that we already proved termination, we can equivalently show the algorithm's *refutation soundness* and *refutation completeness* – the algorithm returns *undefined* if and only if there does not exist a solution. We start with the proof of satisfiability soundness.

**Theorem 9.10 (Satisfiability soundness of *pre*CQE)**
*For a set $C$ of allowed universal constraints and after running preCQE to completion, if $db_{best}$ is a marked database instance, then its positive restriction is an inference-proof database instance.*

*Proof.* The assignation of a marked database instance to $db_{best}$ happens only if in some node $v$, $C_v^{vio}$ is empty (Line 2.2.); that is, no violated constraint is left: there is no $\Phi \in C$ with $eval_v(\Phi) = \neg\Phi$. This again means that $I^{db_{best}} \models C$ and thus (by Definitions 9.3 and 9.4) $I^{db_{best}^{pos}} \models C$. With the help of Corollary 7.8 we conclude that $db_{best}^{pos}$ is an inference-proof database instance. □

Refutation completeness follows from satisfiability soundness as the contraposition of Theorem 9.10.

**Corollary 9.11 (Refutation completeness of *pre*CQE)**
*For a set $C$ of allowed universal constraints, if $C$ is unsatisfiable, then – after running preCQE to completion – $db_{best}$ is undefined.*

*Proof.* If $C$ is unsatisfiable, there does not exist a model of $C$ and thus for every node $v$ there is a $\Phi \in C$ with $eval_v(\Phi) = \neg\Phi$. Because of this, $C_v^{vio}$ is never empty. But then, $db_{best}$ is never assigned a marked database instance and the initialization of $db_{best}$ with *undefined* is never changed. □

As for satisfiability completeness, due to the active domain semantics for allowed universal constraints that was shown in Section 8, a trivially complete algorithm would simply test all possible assignations of truth values for all ground instantiations of atoms with *adom* constants. We state that *pre*CQE has a good chance of doing better than the trivially complete algorithm on *adom*-ground atoms:

*Remark 9.12 (Efficiency of preCQE):* With the *pre*CQE algorithm at best not all *adom*-ground atoms have to be explicitly marked (that is, assigned the truth values *true* or *false*) due to the following reasons:

1. Only violated constraints and the ground atoms affected by these are considered.
2. If a truth assignment is unequivocal, ground atoms are marked directly without splitting.
3. Branches are pruned if a better solution has already been found.
4. Branches are pruned as soon as a conflict occurs.

This means that the *pre*CQE search in the best case does not contain all the possible $2^k$ branches mentioned in Theorem 9.9 but contains considerably less and shorter branches. This definitely applies to Example 9.7 and Figure 6: Observing that

$$adom = \{\mathsf{Mary}, \mathsf{Pete}, \mathsf{Cancer}, \mathsf{Aids}, \mathsf{Flu}, \mathsf{MedA}, \mathsf{MedB}\}$$

we deduce the following set of ground atoms of *Ill* and *Treat* with *adom* constants (respecting the sorts):

$$\begin{aligned}\{\quad & \mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer}), \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids}), \mathit{Ill}(\mathsf{Mary}, \mathsf{Flu}),\\ & \mathit{Ill}(\mathsf{Pete}, \mathsf{Cancer}), \mathit{Ill}(\mathsf{Pete}, \mathsf{Aids}), \mathit{Ill}(\mathsf{Pete}, \mathsf{Flu}),\\ & \mathit{Treat}(\mathsf{Pete}, \mathsf{MedA}), \mathit{Treat}(\mathsf{Pete}, \mathsf{MedB}),\\ & \mathit{Treat}(\mathsf{Mary}, \mathsf{MedA}), \mathit{Treat}(\mathsf{Mary}, \mathsf{MedB}) \qquad \}\end{aligned}$$

That is, a complete search tree (with respect to *adom*) that assigns to each and every of these ground atoms one of the values *true* or *false* along a branch, has $2^{10} = 1024$ distinct branches of length 10 each. In contrast, Figure 6 shows that there are only 3 branches necessary; moreover, along a branch there occur only 4 to 5 truth value assignments (instead of 10). ♢

Owed to this increased efficiency of the search, we have to confirm, that no solution is erroneously skipped with *pre*CQE. More precisely, we first show refutation soundness of the algorithm and then obtain satisfiability completeness as its contraposition. In the proof of refutation soundness, we need one form of "Herbrand's Theorem". We quote it as stated by Cook and Nguyen in [CN09] (page 34) but ignore their incorporation of equality axioms and function symbols both of which we do not need in our settings.

---

**Theorem 9.13 (Herbrand's Theorem (cf. [CN09]))**
*Let $S$ be a set of closed universal formulas. Then $S$ is unsatisfiable iff some finite set $S_0$ of ground instances of formulas in $S$ is propositionally unsatisfiable.*

---

Cook and Nguyen define a "ground instance" of a universal formula as obtained by removing the prenex and substituting the variables in the matrix by constant symbols ([CN09], page 34). There they also define propositional (un-)satisfiability with the help of truth value assignments to ground formulas of a first-order language $\mathscr{L}$. Further, we will take advantage of results of "semantic trees" that are known to be

sound for propositional unsatisfiability. While the concept of semantic trees was first introduced by Robinson for his resolution procedure, we will use [CL73] (pages 56 – 66) as well as [Fit96] (pages 70–72) as references. A semantic tree is usually defined for a set of clauses – that is, a set of disjunctions of literals. This definition is caused by the fact that resolution crucially relies on clausal input. Note however, that we will need a semantic tree for our non-CNF *pre*CQE input in the upcoming Theorem 9.17. Yet, we know that for a ground formula there are equivalence-preserving rewrite rules that transform it into CNF; moreover, any set of ground CNF formulas can be easily transformed into a set of ground clauses by treating each conjunct of a formula as a single clause. Due to this equivalence-preservation we can be sure that a set of ground formulas is (propositionally) satisfiable if and only if the union of the sets of clauses obtained from the formulas is. Hence, we expand the definition of [CL73] to apply to a set of ground formulas (instead of a set of ground clauses).

---

**Definition 9.14 (Semantic tree, failure node)**
*Let $S_0$ be a set of ground formulas and let $A_0$ be the set of ground atoms occurring in $S_0$. A* semantic tree *for $S_0$ is a binary tree where*

1. *the root node has no label*
2. *the inner nodes of the tree have the following labels*
   (a) *each right child node $v$ has as its label a ground atom $\gamma$ chosen from $A_0$*
   (b) *the left sibling of $v$ has $\neg\gamma$ as its label*
3. *for any path from the root node to a node $v$, in the union of all labels along the path, there does not occur a complementary pair of literals.*

*A* failure node *is a node in a semantic tree that falsifies a formula from $S_0$ but none of its ancestors does; that is, the union of labels on the path from the root to the failure node form an interpretation of the ground atoms that makes the formula* false.
*A semantic tree is* closed *if all its leaves are failure nodes.*

---

Note that in the general definition of [CL73] a semantic tree need not be binary; in fact the general condition is that for any set of siblings the disjunction of their labels has to be a valid (that is, tautologous) formula. For our purposes however binary semantic trees as defined in Definition 9.14 suffice. Note, also, that in [CL73] a semantic tree is defined for a fixed enumeration of the Herbrand base pertaining to the considered set of clauses. Yet, for our finite and function-free case, the enumeration is of no importance; this is also noted by [Fit96] (page 63). This implies that we do not run into difficulties with our choosing formulas and splitting literals non-deterministically (in Lines 2.3.5., 4.1. and 4.2.).

It is possible to restate Herbrand's theorem with the help of semantic trees; a similar statement is contained in [CL73] but again they only consider a set of clauses. We thus extend it to a set of universal formulas here.

**Theorem 9.15 (Herbrand's Theorem with semantic tree)**
*Let $S$ be a set of closed universal formulas. Then $S$ is unsatisfiable iff for some finite set $S_0$ of ground instances of formulas in $S$ there is a closed semantic tree.*

We will use this version of Herbrand's Theorem to show refutation soundness of *pre*CQE. We begin with the definition of how a semantic tree is constructed out of a given *pre*CQE search tree. An illustration of the construction and the arguments applied in the proof of the upcoming Theorem 9.17 follows in Example 9.18.

**Definition 9.16 (Construction of semantic tree)**
*Let $T$ be a search tree obtained by running preCQE on a constraint set $C$ (of allowed universal formulas) and an input instance db.*
*$T^*$ is the semantic tree* constructed *from $T$ by traversing $T$ from the root to each leaf and adding nodes to $T^*$ for every node $v$ in $T$ as follows:*

- *If $v$ is the root node in $T$, it is also the unlabeled root node in $T^*$.*
- *For each literal that was marked in $v$ in a* `GROUND` *procedure (in Line 2.3.5.2.), a new level of nodes has to be created in $T^*$ (in the subtree having node $v$ as its root): Assume there are $m$ literals $\lambda_1 \dots \lambda_m$ that are marked in $v$ (according to Line 2.3.5.2.); then,*
  - *for $\lambda_1$ two new child nodes of $v$ have to be created in $T^*$; the left child is labeled with the negative literal $\neg|\lambda_1|$, while the right child is labeled with the positive $|\lambda_1|$. We refer to the two new nodes as the "level of $\lambda_1$".*
  - *for each $\lambda_i$ (for $i = 2 \dots m$) two new nodes are created in $T^*$ which are referred to as the "level of $\lambda_i$". They are appended as child nodes to that node in the level of $\lambda_{i-1}$ that is labeled with $\lambda_{i-1}$; that is, no child nodes are appended to the node in the level of $\lambda_{i-1}$ that is labeled with $\overline{\lambda_{i-1}}$ .*
- *If finally a splitting occurs in $v$ and $v_{left}$ and $v_{right}$ are the two child nodes of $v$ in $T$ (see Line 4.3.), $v_{left}$ and $v_{right}$ are also contained in $T^*$ and labeled with the literal that was marked in the* `SPLIT` *call; that is, $v_{left}$ is labeled with $\neg|\lambda|$ and $v_{right}$ is labeled with $|\lambda|$. We only have to adjust their position in $T^*$ if new nodes were created in $T^*$ immediately before the splitting; that is, if there is a $\lambda_m$ that was marked in $v$ (according to the previous step), then $v_{left}$ and $v_{right}$ in $T^*$ are the child nodes of that node in the level of $\lambda_m$ that is labeled with $\lambda_m$.*

The new tree $T^*$ indeed has the properties of a semantic tree (as defined in Definition 9.14): all nodes in $T^*$ are labeled appropriately and we can be sure that no path contains labels with a complementary pair of literals because labels are based on the markers that are set in $T$ and `MARK` is only called for unmarked literals. It is obvious that the height of $T^*$ is greater than (or equal to) the height of $T$: the new levels possibly increase the height of $T^*$; yet each such level contains only two

nodes and only one of them has two further child nodes while the other is a leaf. We will show later on that any such newly created leaf is indeed a failure node – that is the reason why we do not append any child nodes to such a node in our construction of $T^*$. This in fact will play a significant role in the following theorem of refutation soundness of *pre*CQE.

**Theorem 9.17 (Refutation soundness of *pre*CQE)**
*For a set $C$ of allowed universal constraints and after running preCQE to completion, if $db_{best}$ is undefined, then $C$ is unsatisfiable.*

*Proof.* Let $T$ be a search tree constructed when running *pre*CQE on $C$. By assumption, $db_{best}$ is *undefined*. This can only happen if all branches in the tree $T$ are pruned due to conflicting markers; that is, in each leaf $v$ of $T$ a violation was encountered in $V_v$ for which all markers are set. Recall from Theorem 9.9 that there are only two other cases in which recursion in *pre*CQE is stopped: either an optimum was found or bad bound pruning takes place; both of them imply that $db_{best}$ is not *undefined* but contains a marked database instance instead which immediately violates our assumption.
We aim to prove that the *pre*CQE search tree is a sound method to show the unsatisfiability of the constraint set $C$. The proof consists of two steps:

1. We identify a finite set $C_0$ of ground formulas that are ground instances of formulas in the constraint set $C$.
2. We expand the *pre*CQE search tree to be a closed semantic tree for $C_0$. Then we rely on the fact that this closed semantic tree is a sound method to prove propositional (that is "truth-functional") unsatisfiability of $C_0$.

After these two steps, what follows is that if $db_{best}$ is *undefined*, then $C_0$ is indeed unsatisfiable and by Herbrand's Theorem $C$ also is.
For the first step, we construct $C_0$ with the help of ground formulas that are chosen from the violation sets $V_v$ in *pre*CQE. We make the following observation: For a ground formula $\psi$ in a violation set $V_v$ we know that $\psi$ was obtained by

1. negating a formula $\Phi$ of $C_v^{vio}$ and taking its PLNF representation (Line 2.3.1.1.)
2. dropping the prenex of this formula to obtain an open formula (Line 2.3.1.2.)
3. computing the positive evaluation of the open formula on the database instance $db_v$ (Line 2.3.2.)

That is, it holds that $\psi \in eval_v^{pos}(dropprenex(plnf(\neg\Phi)))$. Due to this and by negating $\psi$ and taking its PLNF representation, we deduce that $plnf(\neg\psi)$ is in fact a ground instance of $\Phi$; the reader should bear in mind the fact that negation is used twice: once on the non-ground constraint formula $\Phi$ and once on the ground violation formula $\psi$. We collect these ground instances while running the *pre*CQE algorithm – that is, when the *pre*CQE search tree $T$ is built. We have a closer look at the `GROUND` procedure and identify the ground formulas that are added to $C_0$:

- `GROUND` Line 2.3.4.: for a $\psi \in V_v$ with $card(unmarked_v(\psi)) = 0$, add $plnf(\neg\psi)$ to $C_0$ if $\psi$ was not simplified in Line 2.3.3.; otherwise, if $\psi$ was obtained by simplification from a ground formula $\psi^*$ (in Line 2.3.3.), add $plnf(\neg\psi^*)$ to $C_0$
- `GROUND` Line 2.3.5.: for a $\psi \in V_v$ with $card(unmarked_v(\psi)) = 1$, add $plnf(\neg\psi)$ to $C_0$ if $\psi$ was not simplified in Line 2.3.3.; otherwise, if $\psi$ was obtained by simplification from a ground formula $\psi^*$ (in Line 2.3.3.), add $plnf(\neg\psi^*)$ to $C_0$

We now come to the second step: show that the set $C_0$ is unsatisfiable by showing that there is a closed semantic tree for $C_0$. Let $T^*$ be the semantic tree constructed from the *pre*CQE search tree $T$ by Definition 9.16. We will in fact show that $T^*$ is a closed semantic tree for $C_0$. Recall that each branch in the search tree $T$ is pruned. We now show that each leaf in the semantic tree $T^*$ is a failure node that falsifies a formula in $C_0$. Hence, let $v^*$ be a leaf in $T^*$.

1. If $v^*$ is identical to a leaf $v$ in $T$, then $v$ was pruned in $T$ because there is a conflicting formula $\psi \in V_v$ (after simplification) such that $card(unmarked_v(\psi)) = 0$. This pruning occurred as a consequence of a `MARK` operation for a literal $\lambda^*$ in $v$ that makes $\lambda^*$ *true* in $v$; but then, by construction of $T^*$, $v^*$ is labeled with $\lambda^*$ and $\lambda^*$ is also *true* in $v^*$. Moreover, all labels on the path from the root of $T^*$ to $v^*$ correspond to the markers in $T$. Hence, $v^*$ falsifies $plnf(\neg\psi)$. Next, let $\psi^*$ be the formula from which $\psi$ was obtained by simplification. Then, $v^*$ also falsifies $plnf(\neg\psi^*)$ because simplification indeed preserves the (un-)satisfiability in a given (partial) interpretation. Lastly, $plnf(\neg\psi^*)$ is a formula of $C_0$ by construction.

2. If $v^*$ is not contained in $T$, then it is a leaf in a new level of a literal $\lambda^*$ in $T^*$. But $\lambda^*$ was marked (as *true*) in a node $v$ in $T$ because $\overline{\lambda^*}$ was chosen from a formula $\psi \in V_v$ (after simplification) with $card(unmarked_v(\psi)) = 1$. This node $v$ is then an ancestor of $v^*$ in $T^*$.

   - If $v^*$ is labeled with $\overline{\lambda^*}$, then $\lambda^*$ is *false* in $v^*$. We convince ourselves that $v^*$ falsifies $plnf(\neg\psi)$: because $\psi$ contains $\overline{\lambda^*}$, $plnf(\neg\psi)$ contains $\lambda^*$; yet, $\lambda^*$ is the only unmarked literal of $plnf(\neg\psi)$ and all other marked literals do not satisfy $plnf(\neg\psi)$. By construction, all ancestors of $v^*$ are labeled according to the markers in $T$. Thus $plnf(\neg\psi)$ is indeed *false* in $v^*$.
     In analogy to the previous case, let $\psi^*$ be the formula from which $\psi$ was obtained by simplification. Then, $v^*$ also falsifies $plnf(\neg\psi^*)$ because of satisfiability-preservation. And again, $plnf(\neg\psi^*)$ indeed is a formula of $C_0$ by construction.

   - If $v^*$ is labeled with $\lambda^*$ then $v$ was pruned in $T$ immediately after $\lambda^*$ was marked in $v$ because otherwise $v^*$ would not be a leaf in $T^*$. But then again, there is a formula $\psi' \in V_v$ such that $card(unmarked_v(\psi')) = 0$. Hence, $v^*$ falsifies $plnf(\neg\psi')$ and also $plnf(\neg\psi'^*)$ if $\psi'^*$ is the formula from which $\psi'$ was obtained by simplification.

No ancestor of $v^*$ in $T^*$ falsifies a formula in $C_0$, because otherwise pruning would have occurred earlier in $T$. These are all cases that have to be considered and thus $T^*$ falsifies $C_0$. From this the claim of the theorem follows. □

*Example 9.18:* As an example for refutation soundness of *pre*CQE we consider the following unsatisfiable constraint set:

$$\begin{aligned} C \;=\; \{&\forall x(\neg \mathit{Treat}(x, \mathsf{MedA}) \vee \mathit{Ill}(x, \mathsf{Aids}) \vee \mathit{Ill}(x, \mathsf{Cancer})),\\ &\mathit{Treat}(\mathsf{Mary}, \mathsf{MedA}) \wedge \neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer}),\\ &\neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids})\}. \end{aligned}$$

We assume that the input database instance is

$$db = \{\, \mathit{Treat}(\mathsf{Mary}, \mathsf{MedA}), \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids})\}$$

*pre*CQE processes $C$ in a tree as the one given in Figure 7; all branches are pruned and no inference-proof solution is found.

The set $C_0$ for this $C$ is

$$\begin{aligned} C_0 \;=\; \{&\neg \mathit{Treat}(\mathsf{Mary}, \mathsf{MedA}) \vee \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids}) \vee \mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer}),\\ &\mathit{Treat}(\mathsf{Mary}, \mathsf{MedA}) \wedge \neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer}),\\ &\neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids})\}. \end{aligned}$$

The closed semantic tree for $C_0$ is pictured in Figure 8. Note that two new levels exist in this semantic tree: one for $\neg \mathit{Ill}(\mathsf{Mary}, \mathsf{Aids})$ and one for $\mathit{Ill}(\mathsf{Mary}, \mathsf{Cancer})$. ♢

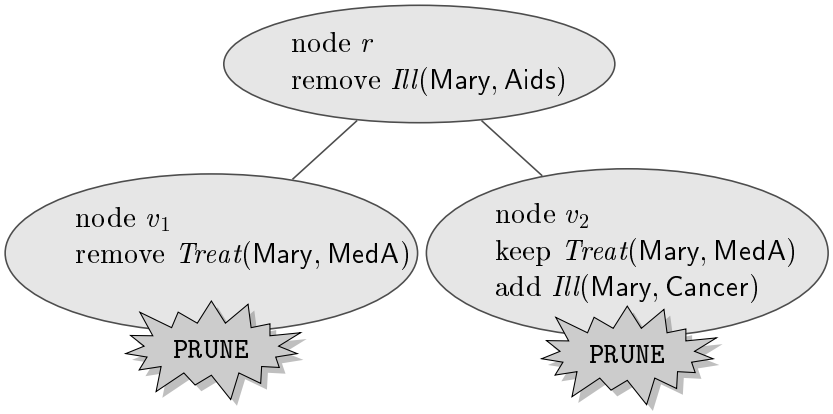


Figure 7: Example for unsatisfiable constraints

Satisfiability completeness now follows from refutation soundness.

**Corollary 9.19 (Satisfiability completeness of *pre*CQE)**
*For a set $C$ of allowed universal constraints, if $C$ is satisfiable, then – after running preCQE to completion – $db_{best}$ is a marked database instance.*

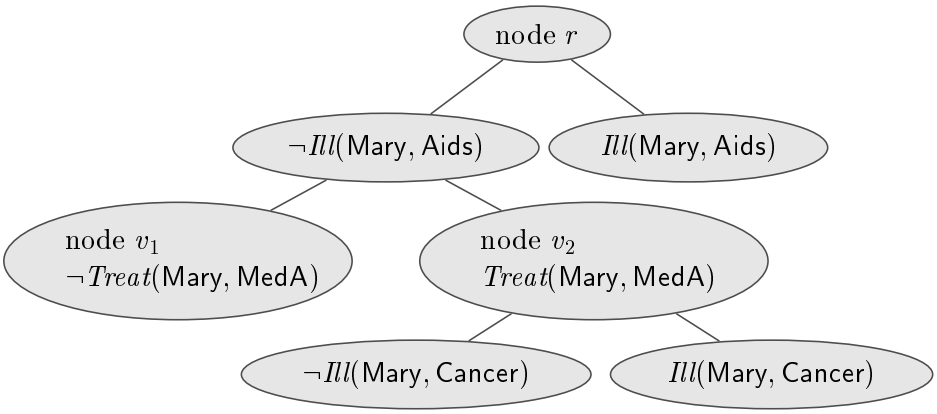


Figure 8: Closed semantic tree

*Proof.* From the fact that $C$ contains only allowed universal formulas and the fact that the set $C$ as a whole is satisfiable we also know that $C$ is DB-satisfiable and thus a solution instance exists. From Theorem 8.10 we know that there also exists a solution instance $db'$ with only *adom* constants such that $I^{db'} \models C$. Now assume that $db_{best}$ is not a marked database instance and thus *undefined*. But then Theorem 9.17 tells us that $C$ has to be unsatisfiable – a contradiction. □

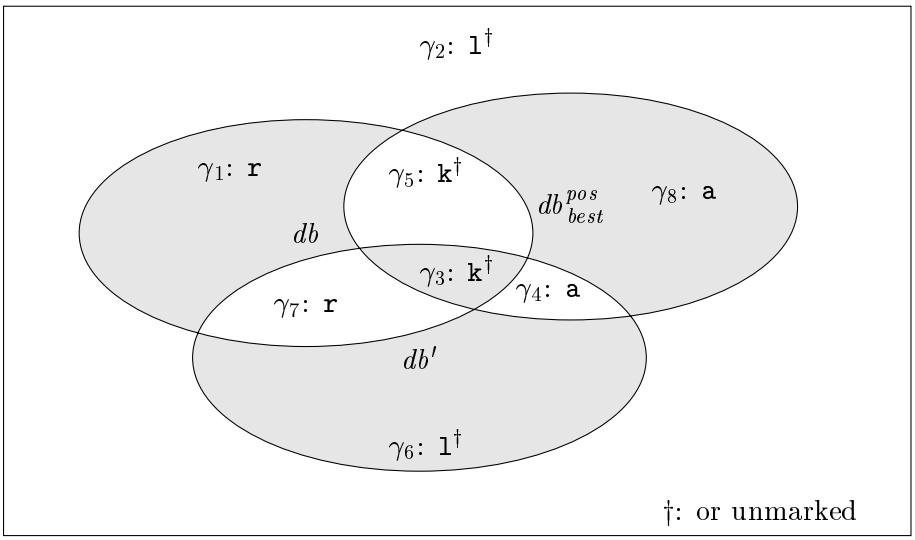


Figure 9: Markers in $db^{pos}_{best}$ for ground atoms with *adom* constants

Last but not least we have to prove that *pre*CQE does not just find some inference-proof solution instance but indeed a distortion-minimal solution; hence the availability requirements are also obeyed as good as possible.

**Theorem 9.20 (Optimality of solution)**
*For a set $C$ of allowed universal constraints, if preCQE finds a solution $db_{best}$, then its positive restriction $db^{pos}_{best}$ is distortion-minimal.*

*Proof.* We show that for an arbitrary inference-proof and distortion-minimal instance $db'$ it holds that

$$db\_dist(db^{pos}_{best}) \leq db\_dist(db') \tag{9}$$

We do this by investigating the influence of the ground atoms with *adom* constants on the distortion distance. The eight different types of *adom* ground atoms are illustrated in Figure 9. There are the following trivial cases where $db_{best}$ and $db'$ coincide and we can easily establish (9):

- $\gamma_1$ and $\gamma_2$ do not cause different distances for $db^{pos}_{best}$ and $db'$: they are contained in neither of the two instances.
- $\gamma_3$ and $\gamma_4$ do not cause different distances for $db^{pos}_{best}$ and $db'$: they are contained in both of the two instances.

The more interesting cases are when there are ground atoms on which $db'$ and $db_{best}$ differ:

- On $\gamma_5$ and $\gamma_6$ $db^{pos}_{best}$ coincides with $db$ while $db'$ does not.
- On $\gamma_7$ and $\gamma_8$ $db'$ coincides with $db$ while $db^{pos}_{best}$ does not.

We will use the *pre*CQE search tree $T$ in the argument. Observe first of all that the local lower bound $min\_lies_v$ for the distortion distance is monotonically nondecreasing along each branch in the tree and counts the number of ground atoms on which the current marked database instance $db_v$ differs from the input instance $db$. There must be a branch in $T$ that corresponds to $db'$: Otherwise $db'$ would deviate into a branch that only exists in the semantic tree $T^*$ constructed from $T$ but not in $T$ itself; but then the single new node in that branch is a failure node – as was shown in Theorem 9.17 – and $db'$ is no inference-proof instance.
We will now show that the distortion distance of $db^{pos}_{best}$ is not worse than the one of $db'$ although literals of type $\gamma_7$ are contained in $db'$and literals of type $\gamma_8$ are not contained in $db'$. $db'$ and $db^{pos}_{best}$ have a common subpath in $T$ starting in the root node $r$ up to a certain node $v'$ ($v'$ itself maybe identical with $r$). After $v'$ they diverge into different branches (the "$db'$-branch" and the "$db^{pos}_{best}$-branch") in $T$ due to a splitting step. Because $db^{pos}_{best}$ is the solution returned by *pre*CQE, the $db^{pos}_{best}$-branch is the only branch in $T$ that is not pruned neither due to conflict nor due to a bad bound value. The $db'$-branch cannot be pruned due to conflict by *pre*CQE because $db'$ is assumed to be inference-proof and thus no conflict occurs along this branch. We analyze how the two branches are positioned in $T$:

1. The first case is that the $db'$-branch is on the left of the $db^{pos}_{best}$-branch; that is, in the splitting either a literal of type $\gamma_5$ was removed in $db'$ while kept in $db^{pos}_{best}$

or a literal of type $\gamma_8$ was left out of $db'$ but added to $db^{pos}_{best}$. Then *pre*CQE must have explored the $db'$-branch before entering the $db^{pos}_{best}$-branch because in a splitting the left child node is treated first. If $db'$ were distortion-minimal, the $db^{pos}_{best}$-branch would be pruned due to a bad bound value and $db'$ (or another distortion-minimal instance that was found earlier in a branch more to the left) would be the solution instance, which contradicts our assumption that $db^{pos}_{best}$ is the solution instance. Hence the case that a distance-minimal solution exists in $T$ on the left of the solution instance does not occur.

2. The second case is that the $db'$-branch is on the right of the $db^{pos}_{best}$-branch; that is, in the splitting either a literal of type $\gamma_7$ was removed in $db^{pos}_{best}$ while kept in $db'$ or a literal of type $\gamma_6$ was left out of $db^{pos}_{best}$ but added to $db'$. The $db^{pos}_{best}$-branch was explored before the $db'$-branch; $db_{best}$ was found as a new optimum with a global upper bound $min\_lies_{best} = db\_dist(db^{pos}_{best})$. But then $db\_dist(db^{pos}_{best})$ is as good as $db\_dist(db')$ because otherwise $db'$ would replace $db^{pos}_{best}$ as the new optimum. Hence, the $db'$-branch must be pruned due to a bad bound as soon as the value of $db\_dist(db^{pos}_{best})$ is reached by the local lower bound in that branch. □

## Summary of Part II

In this part the theoretical basis for *pre*CQE was laid. As the pertinent data model, complete databases and their corresponding DB-interpretations were identified; an evaluation function and the terms "DB-satisfiability" and "DB-implications" were defined. As novel contributions to the field of Controlled Query Evaluation the notions of inference-proofness with respect to a constraint set and distortion minimality were presented to ensure both confidentiality as well as a maximum of availability. Most notably, the precondition for the query-time CQE with lying (that is, the disjunction of the potential secrets is not known a priori) carries over to our case.

For the universal fragment the allowed formulas were specified as syntactic restrictions that guarantee DB-satisfiability of a constraint set. More precisely, only the active domain of the input instance and the constraints is affected in the search for an inference-proof and distortion minimal solution instance.

The listed *pre*CQE algorithm takes advantage of the active domain semantics and the fact that negations of allowed formulas are also allowed and hence yield finite responses. The algorithm makes use of markers and appropriate definitions of an evaluation function and a model operator are adduced. Termination, soundness, completeness and optimality results of the algorithm were established. The proof of refutation soundess reduced the *pre*CQE search tree to a semantic tree for which refutation soundess is a well-known fact.

# III. *pre*CQE for Existential Constraints

## Contents



Whatever *Logic* is good enough to tell me is worth *writing down*,' said the Tortoise. 'So enter it in your note-book, please. We will call it

($E$) If $A$ and $B$ and $C$ and $D$ are true, $Z$ must be true.

Until I've granted *that*, of course I needn't grant $Z$. So it's quite a necessary step, you see?'

'I see,' said Achilles; and there was a touch of sadness in his tone.

– *Lewis Carrol, "What the tortoise said to Achilles"*

# 10 Finite Invention For the Existential Fragment

We now assume that the constraint set $C$ contains constraints in PLNF with only existentially quantified variables. That is, we handle "existential formulas" now:

**Definition 10.1 (Existential formulas)**
*A formula $\Phi = \exists\vec{x}\,\Psi(\vec{x})$ is an* existential formula *iff it is a closed formula in PLNF with existential prenex $\exists\vec{x}$ and $\vec{x}$ consists of all variables occurring freely in the matrix $\Psi(\vec{x})$.*

Again, we include ground formulas in this definition. Intuitively, an existentially quantified variable denotes that there should be at least one substitution of the variable with a constant that makes the matrix of the formula *true*. Based on this, it is a well known fact that the existential fragment of first-order logic has the "finite model property"; that is, if an existential formulas is satisfiable, there is a model with a *finite* domain that satisfies the formulas (see for example [BGG01], Proposition 6.4.27). In fact, in [BGG01] a general result for an existential formula with equality and function symbols is exhibited where terms in a formula can be arbitrarily nested. We will only need their result for pure predicate logic here (where the only terms are constants and variables). More precisely, an existential formula (in pure predicate logic) can be satisfied in a finite domain that has a size of at most the sum of the number of variables and the number of constant symbols occurring in it. Recall from Definition 8.2 that the set of constant symbols occurring in a set $S$ of formulas is called the "active domain" $adom(S)$; we let $vars(S)$ denote the set of variables occurring in $S$.

**Proposition 10.2 (Finite model property of existential formula)**
*A satisfiable existential formula $\Phi = \exists\vec{x}\,\Psi(\vec{x})$ has a model with a finite domain of size $card(adom(\{\Phi\})) + card(vars(\{\Phi\}))$.*

If the actual value is of no interest to the application (for example, when merely testing for satisfiability) often "Skolemization" is used to introduce a new function symbol (or constant symbol) for the existentially quantified variable. This strategy however is not constructive in the sense that the mapping of the new function symbols to values of the domain under consideration is not explicitly stated. Thus, Skolemization is of no direct use for *pre*CQE as the database schema (specifying the fixed domain *dom*) is not changed and both the solution database and the input database are defined over *dom*. Moreover, the distortion distance is computed on ground atoms over *dom*. Explicit values of *dom* for all existentially quantified variables are therefore needed and probably have to be introduced in ground atoms of the solution instance $db'$.
We assume in our system settings (see Section 7.4) that the user is aware first of all of his a priori knowledge *prior* and second of the policy specification *pot_sec*. In this setting, the constant symbols occurring in the constraint set $C$ already have a

particular meaning for the user. Yet, all other constants do not bear such a meaning as they do not occur in the user knowledge. In this sense, the non-$adom(C)$ constants are invariant under isomorphisms from the user's point of view; this property is often called "genericity" (see for example [ST99, GS02]). More precisely, we employ the notion of $adom(C)$-genericity (see [HS94, HS91]; the original term is "$C$-genericity" of a query where $C$ is meant to be a finite set of constants from an infinite domain). We now establish genericity of inference-proof instances with respect to $adom(C)$; distortion minimality will be examined in Corollary 10.5.

**Proposition 10.3 (Genericity of inference-proof instances)**
*For a constraint $C$ and an input instance $db$, inference-proof instances are* $adom(C)$-generic*; that is, for every permutation $\rho$ of the domain $dom$ that is the identity on $adom(C)$, if $db'$ is an inference-proof instance for $db$, then there exists an instance $db''$ that is an inference-proof instance for the input instance $\rho(db)$ (with constant symbols permuted according to $\rho$) and $\rho(db') = db''$.*

As a corollary of Theorem 7.11 and Propositions 10.2 and 10.3 we can now state more precisely what an inference-proof solution instance looks like. Indeed, if a set of existential constraints is satisfiable (in an arbitrary interpretation), it is also DB-satisfiable. Consequently, we only need to apply the CQE precondition for uniform lying (as already used in Theorem 7.11) as a condition for DB-satisfiability of a set of existential constraints; no other restriction – in particular, no syntactic restriction – has to be posed on a set of existential constraints. We need to consider only a finite number of values outside of the active domain of the constraint set $C$ to find a DB-interpretation for $C$: in the worst case, for each variable in a constraint in $C$, one non-$adom(C)$ value has to be chosen in the solution instance (recall that the formulas in $C$ are assumed to be standardized apart). In the context of database queries, this is called "finite invention" in [HS94] and thus if $l$ is the number of variables in $C$, we need at most $l$ invented constants. This is formalized in the following corollary.

**Corollary 10.4 (Inference-proof instances with finite invention)**
*Given a constraint set $C$ of existential formulas and assuming prior $\not\models_{DB}$ pot_sec_disj, there exists an inference-proof solution instance that contains only constants from the set $adom(C) \cup invent$, where $invent \subset (dom \setminus adom(C))$ of cardinality $l$ and $l := card(vars(C))$ is the number of distinct variables occurring in $C$.*

*Proof.* All constants in *invent* are generic by construction. That is, there is a model of $C$ with finite domain $adom(C) \cup invent$ (of size $card(adom(C)) + card(vars(C))$). We keep the interpretation of predicates in the finite model as the interpretation over the infinite domain $dom$. Hence, we have a DB-interpretation for $C$. We easily see that because the finite model satisfies $C$, the DB-interpretation satisfies $C$, too: by construction it holds that $adom(C) \cup invent \subset dom$ and if the finite model

interprets predicate symbols with relations over constants from $adom(C) \cup invent$ such that all formulas in $C$ are satisfied, then the DB-interpretation interprets them the same. □

When looking for an upper bound of the distortion distance, we can define it based on the DNF representations of all constraint formulas and use finite invention for the existentially quantified variables: for a formula $\Phi$ in the constraint set $C$, at least one disjunct $\Delta_i$ of $dnf(\Phi)$ has to be satisfied. For an upper bound for the distortion distance *db_dist* we can count the number of literals in each disjunct, take the maximum and sum up over all formulas in $C$. That is, in the worst case all literals in the longest disjunct of each formula swap their truth values in the solution instance.

---

**Corollary 10.5 (Upper bound of distortion distance)**
*For a constraint set $C$ of existential formulas, the distortion distance for any inference-proof and distortion-minimal database instance $db'$ can be bounded by*

$$db\_dist(db') \leq \sum_{\Phi \in C} \Big( \max_{\Delta_i \in dnf(\Phi)} card(\{\ \Lambda_{ij}\ |\Lambda_{ij}\ \textit{is a literal in}\ \Delta_i\})\Big)$$

---

As a tiny illustration take the constraint set $C = \{\neg Ill(\mathsf{Mary}, \mathsf{Aids}), \exists x\, Ill(x, \mathsf{Aids})\}$ and the input instance $db = \{Ill(\mathsf{Mary}, \mathsf{Aids})\}$: we see that both constraints are in DNF with a single disjunct each. Both these disjuncts contain one literal; this gives an upper bound of 2. And indeed, $Ill(\mathsf{Mary}, \mathsf{Aids})$ has to be removed, and then a new constant for $x$ – for example Pete – has to be invented and $Ill(\mathsf{Pete}, \mathsf{Aids})$ has to be added to result in the solution instance $db' = \{Ill(\mathsf{Pete}, \mathsf{Aids})\}$.

We have to concede that instantiations with *invent* constants are not sufficient to achieve distortion minimality. This influences the way how a solution instance can be found. A justification is given in the following remark.

*Remark 10.6 (Finite invention and distortion minimization):* We have to examine more closely how finite invention influences the distortion distance. In some cases, instantiations with invented constants lead to distortion-minimality. For instance, for

$$db = \{Ill(\mathsf{Mary}, \mathsf{Flu}), Ill(\mathsf{Mary}, \mathsf{Cancer}), Ill(\mathsf{Mary}, \mathsf{Myopia})\},$$

and

$$C = \{\exists x\, (Ill(x, \mathsf{Aids}) \wedge Ill(x, \mathsf{Flu}) \wedge \neg Ill(x, \mathsf{Cancer}) \wedge \neg Ill(x, \mathsf{Myopia}))\}$$

with $adom(C) \cap Name = \emptyset$ and $adom(db) \cap Name = \{\mathsf{Mary}\}$. Satisfying $C$ by instantiating $x$ with Mary results in $db'_1 = \{Ill(\mathsf{Mary}, \mathsf{Aids}), Ill(\mathsf{Mary}, \mathsf{Flu})\}$ and distortion distance 3. In contrast, assuming that $\mathsf{Pete} \in invent$, the solution instance $db'_2 = \{Ill(\mathsf{Pete}, \mathsf{Aids}), Ill(\mathsf{Pete}, \mathsf{Flu})\}$ has distortion distance 2.
Unfortunately, if just choosing values outside of the active domain of $C$ for all existentially quantified variables, the distortion distance might be worse than the

possible optimum, too. One can see that the constants in the database instance $db$ (more precisely, constants in $adom(db) \setminus adom(C)$) have to be taken into account when looking for a distortion-minimal solution. Consider the following example:

$$db = \{\, Treat(\mathsf{Mary}, \mathsf{MedA})\}$$

and

$$C = \{\exists x\, Ill(x, \mathsf{Aids}),\ \ Treat(\mathsf{Mary}, \mathsf{MedA}) \rightarrow Ill(\mathsf{Mary}, \mathsf{Aids})\}$$

with $adom(C) \cap Name = \{\mathsf{Mary}\}$. Adding $Ill(\mathsf{Pete}, \mathsf{Aids})$ (in order to satisfy the first constraint) would not avoid the addition of $Ill(\mathsf{Mary}, \mathsf{Aids})$ (in order to satisfy the second constraint); this solution has a distortion distance of 2. In contrast, adding $Ill(\mathsf{Mary}, \mathsf{Aids})$ for the first constraint already satisfies the second constraint and results in a distortion distance of 1.
Consequently, we see that in terms of distortion minimality $pre$CQE is indeed $adom$-generic (recall that $adom$ is short for $adom(C \cup db)$). More formally, if $db'$ is an inference-proof and distortion minimal solution instance for the input instance $db$, for every permutation $\rho$ of the domain that is the identity on $adom$, $\rho(db')$ is an inference-proof and distortion minimal instance for $db$, too. Hence, we have to check not only $adom(C)$ constants but also $adom(db)$ constants as instantiations (in addition to finite invention) for existentially quantified variables in order to find a distortion-minimal solution. The same technique is used as a rule in the Extended Positive (EP) tableaux system [BT98, BEST98] that checks satisfiability of a set of formulas (without distance minimization). This rule makes the EP tableaux system complete for finite satisfiability in an infinite domain. A similar approach can be found in [HS94] to show equivalence of several query semantics in an infinite domain. ♢

Lastly, we remark on a crucial aspect of DB-interpretations that is based on the fact that DB-interpretations have a finite positive part but the database domain $dom$ is infinite and fixed.

*Remark 10.7 (Tautologies in infinite domain):* A peculiarity of DB-interpretations is that a negative literal containing an existentially quantified variable is always satisfied in an infinite domain: A formula like $\exists x\, \neg P(a, x)$ is a tautology because there are infinitely many constants $b$ in the domain $dom$ for which $\neg P(a, b)$ holds. More generally, we can again analyze the DNF representation of an existential formula $\Phi$ and come up with a sufficient condition for its being tautologous: if in $cnf(\Phi) = \exists \vec{x}(\Delta_1 \vee \ldots \vee \Delta_m)$ there is a disjunct $\Delta_i = \Lambda_{i1} \wedge \ldots \wedge \Lambda_{in_i}$ where each $\Lambda_{ij}$ is a negative literal and contains a variable from $\vec{x}$, $\Phi$ is indeed satisfied in any DB-interpretation. We can exclude those tautologous formulas from the constraint set $C$: performance of $pre$CQE is improved when it does not have to check tautologous formulas that are satisfied by default over and over again; not excluding such formulas has however no impact on correctness of the algorithm. For other kinds of tautology removal see Section 14.1. ♢

# 11 *pre*CQE for Existential Quantification

After the theoretical considerations in Section 10, we now approach an algorithmic description for finding an inference-proof database instance for existential constraints. Violated existential constraints can again be identified as in Definition 9.6: evaluating a violated constraint in $db_v$ (as a closed formula) returns its negation.

To be able to find a distortion-minimal solution instance, violated existential constraints must be treated by several alternative instantiations of the existentially quantified variables. More precisely, we need the notion of active domain in a node $v$ of the *pre*CQE search tree; it is defined as the set of constants occurring in the constraint set $C_v$ or in the marked database $db_v$.

**Definition 11.1 (Active domain in a node $v$)**
*The* active domain *in a node $v$ of the search tree is*

$$adom_v := adom(C_v) \cup adom(db_v)$$

When instantiating an existentially quantified variable in a node $v$, first of all, all elements of the active domain $adom_v$ are tried as instantiations in different branches of the search tree. Finally, in a further branch, one constant symbol outside of $adom_v$ is chosen. Note that the constraint set is now modified upon traversal of the search tree due to instantiations; hence, the active domain $adom(C_v)$ can change from node to node due to the addition of constants chosen from the set *invent*. When handling existentially quantified variables from left to right in the order in which they appear in the quantifier prefix, instantiating one existentially quantified variable with all values of $adom_v$ plus one new *invent* constant corresponds to the creation of $card(adom_v) + 1$ child nodes in the search tree. In each child node, the partially instantiated constraint replaces the original – however, no ground atoms are marked in this step yet. To distinguish these child node creations from the splitting that marks a ground atom, we call this process "case differentiation" from now on. Let us look at an example for such a case differentiation. Figure 10 shows a search tree for Example 11.2 where `l` stands for a "leave" and `a` stands for an "add" marker as before.

*Example 11.2:* We again assume that there are sorts for the variables as were introduced in Example 7.5. Indeed, the notion of active domain carries over to the sorts again as we already saw in Example 8.15. We start the search with an empty input instance $db = \emptyset$ and the constraint set

$$\begin{aligned} C \;=\; \{ & \neg Ill(\mathsf{Pete}, \mathsf{Aids}), \\ & \neg Ill(\mathsf{Mary}, \mathsf{Cancer}) \wedge \neg Ill(\mathsf{Mary}, \mathsf{Aids}), \\ & \exists x\, (Ill(x, \mathsf{Aids}) \vee Ill(x, \mathsf{Cancer})) \qquad \} \end{aligned}$$

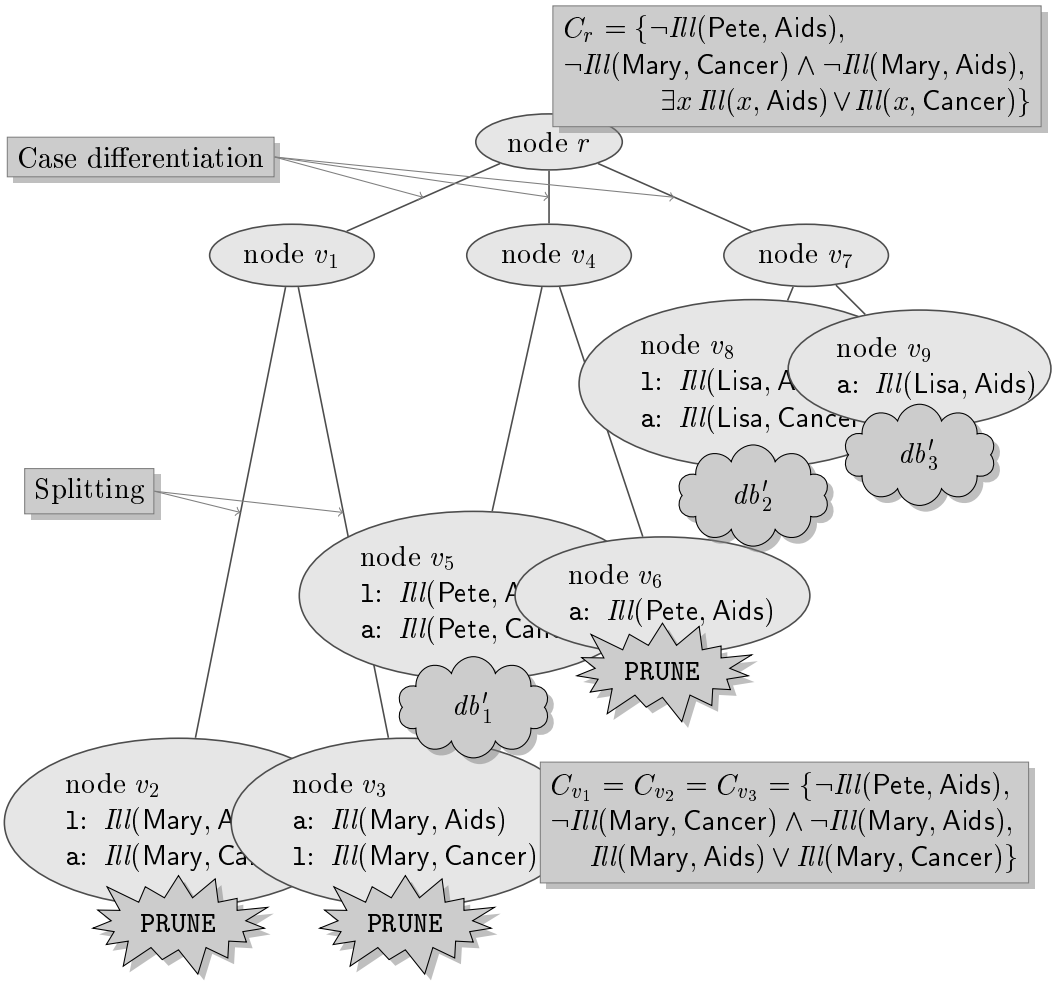


Figure 10: Example for finite invention

Note that the constraints contain only one existentially quantified variable $x$ of the sort *Name*; for the search we need only an invention of cardinality 1 and we assume here that $invent = \{\text{Lisa}\} \subset dom$. In the root node $r$, the existential formula is the only violated constraint and $adom_r \cap Name = \{\text{Mary}, \text{Pete}\}$. Thus, in the root node we start with a case differentiation of size 3. That is, the existentially quantified variable $x$ is instantiated with the $adom_r$ constant Mary in node $v_1$, with the $adom_r$ constant Pete in node $v_4$ and with the *invent* constant Lisa in $v_7$. The constraint sets in these nodes are changed accordingly: the instantiated constraint replaces the original constraint. Figure 10 shows the constraint sets $C_r$ and $C_{v_1}$ (which is equal to the sets $C_{v_2}$ and $C_{v_3}$ later on).
Splitting on ground atoms still has to be executed to determine the truth values for ground atoms and find a distortion-minimal solution instance. Solution candidates are the inference-proof instances $db'_1 = \{Ill(\text{Pete}, \text{Cancer})\}$, $db'_2 = \{Ill(\text{Lisa}, \text{Cancer})\}$

and $db'_3 = \{Ill(\mathsf{Lisa}, \mathsf{Aids})\}$. They each have a distortion distance of 1. ♢

## 11.1 The *pre*CQE Algorithm

We now move on to a description of the procedures for *pre*CQE with existential constraints as presented in Listings 6 to 11.

`INIT` **(Listing 6)** remains unaltered from the previous `INIT` procedure: the root node is created and all local values are initialized; then, `GROUND` is executed.

`GROUND` **(Listing 7)** still determines the set of violated constraints $C_v^{vio}$ (Line 7.1.) and sets the new optimum whenever there are no more violated constraints (Line 7.2.). After these two steps, `GROUND` changes its functionality radically from the universal case. First of all, formulas in the set $C_v^{vio}$ are simplified (Lines 7.3.1. and 7.3.2.; see below). Simplified formulas are saved in the set $C_v^{simp}$; we use this temporary set because simplifying $C_v^{vio}$ directly would modify formulas that will be needed in the `CASE` procedure. Moreover, we cannot simplify formulas in $C_v$ directly because this would alter the active domain of the constraints which will be needed in the `CASE` procedure, too. $C_v^{simp}$ is directly checked for violated ground formulas in which all literals are marked (Line 7.3.3.); if such a ground formula exists, we cannot satisfy it anymore in the current marked database instance in node $v$. Accordingly, the current branch is pruned due to a conflict in the current partial interpretation.
The next step (Line 7.3.4.) covers the case that after simplification there is a violated ground formula that contains only one unmarked literal. In this case we only try setting this literal to *true* (Line 7.3.4.2.) and then recur to `GROUND`.
If there are no such ground formulas but there is indeed an unmarked literal in some formula in $C_v^{simp}$ (see Line 7.3.5.), we try to expand the current database instance $db_v$ with a value for this literal: a `SPLIT` is executed (Line 7.3.5.1.) that creates two child nodes of the node $v$ (see below).
If none of the above cases holds, our last option is to execute a `CASE` (Line 7.3.6.1.) that removes an existential quantifier and creates several child nodes for it (see below). Note that the previous `if`-conditions ensure that $C_v^{simp}$ is not empty and there are no unmarked ground literals left; that is, there must be a literal containing an existentially quantified variable.

6. `INIT`: **Initialization** for root node $r$
   6.1. create root node $r$
   6.2. $db_r := db$;
   6.3. $min\_lies_r := 0$;
   6.4. $C_r := C$;
   6.5. $db_{best} := undefined$;
   6.6. $min\_lies_{best} := \infty$;
   6.7. `GROUND`$(r)$;

Listing 6: *pre*CQE – Initialization for existential formulas

7. GROUND($v$): Check **ground formulas** in node $v$
   7.1. $C_v^{vio} := \{\Phi \in C_v \mid eval_v(\Phi) = \neg\Phi\}$;
   7.2. `if` $(C_v^{vio} = \emptyset)$
      7.2.1. $db_{best} := db_v$;
      7.2.2. $min\_lies_{best} := min\_lies_v$;
   7.3. `else`
      7.3.1. $C_v^{simp} := C_v^{vio}$
      7.3.2. SIMP($v$);
      7.3.3. `if` ( there is ground $\phi \in C_v^{simp}$ with $card(unmarked_v(\phi)) = 0$ )
         7.3.3.1. PRUNE; //(conflicting markers)
      7.3.4. `else if` ( there is ground $\phi \in C_v^{simp}$ with $card(unmarked_v(\phi)) = 1$ )
         7.3.4.1. take unique literal $\lambda \in unmarked_v(\phi)$;
         7.3.4.2. MARK($v$,$\lambda$);
         7.3.4.3. GROUND($v$);
      7.3.5. `else if` ( there is $\Phi \in C_v^{simp}$ with $card(unmarked_v(\Phi)) > 0$ )
         7.3.5.1. SPLIT($v$);
      7.3.6. `else`
         7.3.6.1. CASE($v$);

Listing 7: *pre*CQE – Check ground formulas

8. SIMP($v$): **Simplification** of violated constraints
   8.1. `repeat` until no more changes occur:
      8.1.1. `foreach` subformula $\Phi'$ of a formula $\Phi \in C_v^{simp}$
         8.1.1.1. `if` $(\Phi' = \Psi \wedge \gamma$ and $marker_v(\gamma) \in \{\mathtt{k}, \mathtt{a}\}$
                `or` $\Phi' = \Psi \wedge \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{r}, \mathtt{l}\}$
                `or` $\Phi' = \Psi \vee \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{k}, \mathtt{a}\}$
                `or` $\Phi' = \Psi \vee \gamma$ and $marker_v(\gamma) \in \{\mathtt{r}, \mathtt{l}\})$
            8.1.1.1.1. replace $\Phi'$ with $\Psi$; //(unit resolution)
         8.1.1.2. `if` $(\Phi' = \Psi \wedge \gamma$ and $marker_v(\gamma) \in \{\mathtt{r}, \mathtt{l}\}$
                `or` $\Phi' = \Psi \vee \gamma$ and $marker_v(\gamma) \in \{\mathtt{k}, \mathtt{a}\})$
            8.1.1.2.1. replace $\Phi'$ with $\gamma$; //(unit subsumption)
         8.1.1.3. `if` $(\Phi' = \Psi \wedge \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{k}, \mathtt{a}\}$
                `or` $\Phi' = \Psi \vee \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{r}, \mathtt{l}\})$
            8.1.1.3.1. replace $\Phi'$ with $\neg\gamma$; //(unit subsumption)
         8.1.1.4. if a variable disappeared from $\Phi$ remove its quantifier from the prenex

Listing 8: *pre*CQE – Simplification for existential formulas

9. SPLIT($v$): **Splitting** on a ground atom in node $v$
   9.1. choose $\Phi \in C_v^{simp}$ with $card(unmarked_v(\Phi)) > 0$;
   9.2. choose $\lambda \in unmarked_v(\Phi)$;
   9.3. generate two child nodes $v_{left}$ and $v_{right}$;
   9.4. $db_{v_{left}} := db_{v_{right}} := db_v$;
   9.5. $C_{v_{left}} := C_{v_{right}} := C_v$;
   9.6. $min\_lies_{v_{left}} := min\_lies_{v_{right}} := min\_lies_v$;
   9.7. MARK($v_{left}$,$\neg|\lambda|$);
   9.8. GROUND($v_{left}$);
   9.9. MARK($v_{right}$,$|\lambda|$);
  9.10. GROUND($v_{right}$);

Listing 9: *pre*CQE – Split for existential formulas

10. `CASE(`$v$`)`: **Case differentiation** in node $v$
    10.1. choose $\exists x \Phi(x) \in C_v^{vio}$ where $x$ also occurs in $C_v^{simp}$;
    10.2. `foreach` $a \in adom_v$
        10.2.1. generate child node $v_a$;
        10.2.2. $db_{v_a} := db_v$;
        10.2.3. $C_{v_a} := C_v \cup \{\Phi(a)\} \setminus \{\exists x \Phi(x)\}$;
        10.2.4. $min\_lies_{v_a} := min\_lies_v$;
        10.2.5. `GROUND(`$v_a$`)`;
    10.3. choose $a' \in invent \setminus adom_v$;
        10.3.1. generate child node $v_{a'}$;
        10.3.2. $db_{v_{a'}} := db_v$;
        10.3.3. $C_{v_{a'}} := C_v \cup \{\Phi(a')\} \setminus \{\exists x \Phi(x)\}$;
        10.3.4. $min\_lies_{v_{a'}} := min\_lies_v$;
        10.3.5. `GROUND(`$v_{a'}$`)`;

Listing 10: *pre*CQE – Case differentiation for existential formulas

11. `MARK(`$v$`,`$\lambda$`)`: **Marking** an unmarked ground atom $\gamma$ in $db_v$
    11.1. $\gamma := |\lambda|$;
    11.2. `if (`$\lambda = \gamma$ `and` $eval_v(\gamma) = \gamma$ `)`
        11.2.1. $marker_v(\gamma)$ :=`k;`
    11.3. `else if (`$\lambda = \gamma$ `and` $eval_v(\gamma) = \neg\gamma$ `)`
        11.3.1. $marker_v(\gamma)$ :=`a;`
        11.3.2. $min\_lies_v$`++;`
    11.4. `else if (`$\lambda = \neg\gamma$ `and` $eval_v(\gamma) = \gamma$ `)`
        11.4.1. $marker_v(\gamma)$ :=`r;`
        11.4.2. $min\_lies_v$`++;`
    11.5. `else if (`$\lambda = \neg\gamma$ `and` $eval_v(\gamma) = \neg\gamma$ `)`
        11.5.1. $marker_v(\gamma)$ :=`l;`
    11.6. `if (`$min\_lies_v \geq min\_lies_{best}$`)` `PRUNE; //`(bad bound)

Listing 11: *pre*CQE – Marking for existential formulas

`SIMP` **(Listing 8)** is only slightly changed from *pre*CQE for universal constraints: not the violation set $V_v$ (which is not computed in the existential case) but instead the violated constraints in $C_v^{vio}$ are simplified. $C_v^{simp}$ is the set that gathers all simplified violated constraints. Simplification here not only applies to ground formulas but instead existential quantifiers and variables occur. But still only ground unit resolution and subsumption based on marked ground atoms in the constraint formulas is executed; non-ground atoms are not affected.

`SPLIT` **(Listing 9)** chooses one simplified non-ground violated formula with at least one unmarked literal (see Line 9.1.) and then chooses one of the unmarked literals of that formula (Line 9.2.). Two child nodes (Line 9.3.) are created to try both truth values for the literal (Lines 9.7. and 9.9.). In both cases, a recursion to `GROUND` takes place.

`CASE` **(Listing 10)** instantiates one existentially quantified variable $x$ of a formula in the set of violated constraints (Line 10.1.); $x$ need not be the first existentially quantified variable in the prenex: there may occur other variables in front of $x$. Note that we restrict instantiation to those variables $x$ in a formula $\exists x\, \Phi(x)$ that also occur in the simplified formula in $C_v^{simp}$ because those variables that are simplified away do not influence the satisfiability of the violated formula. For instantiation however we take the unsimplified formula from $C_v^{vio}$ to be able to delete this formula from $C_v$. First of all, child nodes where the variable is instantiated with all possible constants of $adom_v$ are created (see Lines 10.2. to 10.2.5.). Afterward (Lines 10.3. to 10.3.5.) one *invent* constant that is not yet contained in $adom_v$ is chosen as the instantiation. In both cases (Lines 10.2.3. and 10.3.3.) the original existential constraint is removed from the constraint set $C_v$ and the instantiated constraint is added to $C_v$ instead; in other words, the instantiated constraint subsumes the original constraint: once the instantiation is satisfied, the original constraint is also satisfied. In every child node, `GROUND` is called (Lines 10.2.5. and 10.3.5.).

`MARK` **(Listing 11)** is unchanged from *pre*CQE for universal constraints: ground atoms of ground literals are marked depending on their evaluation in the current database instance.

## 11.2 Termination, Soundness and Completeness of *pre*CQE

We now analyze the properties of the above algorithm. At first, we convince ourselves that no infinite recursion occurs and the *pre*CQE algorithm for existential constraints terminates.

**Theorem 11.3 (Termination of *pre*CQE)**
*For a set $C$ of existential constraints, preCQE terminates in a finite amount of time.*

*Proof.* We observe that $C_v$ is always a finite set in every node $v$ because $C$ is finite and each time an instantiated constraint is added, the original constraint is deleted from the set $C_v$ (see Lines 10.2.3. and 10.3.3.). Obviously, $C_v^{vio}$ is a finite subset of $C_v$. This is why the `SIMP` procedure on $C_v^{simp}$ (in analogy to the universal case) takes only finite time: the repetition of the simplification of formulas (see Line 8.1.) only takes finite time because, whenever an application is possible, the length of one of the formulas decreases (see Lines 8.1.1.1.1., 8.1.1.2.1. and 8.1.1.3.1.).
Next we observe that in `CASE` the number of generated child nodes (Lines 10.2.1. and 10.3.1.) is finite: the active domain of a constraint set $C_v$ is not altered in any other procedure than `CASE`; after a `CASE` step, $adom_v$ contains at most one new constant taken from *invent*. Hence we can establish that in any node $v$, $adom_v \subseteq (adom \cup invent)$. That is, the number of generated child nodes is bounded by $card(adom) + card(invent)$. Note that `SPLIT` still creates just two child nodes. All other steps in the six procedures consist of finite instructions.
We still have to show that the recursion in *pre*CQE is bounded and with it the depth of the *pre*CQE search tree. As in the universal case, the recursion is only stopped in the following three cases (which correspond to leaves in the search tree):

1. a (new) optimum is found (Line 7.2.)
2. the first pruning condition arises: the current partial interpretation contains a conflict (see Line 7.3.3.1.)
3. the second pruning condition arises: a better solution than the current partial interpretation has already been found (see Line 11.6.)

In the existential case however, not only `GROUND` and `SPLIT` are involved in the recursion, but also `GROUND` calls `CASE` (Line 7.3.6.1.) and `CASE` recurs to `GROUND` (Lines 10.2.5. and Lines 10.3.5.). We first argue that there can only be a finite number of recursive calls to `CASE`: `CASE` is only called if there is at least one non-ground existentially quantified formula left in $C_v^{vio}$ (see Line 7.3.6.1.); in each of the child nodes generated in `CASE`, the new node's constraint set contains one existentially quantified variable less than the constraint set $C_v$ had before (see Lines 10.2.3. and 10.3.3.). That is, there can be at most $l$ recursive calls to `CASE` (where again $l$ is the number of existentially quantified variables in $C$).
Next we apply the argument that each `GROUND` call is preceded by a `MARK` operation (either in Line 7.3.4.2. or in Lines 9.7. and 9.9.). The ground literals that are marked are either already contained in $C$ or are obtained by instantiating existentially quantified variables with exactly one constant. That is, the number of `MARK` calls is bounded by the number of atoms occurring in $C$. Hence, for

$$k' := card(\{\ \Gamma \mid \Gamma \text{ is an atom in a formula in } C\}),$$

the length of a branch in the *pre*CQE search tree is bounded by $l + k'$.
Note that with $k'$ we do not count multiple occurrences of atoms in $C$ while still assuming that variables are standardized apart. □

Based on this termination result and analogously to the universal fragment, we have to make sure that the *pre*CQE algorithm is satisfiability sound and complete: that is, the marked database instance that the *pre*CQE algorithm finds is really an inference-proof solution and if there exists such a solution, the algorithm actually finds a marked database. Still $db_{best}$ is either a marked database instance or $db_{best}$ is *undefined*. Again, the conceptual counterpart is to analyze refutation soundness and completeness.
For satisfiability soundness nothing changes in comparison to the universal case.

**Theorem 11.4 (Satisfiability soundness of *pre*CQE)**
*For a set $C$ of existential constraints and after running preCQE to completion, if $db_{best}$ is a marked database instance, then its positive restriction is an inference-proof database instance.*

*Proof.* See Theorem 9.10. □

Again, refutation completeness follows from satisfiability soundness.

**Corollary 11.5 (Refutation completeness of *pre*CQE)**
*For a set $C$ of existential constraints, if $C$ is unsatisfiable, then – after running preCQE to completion – $db_{best}$ is undefined.*

To show refutation soundness we employ Corollary 10.4.

**Theorem 11.6 (Refutation soundness of *pre*CQE)**
*For a set $C$ of existential constraints and after running preCQE to completion, if $db_{best}$ is undefined, then $C$ is unsatisfiable.*

*Proof.* Using Corollary 10.4 and the general results on Skolemization, we see that $C$ is unsatisfiable if and only if the set of ground formulas obtained by instantiating each existentially quantified variable with a unique constant from the set *invent* is unsatisfiable. Without loss of generality, we assume that the mapping of existentially quantified variables to *invent* constants is fixed before starting *pre*CQE; that is, we determine a one-to-one mapping $\sigma$ with

$$\sigma : vars(C) \to invent$$

where $vars(C)$ is the set of variables occurring in $C$. Let $C_0$ be the set of constraint formulas instantiated with $\sigma$ values (inclusive of all ground formulas of $C$):

$$C_0 := \{\ \Phi(\sigma(x_1), \ldots, \sigma(x_n)) \mid \exists x_1, \ldots, x_n\, \Phi(x_1, \ldots, x_n) \in C\}$$

Let now $T$ be the *pre*CQE search tree, obtained from a run of *pre*CQE using $\sigma$; that is, the last child node created in a `CASE` operation (Line 10.3.1.) is now assumed to instantiate the case variable $x$ with $\sigma(x)$. We now show that again we can construct a semantic tree $T^*$ from $T$ such that each node of $T^*$ is a failure node for $C_0$. This

can be achieved along the lines of the proof of Theorem 9.17. Here we are interested in (un-)satisfiability of the constraints for which consideration of *invent* constants as instantiations for existentially quantified variables suffices; that is, we disregard instantiations with *adom* constants at once as they only play a role in distortion minimization.

To begin with, we use the same construction as in Definition 9.16 but we add a step to the construction that removes all those nodes from $T^*$ that were created in a `CASE` step – we will call these node "case nodes" in the following. More precisely, whenever a set of sibling case node is encountered upon construction of $T^*$, we only keep the rightmost of these sibling, where the case variable $x$ is instantiated with $\sigma(x)$ – we will call this node the "invention node". All other case nodes (including all the subtrees below them) are immediately removed. Then, the construction of $T^*$ is continued in the invention node. When $T^*$ is fully expanded, we traverse $T^*$ once again and remove all invention nodes (note that invention nodes do not have any siblings in $T^*$), but keep the subtrees below them: we append all child nodes of an invention node to the parent of the invention node; if an invention node has an invention node as its child we also remove the child invention node.

After removal of invention nodes, $T^*$ is indeed a semantic tree obeying Definition 9.14: The nodes that were added upon construction of $T^*$ (due to `GROUND`) and the splitting nodes still are of binary structure; the two siblings are labeled in such a manner that one of them is labeled with a literal and the only sibling is labeled with the literal's conjugate. By moving these nodes up one or more levels (when removing invention nodes), we regain a binary structure of $T^*$. Moreover, we still can be sure that no path contains labels with a complementary pair of literals because labels are based on the markers that are set in $T$ and `MARK` is only called for unmarked literals.

We will now prove refutation soundness by arguing that in the semantic tree $T^*$ every leaf node $v^*$ is a failure node for $C_0$; that is, $C_0$ is unsatisfiable and hence $C$ also is. As was already analyzed in Theorem 9.17 either $v^*$ is also a leaf in $T$ or $v^*$ is a new level node of some literal $\lambda$; the only difference is that for existential constraints the marked literals are taken from a formula in $C_v^{vio}$ (and not from a violation set $V_v$).

1. If $v^*$ is identical to a leaf $v$ in $T$, then $v$ was pruned in $T$ because there is a conflicting ground formula $\phi \in C_v^{vio}$ (after simplification) such that $card(unmarked_v(\phi)) = 0$. This pruning occurred as a consequence of a `MARK` operation for a literal $\lambda^*$ in $v$ that makes $\lambda^*$ *true* in $v$; but then, by construction of $T^*$, $v^*$ is labeled with $\lambda^*$ and $\lambda^*$ is also *true* in $v^*$. Moreover, all labels on the path from the root of $T^*$ to $v^*$ correspond to the markers in $T$. Hence $v^*$ falsifies $\phi$ in $T^*$.

   Let $\phi^*$ be the ground formula from which $\phi$ was obtained by simplification. Then, $v^*$ also falsifies $\phi^*$ because simplification is (un-)satisfiability-preserving for a given (partial) interpretation. Lastly note that $\phi^*$ indeed is a formula of $C_0$ because $T^*$ contains only $\sigma$-instantiated formulas by construction.

2. If $v^*$ is not contained in $T$, then it is a leaf in a new level of a literal $\lambda^*$ in $T^*$. But $\lambda^*$ was marked (as *true*) in a node $v$ in $T$ because it was chosen from a ground formula $\phi \in C_v^{vio}$ (after simplification) with $card(unmarked_v(\phi)) = 1$. This node $v$ is then an ancestor of $v^*$ in $T^*$. By construction, all ancestors of $v^*$ in $T^*$ are labeled according to the markers in $T$.

   - If $v^*$ is labeled with $\overline{\lambda^*}$, then $\lambda^*$ is *false* in $v^*$. We convince ourselves that $v^*$ falsifies $\phi$: $\lambda^*$ is the only unmarked literal of $\phi$ and all other marked literals do not satisfy $\phi$, and thus $\phi$ is indeed *false* in $v^*$.
     Analogously to the previous case, let $\phi^*$ be the formula from which $\phi$ was obtained by simplification. Then, $v^*$ also falsifies $\phi^*$ because of satisfiability-preservation. And again, $\phi^*$ indeed is a formula of $C_0$ by construction.

   - If $v^*$ is labeled with $\lambda^*$ then $v$ was pruned in $T$ immediately after $\lambda^*$ was marked in $v$ because otherwise $v^*$ would not be a leaf in $T^*$. But then again, there is a violated ground formula $\phi' \in C_v^{vio}$ for which $card(unmarked_v(\phi')) = 0$. Hence, $v^*$ falsifies $\phi'$ and also $\phi'^*$ if $\phi'^*$ is the formula from which $\phi'$ was obtained by simplification.

No ancestor of $v^*$ in $T^*$ falsifies a formula in $C_0$, because otherwise pruning would have occurred earlier in $T$. These are all cases that have to be considered and thus $T^*$ falsifies $C_0$. By Herbrand's Theorem the claim of the theorem follows. □

From refutation soundness follows satisfiability completeness with the help of the previous termination result.

---

**Corollary 11.7 (Satisfiability completeness of *pre*CQE)**
*For a set $C$ of existential constraints, if $C$ is satisfiable, then – after running preCQE to completion – $db_{best}$ is a marked database instance.*

---

Lastly, we manifest that *pre*CQE for existential constraints returns a distortion-minimal solution.

---

**Theorem 11.8 (Optimality of solution)**
*For a set $C$ of existential constraints, if preCQE finds a solution $db_{best}$, then its positive restriction $db_{best}^{pos}$ is distortion-minimal.*

---

*Proof.* We have seen in Theorem 9.20 that splitting and the marking in `GROUND` as well as both of the pruning conditions do not miss out a solution. For existential formulas, the two differences are

- that `GROUND` marking as well as conflict pruning are now executed on ground formulas in $C_v^{vio}$.
- that splitting is also executed on non-ground formulas.

Yet, this does not influence the optimality of the solution.
The crux for existential formulas is thus that the `CASE` operation tries out all relevant non-isomorphic instantiations of variables: all constants from $adom_v$ plus one new *invent* constant. □

## 12 ∀∃-Quantified Constraints

We now come to a more general constraint set: $C$ contains closed formulas with a prenex consisting of a sequence of universally quantified variables followed by a sequence of existentially quantified variables. The combination of ∀ and ∃ quantifiers has some undesirable consequences: even when restricted to a set of allowed formulas, this fragment contains formulas that are not satisfiable in an interpretation with a finite positive part – these are the so called "infinity axioms". Such a case of a non-DB-satisfiable constraint set is illustrated in the following example; a similar example was used in [BEST98].

*Example 12.1:* The following constraint set describes an employment hierarchy; it can only be satisfied in an infinite model.

$$\begin{aligned} C \quad = \{ \quad & \forall x(\mathit{Employee}(x) \rightarrow \exists y\, \mathit{Boss}(x,y)), \\ & \forall xy(\mathit{Boss}(x,y) \rightarrow \mathit{Employee}(x) \wedge \mathit{Employee}(y)), \\ & \forall xyz(\mathit{Boss}(x,y) \wedge \mathit{Boss}(y,z) \rightarrow \mathit{Boss}(x,z)), \\ & \forall x(\neg \mathit{Boss}(x,x)), \\ & \mathit{Employee}(\mathsf{Pete}) \quad \} \end{aligned}$$

We observe that each of the formulas has the allowed property: all universally quantified variables appear in a negated literal in each conjunct of the CNF representations of the formulas. The *gen*-relation also holds for the existentially quantified variable but only because the formula is not in PLNF (see below for an explanation). We start with an empty input instance $db = \emptyset$. Figure 11 illustrates the infinite search tree that the *pre*CQE algorithm would construct. Note that each ground atom contained in the illustration is meant to be added to the database instance and *Employee* is abbreviated with *Emp* there. In fact, a new constant has to be invented as the boss of each employee because nobody can be the boss of their own according to the third constraint. Thus the hierarchy is infinite and invention never stops. ♢

We note that the allowed property of formulas of this fragment does not get along well with the PLNF representation: observe in the example that moving the existential quantifier to the prenex destroys the allowed property of the first constraint formula. This mainly lies in the fact that pushing ∃ quantifiers to the front of the formula does not preserve the *gen*-relation for the existentially quantified variables

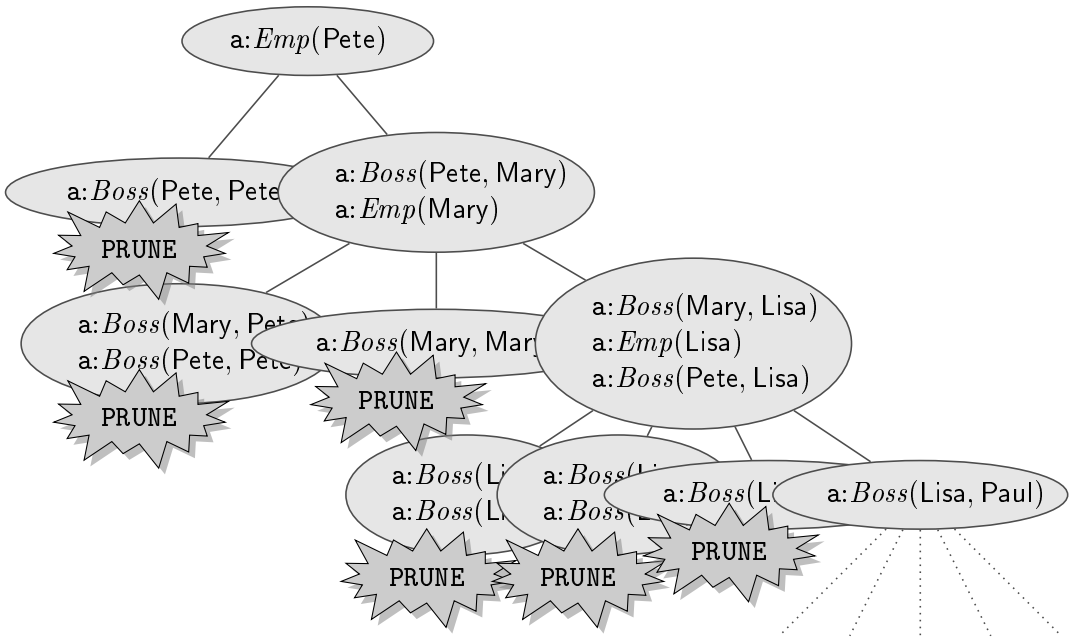


Figure 11: Example for infinity axiom

(see also Example 6.1 of [GT91]). The PLNF representation does however comply with the "evaluable" property (see also [GT91]). Evaluable formulas form a more general class of safe queries which can be defined by a inductive definition of a "con(strained)"-relation akin to the *gen*-relation for allowed formulas. But even with evaluable formulas the constraint set can only be satisfied in an infinite model and thus is not DB-satisfiable. This is why in this section we do not only require the allowed or evaluable property for constraint formulas, but have to follow a different path to restrict this fragment. When viewing the constraints as database dependencies we can apply results from dependency theory to our problem. More precisely, we only allow the combination of $\forall$ and $\exists$ quantifiers in "tuple-generating dependencies" and construct a "dependency graph" of $C$. These concepts are also used by Fagin et al. (see [FKMP05]) in the context of "data exchange"; in particular, the notion of "weak acyclicity" of the dependency graph originates from [FKMP05].

---

**Definition 12.2 (Tuple-generating dependencies)**
*A* tuple-generating dependency (TGD) *is a closed formula of the form*

$$\forall \vec{x}\,(\Phi(\vec{x}) \rightarrow \exists \vec{y} \Psi(\vec{x}, \vec{y}))$$

*where* $\Phi(\vec{x})$ *and* $\Psi(\vec{x}, \vec{y})$ *are conjunctions of atomic formulas.* $\Phi(\vec{x})$ *is called the* body *and* $\Psi(\vec{x}, \vec{y})$ *is called the* head *of the TGD.*
*A TGD is called* full *if there are no existentially quantified variables* $\vec{y}$ *in* $\Psi$.

---

We introduce TGDs here in implicational notation (using material implication $\rightarrow$) for better readability. Yet, the *pre*CQE algorithm will only accept TGDs in which the material implication is replaced by one negation and one disjunction and which is in prenex literal normal form (PLNF; see Section 7). Then, the TGD looks like this:

$$\forall\vec{x}\exists\vec{y}\,(\Phi'(\vec{x}) \vee \Psi(\vec{x},\vec{y}))$$

where $\Phi'(\vec{x}) = \neg\Gamma_1(\vec{x}) \vee \ldots \vee \neg\Gamma_m(\vec{x})$ is a disjunction of negative literals and $\Psi(\vec{x},\vec{y}) = \Gamma'_1(\vec{x},\vec{y}) \wedge \ldots \wedge \Gamma'_n(\vec{x},\vec{y})$ is a conjunction of positive literals (that is, atomic formulas). A variable $x$ in $\vec{x}$ need not occur in each $\Gamma_i$ nor in each $\Gamma'_j$, and a variable $y$ in $\vec{y}$ need not occur in each $\Gamma'_j$ either. We will call this the "PLNF representation" of a TGD.
In addition to TGDs (and going beyond the approach of [FKMP05]), we will still allow existential formulas – including ground formulas in NNF – in a constraint set $C$. $C$ may also contain allowed universal formulas that are a disjunction of negative literals; that is, formulas of the form $\forall\vec{x}\,\neg\Gamma_1(\vec{x}) \vee \ldots \vee \neg\Gamma_m(\vec{x})$ where each atom $\Gamma_i$ need not contain all of the variables in $\vec{x}$. Such formulas are called "denial constraints" in [CM05]; we will borrow this term for our purposes.
In Example 12.1, the first constraint is a TGD, the second and third constraint are full TGDs, the fourth constraint is denial constraint and the last constraint is ground.

We will use weak acyclicity of the dependency graph for a set of TGDs as a sufficient condition for DB-satisfiability of a constraint set $C$ containing not only these TGDs but also existential and denial constraints. While the basic notions are borrowed from [FKMP05], our approach differs in many points from theirs. We will exhibit the differences in more detail in Section 15.

---

**Definition 12.3 (Dependency graph / weak acyclicity ([FKMP05]))**
*For a set $S$ of tuple-generating dependencies, its* dependency graph *is determined as follows:*

- *for each predicate symbol $P$ occurring in $S$, create $arity(P)$ many nodes $P_1, \ldots, P_{arity(P)}$ – the* positions *of $P$*
- *for every TGD $\forall\vec{x}\,(\Phi(\vec{x}) \rightarrow \exists\vec{y}\Psi(\vec{x},\vec{y}))$ in $S$: if a universally quantified variable $x \in \vec{x}$ occurs in a position $P_i$ in $\Phi$ and in a position $P'_j$ in $\Psi$, add an edge from $P_i$ to $P'_j$ (if it does not already exist).*
- *for every TGD $\forall\vec{x}\,(\Phi(\vec{x}) \rightarrow \exists\vec{y}\Psi(\vec{x},\vec{y}))$ in $S$: if a universally quantified variable $x \in \vec{x}$ occurs in a position $P_i$ in $\Phi$ and in a position $P'_{j_1}$ in $\Psi$, and an existentially quantified variable $y \in \vec{y}$ occurs in a position $P''_{j_2}$ in $\Psi$, add a* special edge *marked with $\exists$ from $P_i$ to $P''_{j_2}$ (if it does not already exist).*

*A dependency graph is* weakly acyclic, *iff it does not contain a cycle going through a special edge.*

---

We call a set of TGDs weakly acyclic whenever its dependency graph is weakly acyclic.

*Example 12.4:* The dependency graph for Example 12.1 has one normal edge from $Emp_1$ to $Boss_1$ and one special edge from $Emp_1$ to $Boss_2$ for the first constraint. It has two normal edges from $Boss_1$ and $Boss_2$ to $Emp_1$ for the second constraint. The third constraint causes a normal edge each from $Boss_1$ and $Boss_2$ to themselves. We see in Figure 12 that the dependency graph has a cycle involving a special edge. We could delete the first formula containing the existential quantifier in order to make the graph acyclic; in this case, for example the solution instance $db'_1 = \{Emp(\mathsf{Pete})\}$ satisfies the remaining constraints. Yet we could also remove the second formula to break the cycle in the dependency graph; in this case, a solution instance would be $db'_2 = \{Emp(\mathsf{Pete}), Boss(\mathsf{Pete}, \mathsf{Mary})\}$. ♢

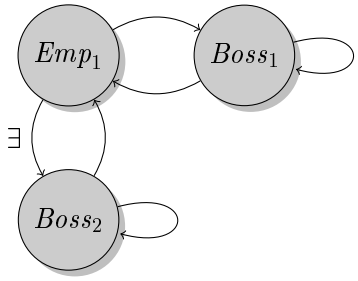


Figure 12: Dependency graph for infinity axiom

As an aside, we remark that the removal of cycles containing special edges in the dependency graph need not be the only way to resolve an infinity axiom: in Example 12.1, we could also remove the third formula (that is, abandon the transitivity property of predicate $Boss$); in this case, a solution instance would be $db' = \{Emp(\mathsf{Pete}), Emp(\mathsf{Mary}), Boss(\mathsf{Pete}, \mathsf{Mary}), Boss(\mathsf{Mary}, \mathsf{Pete})\}$. Or we could remove the fourth formula and get a solution like $db' = \{Emp(\mathsf{Pete}), Boss(\mathsf{Pete}, \mathsf{Pete})\}$.

The salient point of this chapter is that weakly acyclic TGDs can be satisfied with finite invention. There are two crucial differences to finite invention for the existential fragment: First of all, universal quantifiers in the constraints bind variables to constants of $adom(db)$ such that we have to consider constants from the active domain $adom$ of $C$ as well as $db$ (in contrast to just considering the non-isomorphic $adom(C)$ constants in the existential case). Second, TGDs require much more invented constants because for *any* tuple that satisfies the body of the TGD, invention for the existentially quantified variables has to take place. The next lemma analyzes how the invention can be bounded. Its proof is quite analogous to the one of Theorem 3.8 in [FKMP05] with the crucial difference that we only count the amount of invention that is necessary due to existential quantification (instead of counting the total amount of constants that can occur in all positions of the dependency graph); this has the effect that we only take into account those positions in the dependency

graph that have incoming special edges (as opposed to considering all positions in [FKMP05]) because invention only takes place due to existential quantifiers. We furthermore explicitly allow constants to occur in the TGDs.

**Lemma 12.5 (Finite invention for TGDs)**
*Given a constraint set $C$ of weakly acyclic TGDs, and assuming prior $\not\models_{DB}$ pot_sec_disj, there exists an inference-proof solution instance that contains only constants from the set $adom \cup invent_{tgd}$, where $invent_{tgd} \subset (dom \setminus adom)$, and there is a polynomial in $card(adom)$ that bounds the cardinality of $invent_{tgd}$.*

*Proof.* The proof is based on the structure of the dependency graph; it analyzes the number of special edges on paths in the graph. Our first observation is that paths with special edges in the graph are finite: The graph is assumed to be weakly acyclic such that there are no cycles with special edges. Hence, the number of special edges on a path is finite and we can determine the maximum of special edges over all paths; we denote this maximum $\max^{\exists}$.
Next, we group together those positions that have incoming special edges and where no path ending in such an edge has more than $i$ special edges ($i$ ranges over $\{1 \dots \max^{\exists}\}$ and $P_{i'}$ stands for arbitrary positions in the graph):

$$Pos_i := \{ \quad P_{i'} \mid \begin{array}{l} \text{there is a path with } i \text{ special edges ending with a special edge} \\ \text{in } P_{i'} \text{ but there is no such path with more than } i \text{ special edges } \} \end{array}$$

$i$ is called the "rank" of the positions in $Pos_i$. We will also need the maximum number of special edges entering a position in $Pos_i$ (the maximum "fanin" of special edges); we denote this number $\max_i^{fanin}$. An upper bound for the amount $p_i(card(adom))$ of invented constants can be inductively defined over the ranks $i = 1 \dots \max^{\exists}$ as follows. The base case is rank 1: For any position in $Pos_1$ and any incoming special edge, constants in the position from where the special edge originates can only range over $adom$ (otherwise the rank would be greater than 1). Depending on the fanin, invention occurs for tuples of $adom$ constants (due to variables occurring in both the body and the head of a TGD); that is, for each of the $card(adom)^{\max_1^{fanin}}$ tuples at most one new constant has to be invented. As the same position may occur in more than one TGD, we lastly factor in $card(C)$:

$$p_1(card(adom)) := card(Pos_1) \cdot card(adom)^{\max_1^{fanin}} \cdot card(C)$$

We now see that for all other ranks essentially the same arguments apply with the exception that we have to sum up the constants that were invented in nodes with lesser ranks; these invented constants can indeed cause invention in a higher ranked position:

$$p_i(card(adom)) := card(Pos_i) \cdot (card(adom) + \sum_{j=1}^{i-1} p_j(card(adom)))^{\max_i^{fanin}} \cdot card(C)$$

Consequently, to count the amount of invention necessary to satisfy the set $C$ of TGDs, we sum up all the $p_i$ values; hence, the cardinality of $invent_{tgd}$ can be bounded by $\sum_{i=1}^{\max^{\exists}} p_i(card(adom))$. This is in fact a polynomial in $card(adom)$ if $C$ is assumed to be fixed. □

We now go one step further and allow not only TGDs but also existential and denial constraints in $C$. We can establish DB-satisfiability of such a more general constraint set as a corollary of Theorem 7.11 and Lemma 12.5.

---

**Corollary 12.6 (Inference-proof instances for weakly acyclic constraints)** *Given a constraint set $C$ of weakly acyclic TGDs, existential formulas and denial constraints and assuming prior $\not\models_{DB}$ pot_sec_disj, there exists an inference-proof solution instance that contains only constants from the set $adom \cup invent_{\exists} \cup invent_{tgd}$, where*

- $invent_{\exists} \subset (dom \setminus adom)$
- $invent_{tgd} \subset (dom \setminus (adom \cup invent_{\exists}))$
- $card(invent_{\exists}) = l$ *where $l$ is the number of variables occurring in existential constraints*
- *there is a polynomial in $card(adom \cup invent_{\exists})$ that bounds the cardinality of $invent_{tgd}$.*

---

*Proof.* Existential constraints can be satisfied with finite invention as exhibited in Corollary 10.4; this situation does not change in the presence of denial constraints and TGDs: Still at most one constant per variable has to be invented in order to satisfy an existential constraint. That is, we take $invent_{\exists}$ as the invention necessary for the satisfaction of all existential constraints. Denial constraints can only be satisfied by removal of database entries and hence never increase the active domain. Turning our attention to TGDs, we observe that invented constants from $invent_{\exists}$ can potentially be copied to other positions by TGDs. That is, invention for TGDs not only occurs for values of $adom$, but also for values from $invent_{\exists}$. To estimate the amount of constants necessary to satisfy all TGDs, we compute the same polynomials $p_i$ as in Lemma 12.5 but replace $card(adom)$ with $card(adom \cup invent_{\exists})$. □

When looking for an upper bound of the distortion distance, we have to keep in mind the fact that constants can be copied from one position to another by TGDs. This implies that invented constants potentially do not only appear in those predicate symbols (or positions) that are in the scope of an existential quantifier but also in other predicate symbols (or positions) occurring in $C$. That is, in any predicate symbol in $C$, constants from $adom \cup invent_{\exists} \cup invent_{tgd}$ can occur in any position. We estimate that any such tuple changes its truth value in $db'$.

**Corollary 12.7 (Upper bound of distortion distance)**
*For a constraint set $C$ of weakly acyclic TGDs, existential formulas and denial constraints, the distortion distance for any inference-proof and distortion-minimal database instance $db'$ can be bounded as follows*

$$db\_dist(db') \leq \sum_{\substack{P \in \mathcal{P} \\ P \text{ occurs in } C}} card(adom \cup invent_{\exists} \cup invent_{tgd})^{arity(P)}$$

## 13 *pre*CQE for Weakly Acyclic Constraints

In the upcoming algorithm, we combine the previous approaches for universal and existential constraints and treat the quantified variables from left to right (as given by the order of their quantifiers) in a kind of "lazy" partial instantiation: For universally quantified variables again violation sets are computed and according to these the ground instantiations of atoms are determined; for existentially quantified variables again instantiations with active domain constants plus one new invented value are tried. Whenever a ground literal is reached (by a partial instantiation of variables), a splitting first tries to satisfy the constraint by marking the literal appropriately; afterward, instantiation continues to find another ground literal. For this strategy we rely on the fact that all constraint formulas are in PLNF – also the TGDs are assumed to be in PLNF.

Before listing the algorithm, we provide a small example that gives some intuition about the approach taken to handle TGDs. Its *pre*CQE search tree can be found in Figure 13.

*Example 13.1:* We have a set of constraints saying that no patient receives medical treatment without having been diagnosed a disease, no patient is ill with Aids, and there are already two known patients with a treatment

$$\begin{aligned} C \quad = \{ \quad & \forall xz\,(Treat(x,z) \rightarrow \exists y Ill(x,y)), \\ & \forall x\,\neg Ill(x, \mathsf{Aids}), \\ & Treat(\mathsf{Mary}, \mathsf{MedA}), \\ & Treat(\mathsf{Lisa}, \mathsf{MedB}) \qquad\qquad \} \end{aligned}$$

and the input instance

$$db = \{Treat(\mathsf{Mary}, \mathsf{MedA}), Treat(\mathsf{Lisa}, \mathsf{MedB})\}$$

We observe that the dependency graph for the TGD in $C$ is acyclic (and thus also weakly acyclic) and the precondition for Corollary 12.6 holds.

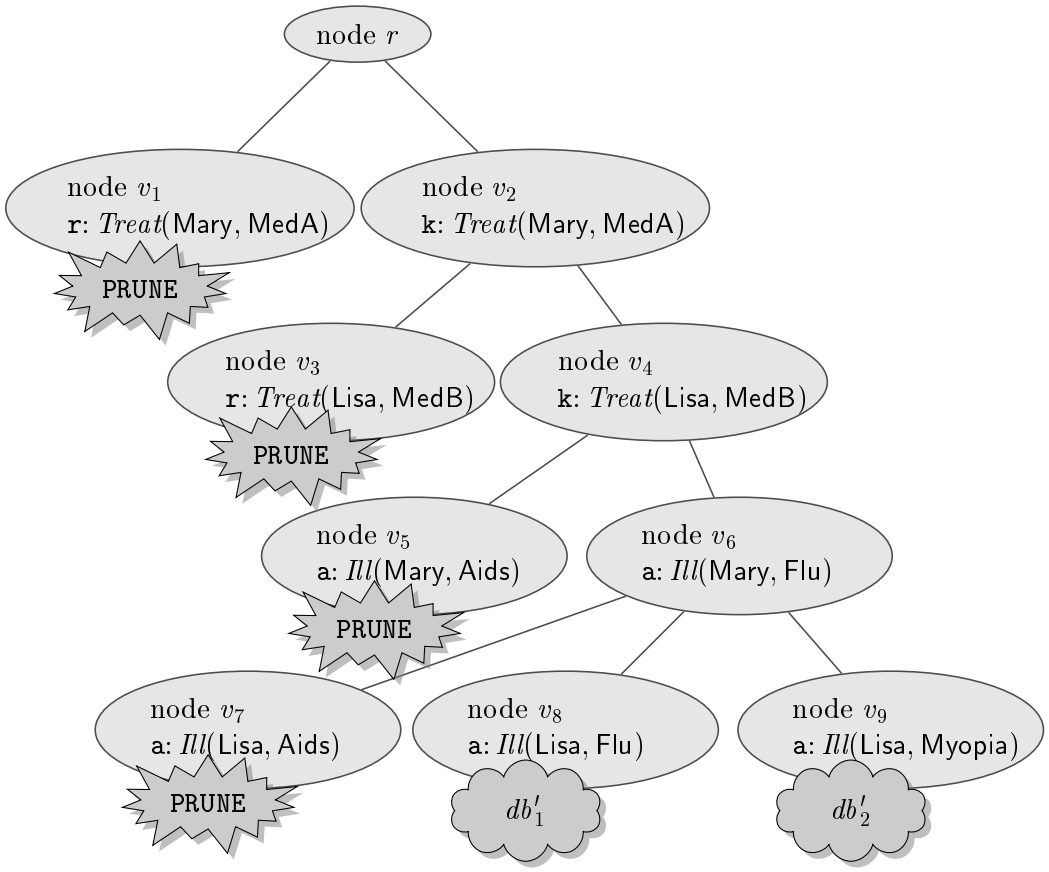


Figure 13: Example for weakly acyclic constraints

We then start with the construction of the *pre*CQE search tree where $db_r$ is identical to $db$. We find only the first constraint violated in $db_r$. The first quantifier is $\forall$; we handle it – as in the universal case – by evaluating the PLNF representation of its negation (without the universal quantifiers) on $db_r$. This gives us

$$V_r = \{ \quad \forall y\, Treat(\mathsf{Mary}, \mathsf{MedA}) \wedge \neg Ill(\mathsf{Mary}, y), \quad \forall y\, Treat(\mathsf{Lisa}, \mathsf{MedB}) \wedge \neg Ill(\mathsf{Lisa}, y) \quad \}$$

as the violation set. Because $Treat(\mathsf{Mary}, \mathsf{MedA})$ is ground, we split on it. Removing it however immediately violates the third constraint: $Treat(\mathsf{Mary}, \mathsf{MedA})$ is marked with r and hence not satisfied in $db_{v_1}$. The same argument applies to $Treat(\mathsf{Lisa}, \mathsf{MedB})$ in node $v_3$. In node $v_4$, still the first constraint is violated with the same violation set as before but all ground atoms are marked. Now case differentiation for the existentially quantified variable takes place: for the patient Mary, $Ill(\mathsf{Mary}, \mathsf{Aids})$ is tried in $v_5$ because Aids is contained in $adom_{v_4} \cap Disease$; but then immediately a conflict is detected in the violation set (because of the second constraint – the denial constraint). In $v_6$ the disease Flu is invented. Case differ-

entiation for Lisa follows: $adom_{v_6} \cap Disease$ now contains both Aids and Flu and Myopia is invented. We see that both $db_{v_8}$ and $db_{v_9}$ result in solution instances with distortion distance 2 each. ♢

## 13.1 The *pre*CQE Algorithm

`INIT` **(Listing 12)** remains unaltered from the previous `INIT` procedure: the root node is created and all local values are initialized; then, `GROUND` is executed.

`GROUND` **(Listing 13)** is more complicated than in the previous sections. It still determines the set of violated constraints $C_v^{vio}$ (Line 13.1.) and sets the new optimum whenever there are no more violated constraints (Line 13.2.).
Instantiations of violated constraints with universal quantifiers are computed first: the violation set $V_v$ (see Line 13.3.3.) is obtained by determining the positive evaluation $eval_v^{pos}$ of the PLNF representations of the negations of violated constraints without the existential prenex. Note that for general TGDs with existential quantifiers, the evaluation is a set of formulas with universally quantified variables; only for full TGDs and denial constraints, the evaluation returns ground formulas. To be able to distinguish the different non-ground violation formulas, variables are standardized apart in Line 13.3.4. (this is exactly what the function *standardizevariables* is supposed to do).
For each formula $\Psi$ in the violation set, the PLNF of its negation ($plnf(\neg\Psi)$) is formed and put into the "instantiation set" $C_v^{inst}$ (Line 13.3.5.). Only those formulas are added to $C_v^{inst}$ of which no instance is contained in $C_v$; in other words, there is no formula in $C_v$, that

- has the same structure of the matrix
- has less or equally many existential quantifiers in its prenex
- has identical constants in the non-quantified positions in the matrix

More formally, there must not exist a substitution

$$\sigma' : vars(plnf(\neg\Psi)) \to dom \cup vars(C_v)$$

such that $(plnf(\neg\Psi))[\sigma'] \in C_v$ (that is, applying the substitution to all variables of $plnf(\neg\Psi)$ and ignoring unnecessary existential quantifiers). For example, the

```
12. INIT: Initialization for root node r
    12.1. create root node r
    12.2. db_r:=db;
    12.3. min_lies_r := 0;
    12.4. C_r := C;
    12.5. db_best := undefined;
    12.6. min_lies_best := ∞;
    12.7. GROUND(r);
```

Listing 12: *pre*CQE – Initialization for weakly acyclic constraints

13. `GROUND(`$v$`)`: Check **ground formulas** in node $v$
  13.1. $C_v^{vio} := \{\Phi \in C_v \mid eval_v(\Phi) = \neg\Phi\}$;
  13.2. `if` $(C_v^{vio} = \emptyset)$
    13.2.1. $db_{best} := db_v$;
    13.2.2. $min\_lies_{best} := min\_lies_v$;
  13.3. `else`
    13.3.1. $C_v^{\forall} := \{\Phi \in C_v^{vio} \mid$ outermost quantifier is $\forall\}$;
    13.3.2. `foreach` $\Phi_i \in C_v^{\forall}$
      13.3.2.1. $\Phi_i' := plnf(\neg\Phi_i)$;
      13.3.2.2. $\Phi_i'' := dropexistentialprenex(\Phi_i')$;
    13.3.3. $V_v := \bigcup_i eval_v^{pos}(\Phi_i'')$;
    13.3.4. $standardizevariables(V_v)$;
    13.3.5. $C_v^{inst} := \{plnf(\neg\Psi) \mid \Psi \in V_v$, no instance of $plnf(\neg\Psi)$ in $C_v\}$;
    13.3.6. $C_v^{\exists} := C_v^{inst} \cup C_v^{vio} \setminus C_v^{\forall}$;
    13.3.7. $C_v^{simp} := C_v^{\exists}$;
    13.3.8. `SIMP(`$v$`)`;
    13.3.9. `if (` there is ground $\phi \in C_v^{simp}$ with $card(unmarked_v(\phi)) = 0$ `)`
      13.3.9.1. `PRUNE; //`(conflicting markers)
    13.3.10. `else if (` there is ground $\phi \in C_v^{simp}$ with $card(unmarked_v(\phi)) = 1$ `)`
      13.3.10.1. take unique literal $\lambda \in unmarked_v(\phi)$;
      13.3.10.2. `MARK(`$v$`,`$\lambda$`)`;
      13.3.10.3. `GROUND(`$v$`)`;
    13.3.11. `else if (` there is $\Phi \in C_v^{simp}$ with $card(unmarked_v(\Phi)) > 0$ `)`
      13.3.11.1. `SPLIT(`$v$`)`;
    13.3.12. `else`
      13.3.12.1. `CASE(`$v$`)`;

Listing 13: *pre*CQE – Ground violations for weakly acyclic constraints

14. `SIMP(`$v$`)`: **Simplification** of violated constraints
  14.1. `repeat` until no more changes occur:
    14.1.1. `foreach` subformula $\Phi'$ of a formula $\Phi \in C_v^{simp}$
      14.1.1.1. `if` $(\Phi' = \Psi \wedge \gamma$ and $marker_v(\gamma) \in \{\mathtt{k}, \mathtt{a}\}$
          `or` $\Phi' = \Psi \wedge \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{r}, \mathtt{l}\}$
          `or` $\Phi' = \Psi \vee \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{k}, \mathtt{a}\}$
          `or` $\Phi' = \Psi \vee \gamma$ and $marker_v(\gamma) \in \{\mathtt{r}, \mathtt{l}\})$
        14.1.1.1.1. replace $\Phi'$ with $\Psi$; `//`(unit resolution)
      14.1.1.2. `if` $(\Phi' = \Psi \wedge \gamma$ and $marker_v(\gamma) \in \{\mathtt{r}, \mathtt{l}\}$
          `or` $\Phi' = \Psi \vee \gamma$ and $marker_v(\gamma) \in \{\mathtt{k}, \mathtt{a}\})$
        14.1.1.2.1. replace $\Phi'$ with $\gamma$; `//`(unit subsumption)
      14.1.1.3. `if` $(\Phi' = \Psi \wedge \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{k}, \mathtt{a}\}$
          `or` $\Phi' = \Psi \vee \neg\gamma$ and $marker_v(\gamma) \in \{\mathtt{r}, \mathtt{l}\})$
        14.1.1.3.1. replace $\Phi'$ with $\neg\gamma$; `//`(unit subsumption)
      14.1.1.4. if a variable disappeared from $\Phi$ remove its quantifier from the prenex

Listing 14: *pre*CQE – Simplification for weakly acyclic constraints

15. SPLIT($v$): **Splitting** on a ground atom in node $v$
    15.1. choose $\Phi \in C_v^{simp}$ with $card(unmarked_v(\Phi)) > 0$;
    15.2. choose $\lambda \in unmarked_v(\Phi)$;
    15.3. generate two child nodes $v_{left}$ and $v_{right}$;
    15.4. $db_{v_{left}} := db_{v_{right}} := db_v$;
    15.5. $C_{v_{left}} := C_{v_{right}} := C_v$;
    15.6. $min\_lies_{v_{left}} := min\_lies_{v_{right}} := min\_lies_v$;
    15.7. MARK($v_{left}$,$\neg|\lambda|$);
    15.8. GROUND($v_{left}$);
    15.9. MARK($v_{right}$,$|\lambda|$);
    15.10. GROUND($v_{right}$);

Listing 15: *pre*CQE – Splitting on a ground atom for weakly acyclic constraints

16. CASE($v$): **Case differentiation** in node $v$
    16.1. choose $\exists x \Phi(x) \in C_v^{\exists}$ where $x$ also occurs in $C_v^{simp}$;
    16.2. foreach $a \in adom_v$
        16.2.1. generate child node $v_a$;
        16.2.2. $db_{v_a} := db_v$;
        16.2.3. $C_{v_a} := C_v \cup \{\Phi(a)\} \setminus \{\exists x \Phi(x)\}$;
        16.2.4. $min\_lies_{v_a} := min\_lies_v$;
        16.2.5. GROUND($v_a$);
    16.3. choose $a' \in invent \setminus adom_v$;
        16.3.1. generate child node $v_{a'}$;
        16.3.2. $db_{v_{a'}} := db_v$;
        16.3.3. $C_{v_{a'}} := C_v \cup \{\Phi(a')\} \setminus \{\exists x \Phi(x)\}$;
        16.3.4. $min\_lies_{v_{a'}} := min\_lies_v$;
        16.3.5. GROUND($v_{a'}$);

Listing 16: *pre*CQE – Case differentiation for weakly acyclic constraints

17. MARK($v$,$\lambda$): **Marking** an unmarked ground atom $\gamma$ in $db_v$
    17.1. $\gamma := |\lambda|$;
    17.2. if ($\lambda = \gamma$ and $eval_v(\gamma) = \gamma$ )
        17.2.1. $marker_v(\gamma)$ :=k;
    17.3. else if ($\lambda = \gamma$ and $eval_v(\gamma) = \neg\gamma$ )
        17.3.1. $marker_v(\gamma)$ :=a;
        17.3.2. $min\_lies_v$++;
    17.4. else if ($\lambda = \neg\gamma$ and $eval_v(\gamma) = \gamma$ )
        17.4.1. $marker_v(\gamma)$ :=r;
        17.4.2. $min\_lies_v$++;
    17.5. else if ($\lambda = \neg\gamma$ and $eval_v(\gamma) = \neg\gamma$ )
        17.5.1. $marker_v(\gamma)$ :=l;
    17.6. if ($min\_lies_v \geq min\_lies_{best}$) PRUNE; //(bad bound)

Listing 17: *pre*CQE – Marking of ground atoms for weakly acyclic constraints

formula $Ill(\mathsf{Mary}, \mathsf{Aids})$ is an instance of $\exists y\, Ill(\mathsf{Mary}, y)$ and hence the second formula would not be added to $C_v^{inst}$. And again $\exists y\, Ill(\mathsf{Mary}, y)$ is an instance of both $\exists x' y'\, Ill(x', y')$ and $\exists y''\, Ill(\mathsf{Mary}, y'')$ and thus both formulas would not be added to $C_v^{inst}$. Note that the second type of instance is sometimes called a "variant" of a formula; to avoid an abundant terminology, we will stick with the term "instance" also for such kinds of formulas. Note, too, that due to standardizing no identical variable names will occur in a formula and its instances. All this will be important after instantiating existentially quantified variables in the `CASE` procedure.
$C_v^{inst}$ is generated because these formulas are later on added to the set $C_v^{\exists}$ of existential formulas (Line 13.3.6.). $C_v^{\exists}$ contains on top of that all existential formulas in $C_v^{vio}$; but it does not contain the formulas with universal quantifiers because with the help of the set $C_v^{inst}$, violating ground instantiations for the universally quantified variables have already been determined. Then, $C_v^{simp}$ is the set that is simplified (see Lines 13.3.7. and 13.3.8.) and later on used to avoid unnecessary instantiations and markers. Keeping $C_v^{\exists}$ unsimplified ensures that in `CASE` the correct formulas can be deleted from $C_v$.
`GROUND` proceeds as in the previous sections by checking whether there is a conflict in a ground formula (Lines 13.3.9. and 13.3.9.1.). When there is a ground formula with one unmarked literal, a marker satisfies this literal (Lines 13.3.10.1. and 13.3.10.2.). As long as there are ground formulas with more than one unmarked ground literal, or non-ground formulas with at least one unmarked literal, a `SPLIT` is executed (Line 13.3.11.1.). If none of the above cases holds, we can be sure that there are only violated constraints with existential prenex in $C_v^{vio}$; then, our last option is to execute a `CASE` (Line 13.3.12.1.) that removes an existential quantifier and creates several child nodes for it (see below).

`SIMP` **(Listing 14)** is the same as for existential formulas.

`SPLIT` **(Listing 15)** chooses one simplified violated formula with at least one unmarked literal (see Line 15.1.) and then chooses one of the unmarked literals of that formula (Line 15.2.). Two child nodes (Line 15.3.) are created to try both truth values for the literal (Lines 15.7. and 15.9.). In both cases, a recursion to `GROUND` takes place.

`CASE` **(Listing 16)** proceeds as in the *pre*CQE version for existential formulas: It instantiates one existentially quantified variable $x$ of a formula in the set of violated existential constraints $C_v^{\exists}$ (Line 16.1.) for which the variable $x$ also occurs in the simplified version of the formula in $C_v^{simp}$. Again, $x$ need not be the first existentially quantified variable in the prenex; that is there may occur other variables in front of $x$ in the prenex that however do not occur in $C_v^{simp}$.
There are only minor differences to the existential case. First, in Line 16.3. we assume that there is a set $invent \subset (dom \setminus adom)$ such that $invent = invent_{tgd} \cup invent_{\exists}$ according to Corollary 12.6. Second, the removal of the existential formula

(in Lines 16.2.3. and 16.3.3.) only succeeds if there was such an existential formula in $C_v$ before; if instead the existential formula was added in $C_v^{\exists}$ due to the violation of some TGD, the instantiated formula is just added to $C_v$ without removing any formula from it. But then, in an interplay of `GROUND` and `CASE`, only instances of the newly added constraint will later on appear in $C_v^{inst}$ (for $v$ and all nodes below $v$). In each of the $card(adom_v) + 1$ child nodes, `GROUND` is called (Lines 16.2.5. and 16.3.5.).

`MARK` **(Listing 17)** is unchanged from the previous *pre*CQE versions: ground atoms of ground literals are marked depending on their evaluation in the current database instance.

## 13.2 Termination, Soundness and Completeness of *pre*CQE

As with the previous versions of the *pre*CQE algorithm we start by showing that the algorithm terminates for weakly acyclic TGDs, existential formulas and denial constraints. The termination proof will be supported by the following lemma that establishes finiteness of the violation sets $V_v$.

**Lemma 13.2 (Finite violation sets with *adom* ∪ *invent* constants)**
*For a set $C$ of weakly acyclic TGDs, existential formulas and denial constraints, the set $V_v$ of ground violations (according to Line 13.3.3.) is always finite for any node $v$ and contains only ground or existential formulas with constants from $adom \cup invent$ (where $invent = invent_{tgd} \cup invent_{\exists}$ from Corollary 12.6).*

*Proof.* The formulas for which violation sets are computed are those with universal quantifiers in their prenex. Denial constraints as well as full TGDs are allowed formulas; in these cases, the same arguments as in Lemma 9.8 apply with the only difference that *adom* is augmented with invention for existentially quantified variables.
We are left with showing that for TGDs with existentially quantified variables the positive evaluation of their negations in $db_v$ is finite. More precisely, for a TGD in PLNF representation $\Phi = \forall\vec{x}\exists\vec{y}\,(\Phi'(\vec{x}) \vee \Psi(\vec{x},\vec{y}))$ (where $\Phi'(\vec{x}) = \neg\Gamma_1(\vec{x}) \vee \ldots \vee \neg\Gamma_m(\vec{x})$ is a disjunction of negative literals and $\Psi(\vec{x},\vec{y}) = \Gamma'_1(\vec{x},\vec{y}) \wedge \ldots \wedge \Gamma'_n(\vec{x},\vec{y})$ is a conjunction of positive literals), we see that

$$plnf(\neg\Phi) = \exists\vec{x}\forall\vec{y}\,(\Gamma_1(\vec{x}) \wedge \ldots \wedge \Gamma_m(\vec{x}) \wedge (\neg\Gamma'_1(\vec{x},\vec{y}) \vee \ldots \vee \neg\Gamma'_n(\vec{x},\vec{y})))$$

and $gen(x, plnf(\neg\Phi))$ (for every $x \in \vec{x}$) holds because $x$ occurs in at least one $\Gamma_i$ (for $i = 1 \ldots m$); note that the $x \in \vec{x}$ are free after an application of *dropexistentialprenex*. As for the $y \in \vec{y}$, we verify that the relation $con(y, \neg(plnf(\neg\Phi)))$ holds because $y$ is absent from all $\Gamma_i$ but contained in at least one $\Gamma'_j$ (for $j = 1 \ldots n$). We do not fill in the details here but refer to [GT91] for the complete definition of the *con*-relation; there, the authors also state that the evaluable formulas form the largest class of domain-independent formulas and thus for database instances, finite query

responses are ensured. What is more, $plnf(\neg\Phi)$ could easily be transformed into an equivalent allowed formula before evaluating it on $db_v$ by pushing the universal quantifier inward. Hence, in any of the ways, the evaluation returns a finite set of formulas that contain only constants from $adom \cup invent$. □

Profiting from the experiences with universal and existential formulas, we can now transfer the termination result to the more general constraint set.

**Theorem 13.3 (Termination of *pre*CQE)**
*For a set $C$ of weakly acyclic TGDs, existential formulas and denial constraints, preCQE terminates in a finite amount of time.*

*Proof.* As in the previous cases, we can argue with the help of Lemma 13.2 that all computed sets of formulas are finite and again simplification with `SIMP` also takes finite time. As before in the existential case, the number of generated child nodes in a `CASE` step is bounded by $card(adom) + card(invent)$ except for the fact that the present set of invented constants is larger.
We now concentrate on showing that recursion stops and thus the *pre*CQE search tree has no infinite branches. First we argue that there can only be finitely many calls to `CASE` along a branch in the search tree. Most notably, it may happen that new existential formulas are added to but none are removed from a constraint set $C_v$ (see Lines 16.2.3. and 16.3.3.): whenever `CASE` is called for the first time for an instance of a TGD in $C_v^{inst}$, the formula $\exists x\Phi(x)$ is only contained in $C_v^{\exists}$ but not in $C_v$ and hence the instantiations $\Phi(a)$ or $\Phi(a')$ are added to $C_v$ without removing $\exists x\Phi(x)$. This addition only happens if all the universally quantified variables of a TGD have been bound to constants in $adom_v$ (with the help of $C_v^{inst}$ in Line 13.3.5.) and all its ground atoms are marked. Moreover, the instance test in Line 13.3.5. ensures that as soon as such a formula has been added to $C_v$ no more general formula for the same tuple of $adom_v$ constants but with more existentially quantified variables will ever be added again. That is, for every instantiation of the universally quantified variables in the head of a TGD with $adom \cup invent$ constants, at most once one existential formula is added to $C_v$: the body of the TGD will be simplified because all its atoms are marked. At an outside estimate, if $d$ is the number of TGDs in $C$ that have existentially quantified variables in their head and $e$ is the maximum amount of universally quantified variables occurring in these TGDs, at most $d \cdot card(adom \cup invent)^e$ existential formulas are added to $C_v$. Let now $f$ be the maximum amount of existentially quantified variables occurring in the heads of TGDs, then at most

$$l' := d \cdot card(adom \cup invent)^e \cdot (f-1)$$

new existentially quantified variables occur in $C_v$ due to Line 13.3.5. and standardization in Line 13.3.4. (we use $f-1$ because one variable is immediately instantiated before adding the formula to $C_v$).

In all subsequent instantiations of existentially quantified variables, indeed the instantiation replaces the original formula in $C_v$. We can in fact be sure that the amount of existentially quantified variables decreases with such a replacement. Hence now we can argue that there cannot be more than $l + l'$ recursive calls to `CASE` (where $l$ is again the number of variables occurring in existential formulas in $C$).
Lastly, we argue (as in the previous versions) that the number of `MARK` calls on ground atoms is bounded. More precisely, there are

$$k'' := \sum_{\substack{P \in \mathcal{P} \\ P \text{ occurs in } C}} card(adom \cup invent)^{arity(P)}$$

different ground atoms and again each recursive `GROUND` call (either in Line 13.3.10.2. or in Lines 15.7. and 15.9.) is preceded by a `MARK` operation – except for the `GROUND` calls in `CASE` which we already have accounted for. That is, after at most $l + l' + k''$ operations, either pruning occurs or a new optimum is found. Hence the length of a branch in the *pre*CQE search tree is bounded by $l + l' + k''$. □

We next see that also in this final version of the *pre*CQE algorithm, $db_{best}$ is only set if no violated constraint is left; that is why satisfiability soundness is still ensured.

**Theorem 13.4 (Satisfiability soundness of *pre*CQE)**
*For a set $C$ of weakly acyclic TGDs, existential formulas and denial constraints, and after running preCQE to completion, if $db_{best}$ is a marked database instance, then its positive restriction is an inference-proof database instance.*

*Proof.* See Theorem 9.10. □

Again, refutation completeness follows from satisfiability soundness.

**Corollary 13.5 (Refutation completeness of *pre*CQE)**
*For a set $C$ of weakly acyclic TGDs, existential formulas and denial constraints, if $C$ is unsatisfiable, then – after running preCQE to completion – $db_{best}$ is undefined.*

Refutation soundness (and satisfiability soundness as its corollary) can be established by combining the proofs of Theorems 9.17 and 11.6.

**Theorem 13.6 (Refutation soundness of *pre*CQE)**
*For a set $C$ of weakly acyclic TGDs, existential formulas and denial constraints, and after running preCQE to completion, if $db_{best}$ is undefined, then $C$ is unsatisfiable.*

*Proof.* We again assume that a mapping from variables to invented constants is fixed and that this mapping is used in the *pre*CQE search tree $T$ whenever invention takes place. Yet, this time we have to expand the mapping to range over all variables that ever occur in $C_v^{\exists}$ – that is, we also include the variables that were newly introduced

by standardization. We denote the mapping $\sigma''$:

$$\sigma'' : \bigcup_{v \text{ in } T} vars(C_v^{\exists}) \to invent$$

where *invent* still is the set $invent_{tgd} \cup invent_{\exists}$ from Corollary 12.6.
The construction of the semantic tree $T^*$ from the *pre*CQE search tree $T$ proceeds as in Theorem 11.6: new levels are introduced for single unmarked literals, only the subtrees below invention nodes are added to $T^*$, and lastly, all invention nodes are removed such that only their binary subtrees are retained in $T^*$.
Next, we identify the set $C_0$ of ground instances of $C$ that will be shown to be refuted by $T^*$. All those formulas ever added to a set $C_v^{\exists}$ with all the existentially quantified variables instantiated according to $\sigma''$ tender themselves as ideal candidates; all universally quantified variables are instantiated according to some violation set $V_v$:

$$C_0 := \bigcup_{v \text{ in } T} \{\ \Phi(\sigma''(x_1), \ldots, \sigma''(x_n)) \mid \exists x_1, \ldots, x_n\, \Phi(x_1, \ldots, x_n) \in C_v^{\exists}\}$$

Showing that every leaf node of $T^*$ is a failure node for $C_0$ follows the same argumentation as in the proof of Theorem 11.6; we repeat the important aspects:

- a leaf of $T^*$ that is also contained in $T$ falsifies the formula of $C_v^{simp}$ that caused the pruning; but then also the unsimplified (and possibly $\sigma''$-instantiated) formula in $C_0$ is falsified.
- a leaf in a new level in $T^*$ that is labeled with the conjugate of a literal marked in $T$ immediately falsifies the formula containing this literal as the single unmarked literal.
- a leaf in a new level in $T^*$ that is labeled with a literal marked in $T$ falsifies a ground formula in $C_0$ that caused the pruning in $T$.

Consequently, $T^*$ is a closed semantic tree for $C_0$ and by Herbrand's Theorem, $C$ is unsatisfiable. □

From refutation soundness follows satisfiability completeness as in the previous cases.

**Corollary 13.7 (Satisfiability completeness of *pre*CQE)**
*For a set $C$ of weakly acyclic TGDs, existential formulas and denial constraints, if $C$ is satisfiable, then – after running preCQE to completion – $db_{best}$ is a marked database instance.*

Likewise, we can resort to Theorem 11.8 to show distortion minimality of the solution instance. The only difference is that now the larger *invent* set for TGDs is used.

**Theorem 13.8 (Optimality of solution)**
*For a set $C$ of weakly acyclic TGDs, existential formulas and denial constraints, if preCQE finds a solution $db_{best}$, then its positive restriction $db_{best}^{pos}$ is distortion-minimal.*

*Proof.* See Theorem 11.8. □

## Summary of Part III

This part started with a consideration of existential formulas. Two significant properties – the finite model property and genericity – were used to derive the fact that only a finite invention of variables is necessary to obtain an inference-proof instance. With respect to distortion minimality however the active domain constants are not isomorphic and hence all instantiations of existentially quantified variables have to be tried to obtain a distortion minimal solution. This in fact is done by the existential version of the *pre*CQE algorithm. The termination result of the *pre*CQE algorithm was based on finite invention and refutation soundness was proved by fixing a one-to-one mapping from variables to invented constants.
The part moved on to $\forall\exists$-quantified constraints. First of all, tuple-generating dependencies (TGDs) were analyzed. The definition and a result for weak acyclicity of the dependency graph were adapted to show applicability of finite invention to TGDs. The constraint set is then extended to additionally contain existential and denial constraints. The algorithm handles universally quantified variables by instantiations and existentially quantified variables by finite invention. The due termination, soundness and completeness proofs were given.

# IV. Extensions and Related Research

## Contents



> Rudy listened to TJ's solution to the two-bisectors problem. TJ had not proved that it was true, but he'd proved that everything else was false, a proof he called reductio ad absurdum. The proof required only a few simple steps, and Rudy was able to follow it without difficulty. The only problem, TJ said, was that some mathematicians didn't accept reductio ad absurdum as a valid principle. But Rudy accepted it, so it didn't matter.
> – *Robert Hellenga, Philosophy Made Simple*

# 14 Adjustments and Extensions

The basic *pre*CQE algorithm can be adjusted to meet special needs that arise in some application areas. In the following, we first discuss techniques that help make the algorithm more efficient in terms of its time-complexity. Afterward we move on to advanced requirements like introducing other distance measures, as well as adding last-minute distortion and explicit availability policies, or considering reliability, incomplete databases and refusal.

## 14.1 Simplification of Constraints

Additional techniques can help optimize the constraint set $C$ such that it can be processed more efficiently. For example, for sets of propositional formulas in CNF, subsumption removal [Zha05], tautology removal, reduction of the number of clauses [EB05] or elimination of variables [SP04] have been proposed in the SAT solving context. As for first-order logic, methods to simplify the input sets have been investigated since long for automated theorem proving (see for example [BH80]). Those techniques can greatly help reduce the size of the constraint set $C$ – both before starting the *pre*CQE algorithm and at runtime after instantiations of existentially quantified variables. Subsumption removal has however been discussed critically in the context of belief revision ([Wil97], see Section 15 for a description of their approach): As it is a costly procedure, the impact on performance might outweigh the advantages of a smaller constraint set.

## 14.2 Iterative Deepening or Best-First Search

If it is impossible to syntactically restrict the input formulas in any of the ways described in this thesis, infinity axioms in the constraint set may not be recognized; in this case, an infinite number of ground atoms are marked in at least one branch of the search tree with the depth-first search approach. Yet, also with syntactically restricted formulas for which we can be sure that they are DB-satisfiable, possibly some of the branches treated early in the depth-first approach execute an unacceptably high amount of marking steps before a better solution is found. To avoid getting stuck in such a branch, we can set a threshold for the number of marking steps in a branch. As soon as the threshold is exceeded, backtracking to the next branch is executed. If no solution within the threshold is found, the threshold can be incremented and conflict-free branches can be reexplored; that is, the branches in the tree are deepened iteratively. Iterative deepening has been proposed in finite model generation where either the domain size (see [CS03]) or the nesting depth of function symbols (see [HRCS02]) is gradually increased. To make iterative deepening for *pre*CQE efficient with respect to time complexity, all partially explored branches must be remembered; hence this is effectively a breadth-first-

search approach which arouses the issue of space requirements. On the other hand the algorithm definitely finds a finite model if one exists – even if some branches of the tree are infinite.
We could also modify the splitting procedure slightly and apply a best-first search look-ahead heuristics: we process that child node first that leaves the lower bound of the distortion distance unchanged.

## 14.3 Non-ground Splitting and Advanced Distance Measures

Resolution is a well-established method to achieve refutation of a set of non-ground input clauses with the help of most general unification. Moreover, splitting on non-ground formulas has been a research topic for the cases of finite satisfiability or model generation (see Section 15 for a description of FDPLL and Darwin) for clause logic. In sharp contrast to our requirements, the concern of these techniques is to either refute the input or find models of minimal size. *pre*CQE requires minimization of a distortion distance with respect to the input database instance *db*. Whether it is possible to incorporate non-ground splitting in the search for such a distance minimal model depends crucially on how the distance is defined. For our cardinality-based – but also for inclusion-based distances – the symmetric difference on ground atoms is calculated. Hence a splitting decision has to be taken for every affected ground atom in the *pre*CQE search tree; moreover local lower bounds for the distortion distance have to be computed based on ground atoms. Non-ground splitting would only improve *pre*CQE if a new distance semantics can be found – without the need to consider ground atoms – or if a reasonable lower bound of the distortion distance can be estimated in a non-ground splitting step.
A different issue regarding the distortion distance is that *pre*CQE could employ an even more fine-grained distance. Instead of calculating distances on sets of ground atoms, attribute-based distances can be used: they measure the amount of distortion in terms of the number of attribute values that are changed (see for example [ADB06, ADB07]). An interesting approach is also the one of [BBFL08] that specifies distances for numerical domains to accommodate aggregate queries on the modified values.

## 14.4 Built-In Predicates, Numerical Domains and Aggregation

Special handling of equality or other built-in predicates are necessary in some application areas, for example, to express functional dependencies. For "consistent query answering" (see Section 15) [ABK00] explicitly mention "built-in predicates that have infinite extensions, identical for all database instances". For resolution, equality is addressed with the means of paramodulation. In the presence of function symbols, [CW94] require equality literals to be flattened. As a fortunate fact for our approach of allowed formulas, [GT91] incorporate equality in their definition of allowed formulas and state that other built-ins can be included analogously. Hence the evaluation-based approach of *pre*CQE can probably adapted to handle equality

and other built-in predicates without much difficulty.
We also remark here that the data modification approach as proposed for *pre*CQE inherently has some difficulties in numerical domains. When for example trying to keep a salary secret, introducing a lie for the salary that however amounts to nearly the same numerical value, is useless. Hence, to be effective for "numerical lies", a range of secret values has to be declared in the confidentiality policy – possibly with the help of built-in predicates.
Similarly, *pre*CQE can be adapted to support aggregate queries as well as data modification that respects some statistical properties of the data. There is a lot of previous work considering these requirements; for example, "microaggregation" (see for example [LZWJ02]), simulatable binding (see [ZJB08]) or multidimensional range queries in OLAP systems (see [WLWJ03]).

## 14.5 Other First-Order Fragments

We could study the applicability of *pre*CQE to fragments of first-order logic other than the ones examined in the previous parts of this thesis. Appropriate restrictions have to be identified that ensure DB-satisfiability of the constraint set as a whole; one could probably take the conjunction of all constraint formulas and apply a restriction to this conjunction. Most notably, the finite model property of constraints is in general not sufficient to ensure DB-satisfiability. For example, a formula of the Bernays-Schönfinkel class like $\exists x \forall y\, \mathit{Ill}(x, y)$ indeed has $\{\mathit{Ill}(\mathsf{Mary}, \mathsf{Flu})\}$ as a model in the finite domain $\{\mathsf{Mary}, \mathsf{Flu}\}$ (assuming sorts as before). But it is not DB-satisfiable in the infinite domain *dom*. Similarly, the PRQ formulas of [BT98, BEST98] are as expressive as full predicate logic and hence contain infinity axioms. The same might apply to the guarded fragment. Analogously to [BT98, BEST98], we may be able to prove that if a DB-model for a constraint set of PRQ formulas exists, it is found by *pre*CQE; otherwise the algorithm is not guaranteed to terminate.

## 14.6 Relaxation of Infinite Domain Assumption

In all practical implementations, the domain size cannot be infinite but must be finite. In some applications hence quantifiers are eliminated by propositionalization: a universal quantifiers leads to a conjunction of all possible instantiations with domain elements; an existential quantifier results analogously in a disjunction. With an infinite domain a full expansion of constraint formulas clearly is impossible: expansion of a universal quantifier results in an infinite conjunction and expansion of an existential quantifier results in an infinite disjunction. But also with a finite domain a full expansion is only feasible with very small domain sizes. Fortunately, the active domain semantics as well as the finite invention approach show that for the considered restrictions a finite subset of the infinite domain suffices. Indeed, for a fixed finite domain the *pre*CQE algorithm is also applicable (recall that the domain is assumed to be fixed by the database schema): instead of DB-implication for the evaluation function we have to use finite implication (denoted $\models_{\mathit{fin}}$). Our

concern can then be formulated as follows: if the constraints are finitely satisfiable in the given domain and the precondition for lying holds, find a model that is closest to the input instance *db*.

## 14.7 Reliable Database Responses

With the lying method, some database responses may be unreliable in the sense that the user cannot be sure whether a response was lied or not, but the user may have higher interest in definitely correct (and thus reliable) responses. The issue of reliability for censor-based CQE was raised in [BB04b]; there, reliability was defined with respect to a query sequence such that an answer (in the corresponding answer sequence) is reliable if it is satisfied in all the database instances that return the same controlled answer sequence.
We propose to define reliability of responses in the context of inference-proof databases such that the inference-proof instance and all its "possible pre-images" are taken into account. There may be several solutions (that is, distortion-minimal and inference-proof database instances) for a given *db*; on the other hand, for one inference-proof instance $db'$ there may be more than one original instance. Seeing the *pre*CQE computation as a mapping from the input instance to a set of solution candidates (that is, instances with the same distortion distance), reliability for inference-proof databases can now be stated in terms of possible pre-images. Because we assume that the user has no incorrect a priori knowledge and an original database specifies correct information, we also require that a pre-image is a model of *prior*:

---

**Definition 14.1 (Possible pre-image)**
*A database instance* $db^{pre}$ *is a* possible pre-image *of an inference-proof database instance* $db'$*, if* $db'$ *is a distortion-minimal inference-proof database instance for* $db^{pre}$ *(with respect to fixed sets of a priori knowledge prior and potential secrets pot_sec) and* $I^{db^{pre}} \models prior$.

---

For inference-proof database, reliability of a database response can thus defined as follows:

---

**Definition 14.2 (Reliable database response)**
*Given prior and pot_sec, the response* $eval^*(\Phi)(db')$ *to a closed query* $\Phi$ *from an inference-proof database instance* $db'$ *is* reliable *if all database instances* $db^{pre}$ *that are possible pre-images of* $db'$ *return the same response for* $\Phi$*; that is,*

$$eval^*(\Phi)(db') = eval^*(\Phi)(db^{pre})$$

*for all possible pre-images* $db^{pre}$.

---

As we assume that the user knows the policy specification (and his a priori knowledge), he can compute whether a database response is reliable or not. Due to the

finiteness of a database instance and our definition of distortion minimality, the set of possible pre-images for a given $db'$ is finite. One could extend this definition in the fashion of probability-based approaches and determine a probability distribution over the pre-images; hence a probabilistic notion of reliability could be established.

## 14.8 Availability-Preservation with an Explicit Availability Policy

We pointed out in Section 4.5 that not only confidentiality but also availability requirements can be explicitly stated. For *pre*CQE this means that still inference-proofness and distortion minimality via the distance *db_dist* is achieved but additionally the significance of some data is accentuated. Analogous to the confidentiality policy, the availability requirements are declared in an availability policy called *avail*. More precisely, *avail* is a set of formulas of the chosen base language $\mathscr{L}$ – in contrast to the confidentiality policy we also allow open formulas in *avail*. For *pre*CQE we give the availability policy a special, instance-dependent semantics: for each *closed* formula $\Theta$ in *avail*, its truth value (that is, $eval^*(\Theta)(db)$) should be retained in the solution instance $db'$ – if this does not contradict the inference-proofness of $db'$; for each *open* formula $\Theta$ in *avail*, the truth values of all the closed ground instantiations (substitutions of the free variables with constants from *dom*) should be retained in $db'$.
To achieve this, we determine a set of closed formulas by evaluating the policy *avail* in *db* and taking only the positive part of the evaluation with an evaluation function $eval^{pos}$ (in analogy to the positive evaluation for a marked database instance in Definition 9.5). We have to ensure that this is a finite set of closed formulas and thus restrict the availability policy *avail* to safe formulas (that is, those with a finite positive part of their evaluation); as a syntactical condition for this we employ the allowed property of Definition 8.3 once again. In this way, we define the following "availability distance" as the primary measure of availability in a solution instance $db'$: we count the number of closed formulas and the number of ground instantiations of open formulas that have a different evaluation in *db* than in $db'$. It is sufficient to compute the positive evaluation to count these differences.

---

**Definition 14.3 (Availability distance)**
*The* availability distance *of a database instance $db'$ with respect to an availability policy avail of ground and allowed formulas and an input instance db is*

$$avail\_dist(db') := card\Big(\bigcup_{\Theta \in avail} eval^{pos}(\Theta)(db) \;\oplus\; \bigcup_{\Theta \in avail} eval^{pos}(\Theta)(db')\Big).$$

---

We decide to maximize this availability distance in the first place and thus make "availability maximality" our primary optimization criterion:

**Definition 14.4 (Availability maximality)**
*An inference-proof database instance $db'$ is* availability maximal, *iff there is no other inference-proof database instance $db''$ such that*

$$avail\_dist(db'') > avail\_dist(db').$$

Distortion minimality is maintained as a secondary optimization criterion: from all availability-maximal candidate instances we choose one with minimal $db\_dist$. This way, the availability of *avail* entries is respected as good as possible, albeit at the cost of more distorted entries in the solution instance.
We reinforce that inference-proofness – that is, $I^{db'}$ being a model of $C$ – still is our main interest: no potential secrets will be divulged by an availability-maximal solution instance.

For the Branch and Bound approach on the availability distance we need a local lower bound of this distance for each node $v$ in the search tree. We call it $min\_unavail$ and define it using the initial $db_r$:

$$min\_unavail_v := card(\bigcup_{\Theta \in avail} eval_r^{pos}(\Theta) \oplus \bigcup_{\Theta \in avail} eval_v^{pos}(\Theta)).$$

After a ground literal has been marked in a node $v$, the lower bound $min\_unavail_v$ can be calculated in the `MARK` procedure. Based on the fact that only a single ground atom $\gamma$ changes its truth value in one call to `MARK`, $min\_unavail_v$ can be calculated more efficiently than evaluating all *avail* formulas in each marked database instance $db_v$ with $eval_v^{pos}$: evidently, only the *avail* formulas affected by $\gamma$ can potentially increase $min\_unavail_v$.
Lastly, we have to adjust the pruning condition due to a bad lower bound to consider both $min\_unavail_v$ and $min\_lies_v$: a branch is pruned in the `MARK` procedure whenever

$$min\_unavail_v > min\_unavail_{best} \text{ or}$$
$$min\_unavail_v = min\_unavail_{best} \text{ and } min\_lies_v \geq min\_lies_{best}.$$

We lastly illustrate this approach with an example. As the original database instance $db$, we again take

| *Ill* | *Name* | *Diagnosis* |
|---|---|---|
| | Pete | Aids |
| | Mary | Cancer |

| *Treat* | *Name* | *Treatment* |
|---|---|---|
| | Pete | MedA |
| | Mary | MedB |

As a priori knowledge and confidentiality policy we again have

$$\begin{aligned} prior \quad = \quad & \{\forall x(Treat(x, \mathsf{MedA}) \rightarrow Ill(x, \mathsf{Aids}) \vee Ill(x, \mathsf{Cancer})), \\ & \forall x(Treat(x, \mathsf{MedB}) \rightarrow Ill(x, \mathsf{Cancer}) \vee Ill(x, \mathsf{Flu}))\} \end{aligned}$$

$$pot\_sec \quad = \quad \{\exists x\mathit{Ill}(x, \mathsf{Aids}), \exists x\mathit{Ill}(x, \mathsf{Cancer})\}$$

We add an availability policy that says that the ground instantiations for a treatment with MedA and MedB should best be unchanged – for both the cases that such an instantiation is *true* in $I^{db}$ and that it is *false*. This might be justified by the fact that the medication has serious side effects or mutual reactions with other medicine; a distortion of information regarding the medication might thus cause detrimental effects for a patient.

$$avail \quad = \quad \{\mathit{Treat}(x, \mathsf{MedA}), \mathit{Treat}(x, \mathsf{MedB})\}$$

In Figure 14 it is shown how a solution instance is found. We find a unique solution in $v_3$ because we calculate

$$\begin{aligned} &min\_unavail_{v_3} \\ &= card(\{\mathit{Treat}(\mathsf{Pete}, \mathsf{MedA}), \mathit{Treat}(\mathsf{Mary}, \mathsf{MedB})\} \oplus \{\mathit{Treat}(\mathsf{Mary}, \mathsf{MedB})\}) \\ &= 1 \end{aligned}$$

while

$$\begin{aligned} min\_unavail_{v_2} \quad &= \quad card(\{\mathit{Treat}(\mathsf{Pete}, \mathsf{MedA}), \mathit{Treat}(\mathsf{Mary}, \mathsf{MedB})\} \oplus \emptyset) \\ &= \quad 2 \end{aligned}$$

Because of the solution's uniqueness, we do not even have to use *db_dist* here.

## 14.9 Adding Last-Minute-Distortion

With an inference-proof database instance, availability of correct information is enforced globally by distortion minimality. More precisely, in the setting with allowed universal constraints, when a query sequence covers the whole active domain, maximal availability is asserted for it. Yet, if not the whole active domain is covered in a query sequence, $db'$ may return more lies than necessary for this specific sequence: if there is a second distortion-minimal inference-proof database, it might return more correct answers for the query sequence than $db'$ does. But because we assume that the query sequence is not known beforehand and may also be infinite, those dynamic aspects are hard to capture. Yet, for censor-based CQE, that dynamically determines at runtime which responses can be given, it is possible to employ a "last-minute distortion strategy" (see [BB04b]): Depending on a specific query sequence, the CQE system should return correct answers as long as possible before effecting a distortion. This is a heuristics to improve availability for short query sequences in censor-based CQE.
When considering the preprocessing approach of this thesis, with an inference-proof solution instance $db'$, last-minute distortion might be lost for some query sequences but not for others; that is, $db'$ may return lies too early when considering a specific query sequence $Q$. To overcome this restriction of inference-proof databases,

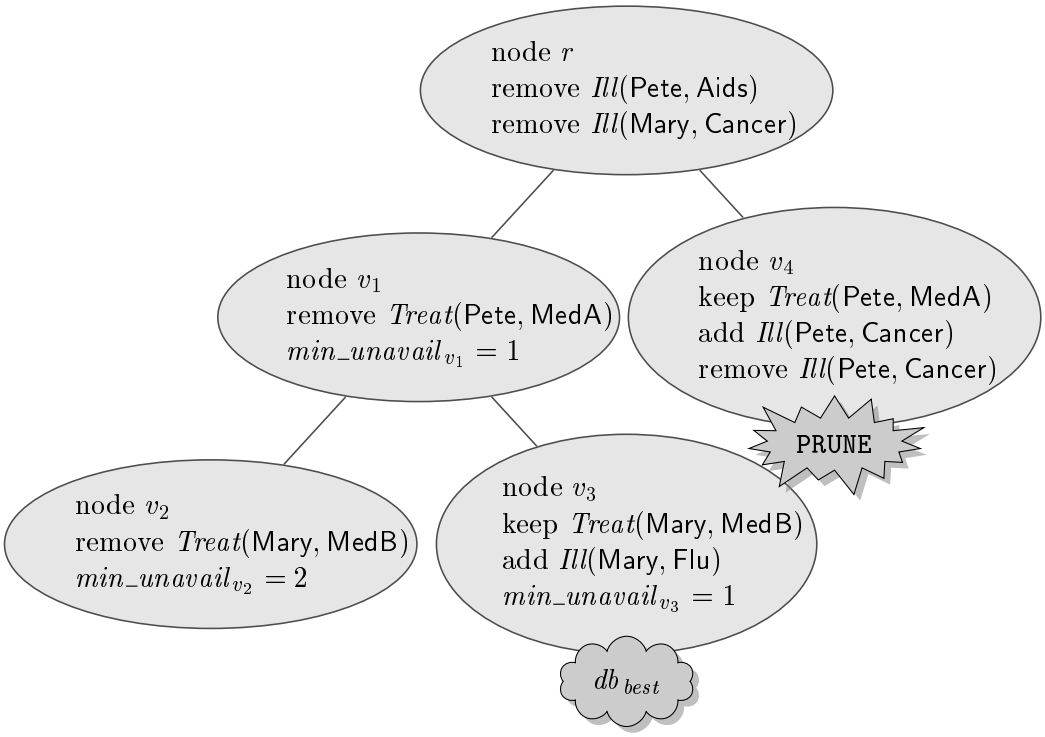


Figure 14: *pre*CQE with explicit availability policy

we propose to store the whole set (or at least a subset) of all inference-proof and distortion-minimal solution instances (with respect to fixed *db*, *prior* and *pot_sec*) whenever this is possible from the point of view of available preprocessing time and storage capacity. All these solution instances ensure confidentiality of the given potential secrets with a minimum of lies; however at runtime when a specific query sequence is handled, the system can choose one solution instance out of the stored set such that a correct answer is returned as long as possible. As a similar concept, [BCFP03] define a model-theoretic semantics for access control that "assigns to a logic program a number (possibly zero) of alternative models, each representing a set of consistent authorizations that can be possibly assigned to subjects".
Note that last-minute distortion is just a heuristics and may be contradictory to query-dependent availability (or "cooperativeness") also in the dynamic censor-based setting: returning correct answers in the beginning may lead to more lies than necessary for a specific query sequence in the long run.

## 14.10 Role-Based Access Control and Authorization Views

In the formal presentations in this thesis we abundantly talked of a "single user". This can easily be extended by introducing roles and specifying a confidentiality policy and a priori knowledge for each role; that is, roles enable modeling groups of

users instead of each single user.
Another way of simplifying the specification of the *pre*CQE input is the utilization of parameters. This mechanism is having a wide repercussion in the access control field; for example, [RMSR04] present authorization views (a special type of parameterized views) as a common means of access control; the virtual private database (VDP) mechanism of the Oracle database system is simpler but similar. The underlying idea is that there is one access control rule defined with some parameters that are instantiated at query time (for example, the user identity or the access time). Access control is enforced by using these parameters in some more or less complicated `where`-clause of a SQL query.
Although parameterized views are inherently a query-time mechanism, we could incorporate parameters into our confidentiality policy of potential secrets as well as the a priori knowledge: they can be declared once by the administrators but automatically have a different instantiation according to each particular user. Hence, *pre*CQE (and CQE in general) can combine parameterization of policies with the logic-based confidentiality properties of inference control; in the opposite direction, CQE can influence the incorporation of a user model into view-based access control.
Conceiving inference-proof database instances as views raises the question in how far the *pre*CQE approach can be pushed back in the direction of a query-time mechanism. Usually a distinction is made between materialized view (data in the view are stored in the database system) or a virtual view (the data in the view are computed at query time). It is an interesting open question if it is possible to maintain the *pre*CQE input as a virtual view; similar problems are tackled by data integration systems (see for example [CCGL02]). It may also be possible to materialize only relevant parts of an inference-proof instance according to a constraint set with parameters.
Changing the Interaction Model and allowing updates has tremendous consequences: when on the one hand updates by the database administrator are allowed, the inference-proof instance has to be updated, too. What is more, it has to be updated without violating confidentiality of secrets. If on the other hand the user is allowed to execute updates on the inference-proof instance, we face the problem of view updates and also have to ensure that no harmful inferences can occur due to an update.

## 14.11 Hierarchical Constraint Solving

In order to give the database administrator greater flexibility when specifying the policies, instead of solving a maximization problem where *prior* and $Neg(pot\_sec)$ have to be fully satisfied and *avail* has to be maximally satisfied, we can extend the algorithm to solve a hierarchical maximization problem: while only *prior* is in hierarchy level 0 (and thus has to be fully satisfied), there can be other hierarchy levels consisting of alternating sets of confidentiality and availability policies that have to be satisfied as good as possible while lower levels take precedence over higher levels. In this way, the administrator can specify fine-grained confidentiality

and availability requirements. In this setting, we can also skip requirement (e) (see Section 7.4) as we now solve a MAXSAT problem for the negations of the potential secrets and not all secrets have to be protected. This approach has been published in a more generalized form for a hierarchical constraint solver in [BBWW07].

## 14.12 Incomplete Databases and Refusal

An incomplete database instance $db$ in the CQE context (see Section 4) is a finite and consistent set of formulas. With this data model we cover null values (via existential quantification) as well as disjunctive information. The evaluation function is then based on logical implication: query evaluation returns the value *undefined* to those closed queries for which neither the query sentence nor its negation are implied by $db$.

$$eval(\Phi)(db) \quad := \quad \begin{cases} true & \text{if } db \models_{FOL} \Phi \\ false & \text{if } db \models_{FOL} \neg\Phi \\ undefined & \text{else} \end{cases}$$

Inference-proofness for incomplete databases has a strong similarity with belief revision or belief contraction techniques (see [CW94] for a well-known belief revision approach that we also describe in Section 15) in the following way: for a potential secret $\Psi$, we could for example revise its negation $\neg\Psi$ into the incomplete original instance $db$ (thus using negation as a lie) or contract $\Psi$ from $db$ (thus using *undefined* as a lie). The protection of an *undefined* value is probably more intricate. In a similar way, data restriction ("refusal") can probably be implemented: whenever a query or its negation is implied by the input instance but not by the contracted instance, the query response is *refused* ("access denied").
However, consistency with the a priori knowledge *prior* has to be ensured without actually revising it into $db$. The notion of belief change (see [DS07, DS04]) might be helpful to tackle this problem: belief change is an approach for simultaneous belief revision (in a consistency based way) and belief contraction (in an entailment based way) for propositional knowledge bases. Optimization criterion for solutions is the inclusion-based maximization of a set of equivalences between the propositional language of the knowledge base and the propositional language of the revision set. Transforming $db$ into an inference-proof incomplete database $db'$ may be seen as a so-called belief change scenario $B = (db, prior, pot\_sec)$ such that $db'$ represents the knowledge

$$Cn(db \cup prior \cup EQ) \cap (pot\_sec \cup \{\bot\}) = \emptyset$$

where $Cn$ is an appropriate consequence operator computing the deductive closure of its input, $db$ is appropriately renamed with a language disjoint to $prior$, $EQ$ is a set of equivalences between the languages and the bottom element $\bot$ avoids inconsistencies if $pot\_sec$ is empty.
Yet, the consequence operator presents us with a new problem: $db'$ must be representable as a finite set of formulas. Hence, belief *base* revision (that is meant

to return a finite belief base as its result) may be appropriate here. Furthermore, [Ros06, CLR03] provide several semantics for query evaluation over incomplete and inconsistent data.

For incomplete databases, we also want to maximize availability by minimizing a distortion distance. Apart from cardinality-based distances (as defined and used in this thesis), preorders on models can be defined by considering the set inclusion of their symmetric differences (see for instance [Win90, CW94] for a description of both methods and a characterization of their local and global behavior); these inclusion-based orders might be reasonable when starting with a set of original models (that is, an incomplete database in our case).

## 15 Related Research Areas

There are several research areas that – although not focusing on security requirements – have influenced the development of *pre*CQE. Yet none of them readily fits the bill of the *pre*CQE problem statement as they either do not consider database instances in an infinite domain or do not follow a distance minimization approach or they apply a different data model as well as different modification primitives. With *pre*CQE we made an effort to combine the expedient features of all of them. We discuss the approaches and their influences on *pre*CQE in the following subsections; Figure 15 illustrates the situation.

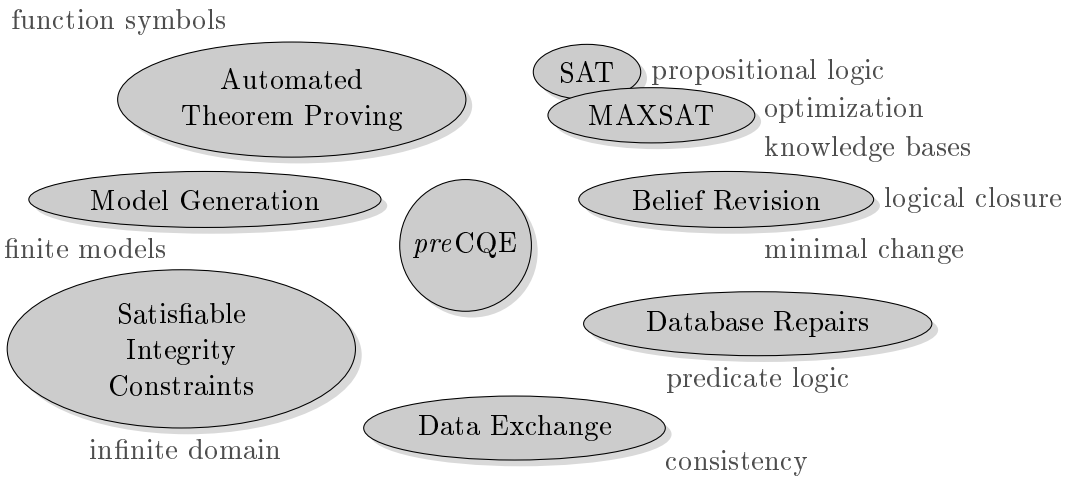


Figure 15: Related research areas

## 15.1 Automated Theorem Proving and Model Generation

Automated theorem proving provides mechanisms that aim to show that a first-order theorem follows from a set of axioms by deriving a refutation from the axioms and the negation of the theorem. Model generation on the other hand strives for (in most cases) finding finite models of a set of first-order formulas. Both topics do not incorporate a notion of "minimal change" as it is the case with distortion minimization in *pre*CQE.
In the propositional context, satisfiability problem solvers (SAT solver) try to find a satisfying interpretation for a set of clauses (i.e. disjunctions of literals). The basis for nearly all non-probabilistic SAT solvers is the so-called DPLL-algorithm (see [DP60, DLL62]). It builds an interpretation step-by-step by assigning propositional variables a truth value with the methods:

1. elimination of one-literal clauses (also called "boolean constraint propagation", BCP): a unit clause (i.e., a clause consisting of just one literal) must be evaluated to *true*
2. affirmative-negative rule: whenever a literal occurs "purely" in the clause set – that is, its conjugate does not occur –, set it to *true*
3. splitting on variables: take one yet uninterpreted variable, set it to *false* (to get one subproblem) and to *true* (to get a second subproblem), and try to find a solution for at least one of the subproblems

Whenever a variable is assigned a value, the set of clauses can be simplified by unit subsumption (if a clause contains a literal that is evaluated to *true*, remove the whole clause) or unit resolution (if a clause contains a literal that is evaluated to *false*, remove this literal from the clause but keep the remaining clause). If there is only the empty set left (which is equivalent to *true*), the current interpretation is satisfying; however, if the clause set eventually contains the empty clause $\square$ (which is equivalent to *false*), the interpretation is not satisfying.
For SAT indeed optimization variants exist: the maximum SAT (MAXSAT) and the partial MAXSAT (PMSAT) problems as well as their weighted extensions. A prototypical implementation of *pre*CQE that is based on SAT solving will be the topic of the upcoming Part V.

Coming back to first-order logic, we present two examples of model generation. [BT98, BEST98] employ the "extended positive tableaux" (EP tableaux) method in their "Satisfiability Checking for Integrity Constraints" (SIC) system. If a set of database constraints is finitely satisfiable, SIC generates a minimal finite model; if they are unsatisfiable, it refutes the constraints. They consider a logical language with a denumerable number of constants but no function symbols. They define term interpretations to be Herbrand interpretations – with the difference that the interpretation's domain is the set of all constants occurring in ground atoms that are satisfied by the interpretation. Therefore, if a term interpretation satisfies a

set of formulas, the constants (in the formula set) outside of the domain of the interpretation are mapped to a special constant.
The SIC database constraints are a set of syntactically restricted formulas – so called "positive formulas with restricted quantification" (PRQ formulas) having the same expressive power as unrestricted first-order logic. Arbitrary formulas can be transformed into PRQ formulas by restricting quantified variables with the help of an explicit predicate ranging over the domain of the term interpretation and all constants occurring in the set of formulas. Four tableaux expansion rules are applied to unsatisfied formulas in the constraint set either until all branches contain the atom $\perp$ (denoting unsatisfiability), or a branch with all formulas satisfied has been constructed (denoting finite satisfiability), or rule applications never stop (denoting infinite satisfiability). Also fairness of a tableau is considered meaning that effect of each possible rule eventually takes place. The tableaux method is shown to be sound and complete for unsatisfiability, as well as sound for satisfiability and complete for finite satisfiability. We expounded previously in Section 10 that the handling of existential formulas in *pre*CQE originates from the according EP tableaux rule. The restriction to process only unsatisfied formulas (hence "violated" constraints in the *pre*CQE terminology) is also applied in *pre*CQE. Yet SIC and *pre*CQE differ in the model finding strategy: SIC is a tableaux system and as such has very little in common with the *pre*CQE search tree; what is more, the SIC system does not make use of a query evaluation function, nor does it follow a minimal change approach (that is, minimize a distance between database instances).
FDPLL [Bau00] and its successor Darwin with the Model Evolution Calculus [BT05, BT08] provide a procedure that generates models of possibly infinite size. It processes clauses in predicate logic (without equality). Models are represented as sets of non-ground formulas. The authors state that Darwin is a decision procedure for the Bernays-Schönfinkel class. The main achievement of both FDPLL and Darwin is that they lift the rules of the propositional DPLL method to first-order logic (for example, the splitting rule). Both aim purely at model generation – hence, they do not consider minimal change either. The work on FDPLL contains a hint on the use of semantic trees that led to the author's involvement with the exposition in [CL73]. Anyway, the work on FDPLL does not elaborate any proofs with the help of semantic trees as the ones that can be found in this thesis. We reinforce the point that the Model Evolution Calculus is designed for clause logic; in contrast, *pre*CQE allows a much more general syntax of constraint formulas. We already discussed in Section 14.3 that incorporation of lifted splitting into *pre*CQE is desirable but then the computation of the distortion distance has to be supported accordingly.

## 15.2 Database Repairs and Data Exchange

Both, database repairs [ABK00, BC03, CM05, BBFL08] and data exchange [AK08, FKMP05, MS07], are concerned with restoring consistency with respect to a set of constraints. Database repairs emerged in the context of consistent query answers (CQA): when a user queries a database that is inconsistent with respect

to a set of constraints, identify those responses that are consistent with the constraints. Most approaches use the smallest symmetric set difference to ensure only a minimal amount of changes that make a database instance consistent with the constraints. In this way, their approach is identical to the inclusion-based distance measure used in belief revision. In [BBFL08] a generalization that also considers cardinality-based distances is given. Such a minimally changed database instance is called a "repair". A consistent database answer (for an open query) is then defined as a tuple of constants for which the query is satisfied in all repairs of the original (inconsistent) database instance; similarly, *true* or *false* can be consistent answers for a closed query. The approaches are mostly restricted to universal constraints (sometimes only denial constraints) and they only allow deletions of tuples to repair a database. Extensions to inclusion dependencies with existential quantifiers are presented in [BB04c, CM05]; but for their purposes the authors only deem the addition of null values as appropriate. This makes the resulting database repairs incomplete: "[a]llowing repairs constructed using insertions makes sense only if the information in the database may be incomplete" ([CM05]). Hence what makes the main difference between database repairs and *pre*CQE is the incorporation of tuple additions and the treatment of existential quantifiers; the result of *pre*CQE remains in the complete data model. On the other hand, [CM05] are interested in tractability and identify minimal classes of constraints for which their consistent query answer problem lies outside of the complexity class P – while we are interested in finding a maximal set of constraints that is decidable and soundness and completeness of *pre*CQE can be ensure.

Data exchange on the other hand materializes data from sources into a target based on a set of "source-to-target" dependencies (and possibly respecting an additional set of target dependencies). [FKMP05] apply the chase procedure to the dependencies to take over data from the sources into the target. As already mentioned in Section 12, the notion of weakly acyclic TGDs originates from them. But during their chase procedure, existential quantifiers lead to introducing null values in the target which makes the target incomplete; this again in contrast to how *pre*CQE handles existential quantifiers. Furthermore, every time the body of a TGD is satisfied, the chase adds tuples in the head to the target; in contrast, with *pre*CQE it is also possible to delete tuples affected by the body to achieve consistency. In contrast to [FKMP05], *pre*CQE is able to handle additionally existential and denial constraints.

## 15.3 Belief Revision

Belief revision is about accommodating a set of knowledge (a "knowledge base" or "belief base") to new – possibly contradictory – information. A knowledge base corresponds to our notion of an incomplete database and hence as we require our *pre*CQE result to be complete is not totally applicable. Nevertheless, minimal changes semantics is inherent to belief revision and most notions of distances on models originate from it. But research on belief revision mostly focused on theo-

retical properties ("postulates") rather than practical implementations. In the following, we present two of the few algorithmic approaches here. Moreover, we mentioned already in Section 14.12 the concept of belief change scenario [DS07, DS04] for propositional knowledge bases.

[CW94] implement model-based belief revision: they minimize the distances between models of the knowledge and the new information. They use negation ("inconsistencies") to determine minimal flippings of ground literals in order to establish consistency with the new information. Inconsistencies are akin to our violation sets but require putting the formula into CNF beforehand – which we with *pre*CQE do not. After a foundational propositional algorithm, they propose a first-order algorithm including the handling of function symbols. Yet this approach remains preliminary as the universe is assumed to be finite and thus allows for elimination of quantifiers.

[Wil97] widen the extent of belief revision to include rankings among the beliefs: the higher the rank, the more firmly the belief is held. As a result, forfeiting several lower ranked beliefs might in total be better than withdrawing a higher ranked belief. They call the input of their algorithm a "theory base". The algorithm incorporates a procedure that moves formulas up or down in the ranking; yet, it does not alter the syntactical appearance of the formulas. Hence, the outcome of the revision depends on the syntax of the input (not on its models). While this approach seems to be quite powerful for its purpose of belief revision (dealing with inherently incomplete knowledge bases), it is not the right tool for *pre*CQE in complete databases.

## Summary of Part IV

Several open issues that extend *pre*CQE to meet special needs were regarded in this part. Non-ground splitting and "numerical lies" give some room for optimization. More abstract goals like reliability and explicit availability requirements were handled appropriately. The incorporation of updates, incomplete information and refusal are major open problems.

Last but not least, a connection between *pre*CQE and some related approaches was drawn. The differences between *pre*CQE and the presented research areas show that *pre*CQE indeed adds new results to the topic.

# V. Implementation and Analysis of a Prototype

## Contents



```
This error message was generated by an \errmessage
command, so I can't give any explicit help.
Pretend that you're Hercule Poirot: Examine all clues,
and deduce the truth by order and method.
```

– *LaTeX error message*

# 16 Propositional Logic

At our department, we implemented a prototypical application that accepts propositional input. The reason why we confine ourselves to propositional logic is that we can use up-to-date highly efficient SAT solving technology (which we introduce in Section 18) for the computation of *pre*CQE solution instances.
We assume to have a propositional language based on an infinite number of propositional variables (the propositional "alphabet" $\mathcal{A}$). This is equivalent to banishing quantifiers and variables from formulas in our language $\mathscr{L}$ of predicate logic. This equivalence is shown in [GA07]. Beyond the restrictions of FOL that Gammer and Amir define in their article, we do not even allow equality and free variables; that is, we use a subset of their $GL^{=}$ logic. In this case, each propositional variable from $\mathcal{A}$ can be identified with a ground atom of $\mathscr{L}$. This has the following effects on our input formulas and thus the constraint set $C$: While *db* is still defined as a set of ground atoms, *prior*, *pot_sec* and *avail* are now restricted to contain only ground formulas with a finite number of conjunctions and disjunctions. Furthermore, SAT solving normally refers to input formulas in CNF such that all *pre*CQE input formulas have to be converted into an equivalent or equisatisfiable set of "clauses"; recall that a clause is a disjunction of literals.

One crucial point for the efficiency of the algorithm is to note that only the finite number of ground atoms (in other words, the finite number of corresponding propositional variables) that are contained in *prior*, *pot_sec* or *avail* have to be considered when searching for an inference-proof and distortion-minimal solution instance; we will call these ground atoms "decision atoms" (or "decision variables" in the propositional way of talking):

---

**Definition 16.1 (Decision atoms / decision variables)**
*If prior, pot_sec and avail contain only propositional formulas, the* decision atoms *(or equivalently,* decision variables*) are*

$$Dec := \{\gamma \mid \gamma \text{ is a propositional variable that occurs in } prior, pot\_sec \text{ or } avail\}$$

---

We can be sure that upon execution of *pre*CQE, $min\_lies_v$ is never greater than the number of the decision atoms: in the worst case, interpretations for all decision atoms have to be swapped in $db'$ but not more than these. Interpretations for the remaining – non-decision – atoms can remain unchanged in $db'$ which gives us a distortion distance of 0 for them; this is the best value we can achieve in terms of distortion minimization. More formally, for any node $v$ in the *pre*CQE search tree,

$$min\_lies_v \leq card(Dec) \tag{10}$$

This way we have found an upper bound for the distortion distance in the propositional case.

**Corollary 16.2 (Upper bound of distortion distance)**
*If prior, pot_sec and avail contain only propositional formulas, the distortion distance for any inference-proof instance $db'$ can be bounded by*

$$db\_dist(db') \leq card(Dec)$$

That is, in Line 1.5. of the algorithm, $\infty$ can be replaced with the value $card(Dec)$ for $min\_lies_{best}$ without changes in the execution of the algorithm.

*Example 16.3:* Example 7.5 can be transformed into a propositional problem as follows: The database administrator specifies the infinite alphabet of propositional variables

$$\begin{aligned}\mathcal{A} = \{\ & \texttt{pete\_aids}, \texttt{pete\_cancer}, \texttt{pete\_flu}, \ldots, \\ & \texttt{mary\_aids}, \texttt{mary\_cancer}, \texttt{mary\_flu}, \ldots, \\ & \texttt{lisa\_aids}, \texttt{lisa\_cancer}, \texttt{lisa\_flu}, \ldots, \\ & \texttt{pete\_medA}, \texttt{mary\_medA}, \texttt{lisa\_medA}, \ldots, \\ & \texttt{pete\_medB}, \texttt{mary\_medB}, \texttt{lisa\_medB}, \ldots \ \}\end{aligned}$$

As the original database instance, we then have

$$db = \{\texttt{pete\_aids}, \texttt{mary\_cancer}, \texttt{pete\_medA}, \texttt{mary\_medB}\}$$

The user administrator *useradm* declares the user's a priori knowledge in *prior* as propositional formulas; due to the abandonment of quantifiers, only a finite amount of propositions can be declared:

$$\begin{aligned}prior = \{\ & \texttt{pete\_medA} \rightarrow \texttt{pete\_aids} \vee \texttt{pete\_cancer}, \\ & \texttt{pete\_medB} \rightarrow \texttt{pete\_cancer} \vee \texttt{pete\_flu}, \\ & \texttt{mary\_medA} \rightarrow \texttt{mary\_aids} \vee \texttt{mary\_cancer}, \\ & \texttt{mary\_medB} \rightarrow \texttt{mary\_cancer} \vee \texttt{mary\_flu} \ \}\end{aligned}$$

Analogously, the security administrator *secadm* specifies the potential secrets as propositional formulas:

$$pot\_sec = \{\texttt{pete\_aids}, \texttt{pete\_cancer}, \texttt{mary\_aids}, \texttt{mary\_cancer}\}$$

We see that

$$\begin{aligned}Dec := \{\ & \texttt{pete\_aids}, \texttt{pete\_cancer}, \texttt{pete\_flu}, \\ & \texttt{mary\_aids}, \texttt{mary\_cancer}, \texttt{mary\_flu}, \\ & \texttt{pete\_medA}, \texttt{pete\_medB}, \texttt{mary\_medA}, \texttt{mary\_medB}\}\end{aligned}$$

Thus, the distortion distance will be less or equal to $card(Dec) = 10$. Analogously to Example 7.14, we obtain the inference-proof and distortion-minimal solution instances $db'_1 = \emptyset$ and $db'_2 = \{\texttt{mary\_medB}, \texttt{mary\_flu}\}$ each with distortion distance 4. ♢

# 17 Propositional Encodings

The Branch and Bound approach for propositional logic can be encoded by a transformation of the input constraints such that the distance value need not be maintained explicitly. More precisely, *pre*CQE for propositional logic can be seen as a variant of an optimization problem for the satisfiability (SAT) problem. In the upcoming sections we present three possible representations of *pre*CQE as SAT optimization problems: MINONES, weighted MAXSAT and weighted PMSAT.

## 17.1 Translation to MINONES Without Availability Policy

The "minimum ones" MINONES problem is the problem of finding a model for a set of propositional clauses and in the process minimizing the number of propositional variables that are interpreted with *true* (see [Jon97, Kan94, HS92] for analysis and applications of MINONES and its generalization "minimum distinguished ones" MINDONES). A *pre*CQE input can be transformed into a MINONES input; then, explicit maintenance of the distortion distance is unnecessary because the distance value can immediately be read off from the output. The MINONES transformation depends on the evaluation of decision atoms in the input instance $db$.

In detail, each decision atom $\gamma$ in a formula of the constraint set $C$ is replaced by a literal (containing a new propositional variable $X_\gamma$ from a new propositional alphabet $\mathcal{A}'$ disjoint from $\mathcal{A}$) as follows:

$$\begin{aligned} replace &: Dec \mapsto \mathcal{A}' \\ replace(\gamma) &= \begin{cases} \neg X_\gamma & \text{if } eval^*(\gamma)(db) = \gamma \\ X_\gamma & \text{else} \end{cases} \end{aligned}$$

$replace(C)$ is the set of formulas resulting from a replacement of all ground atoms in $C$. An interpretation $I_\mathcal{A}$ of all the newly introduced variables $X_\gamma$ has to be found that satisfies the new constraint set $replace(C)$. Based on this interpretation, the $X_\gamma$ indicate whether the truth valuation of the ground atom $\gamma$ has to be flipped in the solution database instance $db'$ or not; that is, the following is valid:

$$\begin{aligned} eval^*(\gamma)(db') &= \overline{eval^*(\gamma)(db)} \quad \text{if } I(X_\gamma) = true \\ eval^*(\gamma)(db') &= eval^*(\gamma)(db) \qquad\qquad \text{else} \end{aligned}$$

Minimizing the distortion distance for $db'$ while satisfying the constraints $C$ is now equivalent to minimizing the amount of *true*-assignments in an interpretation for

*replace*$(C)$. More precisely, the number of *true*-assignments (that is, "ones") in the resulting interpretation with minimum ones is equal to the distance value *db_dist* of the solution instance $db'$ found with Branch and Bound. A Branch and Bound solver for MINONES can be implemented analogously to the one presented in Section 9; yet, instead of maintaining the distortion distance, the amount of *true*-assignments is counted and minimized.
While the (decision version of) the general MINONES problem is NP-complete, it is well known that plain SAT solving for a Horn formula (that is, a set of clauses where each clause contains at most one positive literal), is decidable in polynomial time, and thus is in the complexity class $P$; it can even be solved in time linear with respect to the formula length, if a special data structure is used. This plain SAT algorithm for Horn clauses consists of initializing an interpretation with all propositional variables interpreted as *false* and then – starting from the unit clauses – setting as few variables as possible to *true*. This effectively minimizes the amount of *true*-assignments in the interpretation and thus results in a MINONES interpretation of the Horn clauses.
What advantage can we draw from this for *pre*CQE? Notably, we can assure that if for fixed inputs *db* and $C$ the MINONES formulation yields a set of Horn clauses, a solution instance can be found in linear time – provided that the mentioned efficient data structure is employed. As this formulation depends on the input instance *db*, this case occurs if in all the input clauses $c$ from $C$ at most one literal is in conflict with *db* (that is, evaluated as *false* in *db*), and necessarily at least one violated unit constraint has to be in $C$. Still, as the occurrence of this case depends on actual inputs *db* and $C$, we cannot give a general syntactic condition for this to happen. A MINONES solver for the Horn case runs without splitting.
In Example 16.3, if we change *db* to $db^{Horn}$ that additionally contains the ground atoms `pete_cancer` and `mary_flu`, we have an example for the Horn clause case: each of the constraints only has at most one conflicting literal with respect to $db^{Horn}$. To find the solution instance $db'$ (which effectively is the empty database in this example), only unit propagation has to be applied.

## 17.2 Translation to W-MAXSAT and PMSAT

Another way to encode the problem without explicitly maintaining the distortion distance is to transform it into a weighted maximum satisfiability (W-MAXSAT) problem. Here it is crucial to see the constraints $C$ as a set of clauses. Each clause has an associated non-negative integer as a weight. The optimization function of W-MAXSAT is to maximize the sum of weights of satisfied clauses in an interpretation. Giving the $C$-clauses a sufficiently higher weight than the *db*-clauses ensures that the resulting interpretation satisfies the constraints.

When considering the finite set *Dec* of decision atoms, *pre*CQE without availability policy can be represented by a W-MAXSAT problem with the following weights for the clauses. Let $n$ be the number of decision atoms $n := card(Dec)$. We make

the clauses in $C$ so called hard constraints (that necessarily have to be satisfied) by giving each constraint in $C$ a weight of $n+1$. One more type of clauses is necessary: All decision atoms have to be transformed to soft constraints with respect to their evaluation in $db$. That is:

$$eval^*(Dec)(db) := \bigcup_{\gamma \in Dec} eval^*(\gamma)(db)$$

is the set of soft constraints that all have weight 1. In this setting, any solution that violates a hard constraint but instead satisfies more soft constraints has lower weight than the one that satisfies more hard constraints.
When we have found an interpretation $I_{MAX}$ for the W-MAXSAT input, the *pre*CQE solution instance $db'$ can be derived from it as follows (assuming that all hard constraints are indeed satisfied):

$$db' := \{\gamma \mid \gamma \in Dec \text{ and } I_{MAX}(\gamma) = true\} \ \cup \ (db \setminus Dec)$$

That is, for decision atoms their interpretation is dictated by $I_{MAX}$; for all non-decision atoms the interpretation is the same as in the input instance $db$. The distortion distance $db\_dist(db')$ is equal to the number (or in this case equivalently the weight) of unsatisfied soft constraints.
It is quite interesting to note that without availability policy non-CNF constraint formulas in $C$ can also be handled by this procedure: although a non-CNF formula in general must be transformed into several clauses(in order to be processable by a W-MAXSAT solver; see Section 18 below), each of these clauses receives the weight $card(Dec) + 1$; that is, all these clauses have to be fulfilled by the W-MAXSAT solution. The soft constraints – which do not have to be satisfied totally – are a set of ground literals and thus are not modified in the CNF representation. Thus, we do not face the problem of loss of structure in this case.
In Example 16.3, all constraint formulas (that are in CNF when replacing the material implication $\rightarrow$) have weight 11 in a W-MAXSAT translation of the problem. As soft constraints we have `pete_aids`, `mary_cancer`, `pete_medA` and `mary_medB` as well as $\neg$`pete_cancer`, $\neg$`pete_flu`, $\neg$`pete_medB`, $\neg$`mary_aids`, $\neg$`mary_flu` and $\neg$`mary_medA` each with weight 1.

The translation of a *pre*CQE input with explicit availability policy into a W-MAXSAT input is also possible but more involved: First, a third, intermediate weight has to be determined for the formulas in $avail$. Second, recall that the semantics of the availability policy is that only a maximum but possibly not all of the formulas in $eval^*(avail)(db)$ can be satisfied in the solution instance $db'$ (we use $eval^*(avail)(db)$ as an abbreviation for $\bigcup_{\Phi \in avail} eval^*(\Phi)$). If we take a formula $\theta$ out of $eval^*(avail)(db)$ and determine its CNF representation $cnf(\theta)$ (in order to be processable by a W-MAXSAT solver), all the clauses (that is, conjuncts) of $cnf(\theta)$ have to be treated as "belonging together" when counting their weight. We can achieve this with the help of auxiliary 0-ary predicate symbols (or simply, auxiliary

propositional variables) denoted $S_\theta$. For each formula $\theta$ in $eval^*(avail)(db)$, $cnf(\theta)$ is transformed as follows:

1. To each clause $c$ of $cnf(\theta)$ conjoin $\neg S_\theta$ which gives us $c \vee \neg S_\theta$
2. Add these augmented conjuncts to $C$

Now we can start assigning weights to clauses. For the decision atoms nothing changes: $eval^*(Dec)(db)$ is the set soft constraints at the lowest level with weight 1. Then we add a second level of soft constraints with intermediate weight: for each $S_\theta$, we create a clause $S_\theta$ with weight $card(Dec) + 1$; we call them the "auxiliary constraints" or "auxiliary clauses" from now on. Finally all the clauses in $C$ (including the newly added, augmented *avail*-clauses) have as weight the sum of the weights of all the constraints at lower levels plus 1: $(card(avail) \cdot (card(Dec) + 1) + card(Dec)) + 1$. We can show that a solution of this W-MAXSAT input represents an inference-proof, availability-preserving and distortion-minimal propositional solution instance for the *pre*CQE input. The main argument is that whenever an auxiliary variable $S_\theta$ is *true*, all clauses of $cnf(\theta)$ also have to be *true* in order to satisfy the augmented hard constraints created from $cnf(\theta)$ and thus $\theta$ is satisfied in $db'$; yet, if $S_\theta$ is *false*, all augmented hard constraints created from $cnf(\theta)$ are already satisfied without satisfying each clause of $cnf(\theta)$ and thus $\theta$ is not satisfied in $db'$.
We could also specify the auxiliary clauses and the augmented clauses based on the DNF representation $dnf(\theta)$ for $\theta \in eval^*(avail)(db)$; this is advantageous if $dnf(\theta)$ is shorter than $cnf(\theta)$. We do not go into detail here; essentially, for every formula in *avail* one auxiliary clause is created with intermediate weight and several hard constraints are added to $C$ that represent the formula $\theta \in eval^*(avail)(db)$. A full account of the technical details can be found in the technical report [Tad08].

W-MAXSAT can be simulated by plain MAXSAT (without weights) by simply duplicating clauses according to their weight and then maximizing the number of satisfied clauses. Internally, many MAXSAT solving programs (see Section 18 below) actually implement a Branch and Bound algorithm for propositional input in CNF. Interestingly, MAXSAT is still in NP if the input consists only of propositional Horn clauses whereas SAT for Horn clauses is in $P$.

"Weighted Partial MAXSAT" (W-PMSAT) is a variant of W-MAXSAT where some clauses can explicitly be designated as hard constraints that necessarily have to be satisfied. Its advantage is that optimization has to be done only on the soft constraints. Internally, with the W-PMSAT input format this is done by specifying the weight of a hard constraint explicitly in the input; all clauses with this weight (or above) are satisfied with the plain SAT procedure without referencing their weight. Put differently, the disadvantage of the W-MAXSAT input format is that the weights for the hard constraints are maintained and summed up explicitly and suboptimality of a solution with an unsatisfied hard constraint is possibly detected very late, because probably solutions where some hard constraints are not satisfied

are tried first; these suboptimal solutions accordingly have a lower weight than the precomputed lower bound in the W-PMSAT input format.
For soft constraints, the weights for the W-PMSAT encoding for a *pre*CQE input are equal to the ones in the W-MAXSAT encoding: Without an availability policy, $C$ is the set of hard constraints; $eval^*(Dec)(db)$ is the set of soft constraints with weight 1. With an explicit availability policy, $C$ and the augmented clauses for the *avail*-formulas comprise the set of hard constraints; the auxiliary constraints at the intermediate level and the soft constraints from $db$ at the lowest level are weighted in analogy to the W-MAXSAT encoding. For our test cases (see Section 18), a problem in the W-MAXSAT encoding performed worse than in the W-PMSAT encoding.

## 18 A *pre*CQE Implementation for Propositional Logic

We have seen in Section 16 how a *pre*CQE problem based on finite, quantifier-free input formulas is reduced to the purely propositional case. More precisely, in Section 17.2 it could be seen how a reduction to the W-MAXSAT and the W-PMSAT problems can be made. In recent years, propositional SAT solving has seen a huge improvement in performance. Several highly efficient implementations take part in the yearly SAT competition (in conjunction with the SAT conference). We should reinforce the point that SAT solvers are just interested in pure satisfiability of propositional CNF-formulas without any optimization. Luckily, as a subpart of the SAT competition there also is a "MAXSAT evaluation" that includes competition categories for both the W-MAXSAT and W-PMSAT problems. As a consequence, there are several MAXSAT solvers based on the same technology that employ a Branch and Bound strategy for propositional input. While the SAT competition is already quite established, the MAXSAT evaluation has been organized just for the third time in 2008. This shows that the interest in efficient solving strategies for this optimization problem has come up very recently.

We wanted to apply this highly efficient MAXSAT technology to our problem and benefit from up-to-date solver implementations. To this end, at our department Cornelia Tadros developed a program that translates propositional *pre*CQE input formulas into a W-MAXSAT or a W-PMSAT instance. The program offers the following functionality:

1. It offers a convenient graphical interface for the specification of the input ($db$, *prior* and *pot_sec*) and the presentation of the solution database.
2. It transforms the specified input into a W-MAXSAT or W-PMSAT instance.
3. It transforms this input into the input format of the selected solver.

4. It calls the selected solver on this instance (in W-MAXSAT or W-PMSAT encoding).

5. It measures the runtime of the whole computation as well as the runtime for the solver alone.

6. It transforms the solver output into the solution instance $db'$.

The graphical interface can be found in the package called *pre*CQE *view*, all other procedural parts in the package *pre*CQE *core* as visualized in Figure 16. Figures 17 and 18 show screenshots of the program with *db* and *prior* as well as *db* and $db'$. As the input format we chose the TPTP format for first-order formulas (see [Sut07]); it is a standard format for Automated Theorem Proving and comes along with a set of libraries to convert first-order input into several solver specific formats.

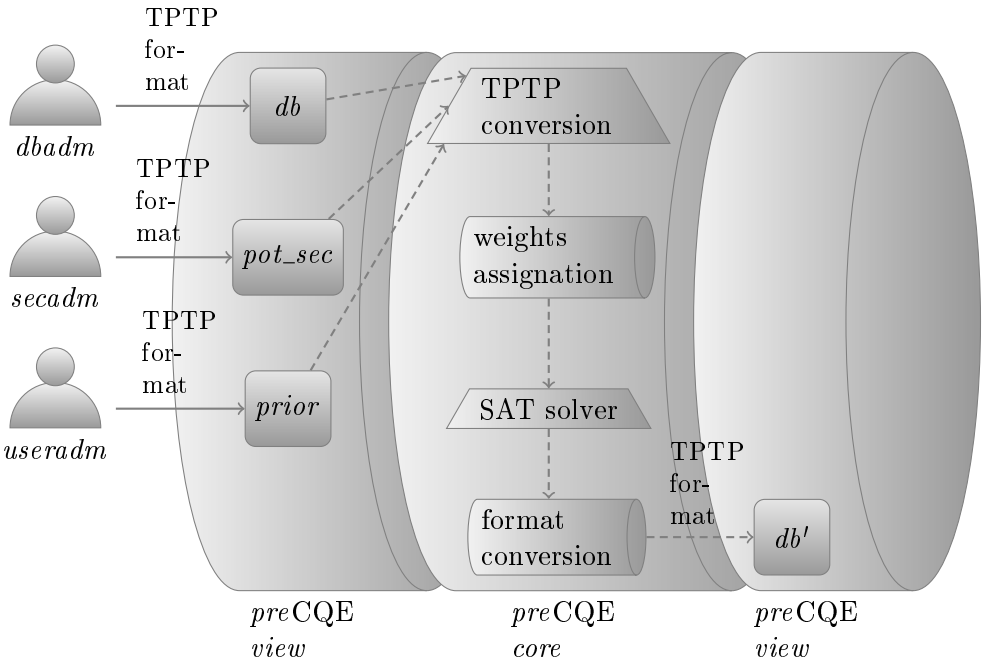


Figure 16: Implementation of Prototype

Our program has been tested with three different solvers for both the W-MAXSAT and the W-PMSAT problem:

- MiniMaxSAT (see [HLO08])
- MAX-DPLL (as part of the SAT solver Toolbar; see [LHdG08])
- SAT4J (`http://www.sat4j.org/`)

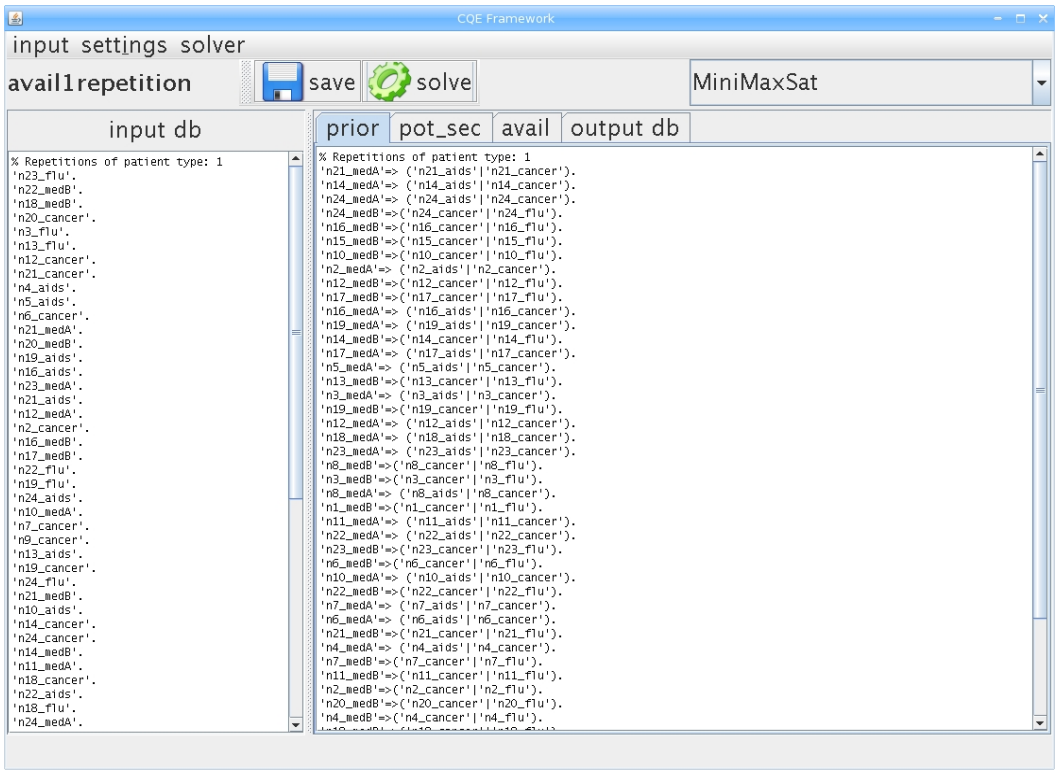


Figure 17: Screenshot with *db* and *prior*

The technical advantage of all these solvers is that they are freely available. MiniMaxSAT is run on a Linux system while we executed Toolbar on a Solaris platform. SAT4J is written purely in Java.

MiniMaxSAT is one of the most efficient solvers of the 2007 and 2008 MAXSAT evaluation; the number of instances solved and the runtime evaluation are competitive. Toolbar performed well in the 2007 and 2008 evaluations, too. During our test runs it however showed difficulties with larger inputs; already for small input sizes it had to be forcefully stopped without having found a solution. SAT4J in comparison is the worst solver in nearly all the categories. This might be compensated by its platform-independence thanks to the Java technology: it can be run on nearly all operating systems, albeit for relatively small input instances.

As already mentioned, TPTP offers conversion libraries that transform an input in TPTP format into other input formats – to wit, all usual formats of current automated theorem provers. The W-MAXSAT solvers we chose are all able to process the DIMACS `wcnf-maxsat` format such that the input *db*, $Neg(pot\_sec)$ and *prior* are converted into this format by calling the external TPTP conversion library. Afterward, the weights of the MAXSAT clauses are calculated internally

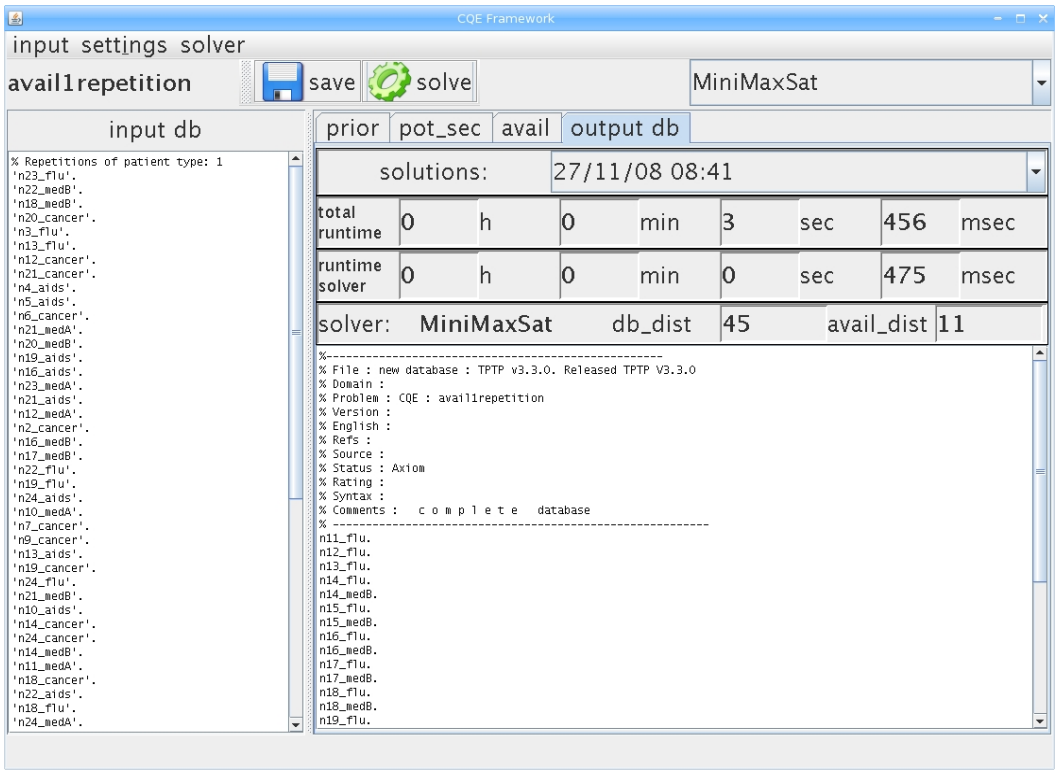


Figure 18: Screenshot with $db$ and $db'$

and set for each clause as described above. With this step, the CQE input has been fully transformed into a W-MAXSAT instance. On this instance, an external W-MAXSAT solver is run to find an optimal solution. The runtime of the solver is internally recorded. When the solution is found, the TPTP output specifies which ground atom of the *pre*CQE input was mapped to which internal propositional variable. The *pre*CQE core uses this information to translate the TPTP output into a *pre*CQE output instance $db'$.

## 19 Test Cases

The MAXSAT evaluation is – among others – run on a benchmark of W-MAXSAT and W-PMSAT instances; more precisely, the benchmark consists of three separate subcases: randomly generated, crafted and (only for W-PMSAT) industrial instances. Unfortunately, neither of these cases readily fit our requirements such that the evaluation results could not be transferred plainly to the CQE case. Thus, to test our prototype we made an effort to simulate problems specific to the database domain. Yet, it was not an easy task to generate appropriate test data. We started

with tests for the previously used example medical records that we gradually expanded and then moved on to tests for cascading constraints. All our test runs below have been performed with MiniMaxSAT to achieve optimal results. Toolbar performed much worse than MiniMaxSAT and seems to have a worse memory management because test often aborted due to insufficient memory. SAT4J showed similar problems for which probably the slower Java technology can be blamed.
The tests were run on several standard desktop PCs with a Linux operating system. As the tests were run with differently sized inputs, for every input size we tested 10 randomly permuted instances to avoid a bias caused by the input order. The runtime graphs below show the average runtime taken from all 10 instances per size as well as the deviation of the individual running times.

## 19.1 Medical Record Tests

The first tests are a generalization of Example 7.5 and its propositional version in Example 16.3. At the outset, we chose 3 constants of the sort *Diagnosis* and 2 constants of the sort *Treatment* to take part in decision atoms; the amount of constants of the sort *Name* in the decision atoms is gradually increased. For one fixed *Name* constant there are $2^3 = 8$ possible combinations with a *Diagnosis* constant; that is, we have 8 decision variables for the combination of a patient's name with a diagnosis. Similarly, there are $2^2 = 4$ possible combinations of a *Name* constant with a *Treatment* constant; that is, we have 4 decision variables for the combination of a patient's name with a diagnosis. Hence, in total there $2^{3+2} = 2^5 = 32$ propositional decision variables. The empty combination (that is, only the *Name* value set without any *Diagnosis* or *Treatment*) is not allowed because the combination has to define a full tuple in the database where a name is combined with a diagnosis or a treatment. Hence, there remain 31 permissible combinations. Further, our precondition on *db* demands that entries have to be consistent with *prior*. Consequentially, all combinations with a *Treatment* value set but without one of the demanded *Diagnosis* values are left out; there are 7 of them. Hence, we are left with 24 "patient types" that are consistent with *prior*. They are listed in Table 2 in propositional notation. We used the abbreviations N1 to N24 to denote 24 different patient names. Then we (in the role of the *dbadm*) entered a propositional input instance *db* that contains each patient type exactly once; that is, if the *db* contains for example *Ill*(N1, Aids) in first-order notation, this ground atom is propositionally represented[1] as `'n1_aids'`. Note that there are 66 propositional variables in the propositional *db*.

Then again, the potential secrets and the a priori knowledge are entered (in the roles of the *secadm* and the *useradm*, respectively) in TPTP syntax for each of the 24 patient names as propositional formulas. For example, for N1 the propo-

[1] Actually the exact TPTP syntax is `fof(r0,axiom,'n1_aids').`; we only state the relevant part here.

| Name | *db* entries | Name | *db* entries | Name | *db* entries |
|---|---|---|---|---|---|
| N1 | n1_aids | N2 | n2_cancer | N3 | n3_flu |
| N4 | n4_aids, n4_cancer | N5 | n5_aids, n5_flu | N6 | n6_cancer, n6_flu |
| N7 | n7_aids, n7_cancer, n7_flu | N8 | n8_medA, n8_aids | N9 | n9_medA, n9_cancer |
| N10 | n10_medA, n10_aids, n10_cancer | N11 | n11_medA, n11_aids, n11_flu | N12 | n12_medA, n12_cancer, n12_flu |
| N13 | n13_medA, n13_aids, n13_cancer, n13_flu | N14 | n14_medB, n14_cancer | N15 | n15_medB, n15_flu |
| N16 | n16_medB, n16_aids, n16_cancer | N17 | n17_medB, n17_aids, n17_flu | N18 | n18_medB, n18_cancer, n18_flu |
| N19 | n19_medB, n19_aids, n19_cancer, n19_flu | N20 | n20_medA, n20_medB, n20_cancer | N21 | n21_medA, n21_medB, n21_aids, n21_cancer |
| N22 | n22_medA, n22_medB, n22_aids, n22_flu | N23 | n23_medA, n23_medB, n23_cancer, n23_flu | N24 | n24_medA, n24_medB, n24_aids, n24_cancer, n24_flu |

Table 2: Permissible patient types in *db*

sitional *prior* contains[2] the formulas `'n1_medB'=>('n1_cancer'|'n1_flu')` and `'n1_medA'=> ('n1_aids'|'n1_cancer')` and the propositional *pot_sec* contains `'n1_aids'` as well as `'n1_cancer'`. These entries are entered for all 24 patients; that is, we have 48 entries in *prior*, and 48 entries in *pot_sec*, too. As mentioned previously, all input is permuted at random to make tests independent of the order of input.

We recall that the potential secrets are put into negative form $Neg(pot\_sec)$ and joined with the a priori knowledge in the constraint set $C$ is before starting the TPTP conversion into the `wcnf_maxsat` format or the `wcnf_pmsat` format.
As for the weights, they are calculated in the *pre*CQE *core* package for this example as follows: all the $24 \cdot 5 = 120$ decision variables are transformed into soft constraints receiving the weight 1; the 66 positive entries of *db* as positive literals and the remaining 114 decision variables occurring in *prior* and *pot_sec* as negative literals. All constraint formulas in $C$ receive the weight 121.

For this simplest input, a solution was found in milliseconds. Obviously, we are interested in more meaningful results for databases with much more entries. The general idea for the expansion of our tests was to uniformly repeat the 24 patient types and test up to what number of repetitions a moderate runtime performance can be achieved. So, our first step was to repeat each patient type 10 times (each repetition with a new name) such that we have a *db* with 660 entries, *prior* with 480 entries and *pot_sec* with 480 entries; for 10 repetitions there are hence $24 \cdot 5 \cdot 10 = 1200$ decision variables. We ran tests up to 150 repetitions with 9900 *db* entries, 7200 *prior* and *pot_sec* entries each and 18000 decision variables. Figure 19 shows the runtime for the `wcnf_pmsat` format which performed better than `wcnf_maxsat` for the reasons given in Section 17.2; Table 3 shows the measured running times as well as the number of decision variables and soft and hard clauses. What can be seen is that a huge amount of time is needed for the creation of the `wcnf_pmsat` input – this includes the TPTP conversion, the creation of $Neg(pot\_sec)$, as well as the calculation and assignment of weights – whereas the MiniMaxSAT solver appears quite unimpressed by the increased size of the input.

After these promising results, we made two more test runs with different patient types: a thorough analysis of the patient types reveals that there are four patient types with multiple optimal solutions; namely, N14 (`n14_medB`, `n14_cancer`), N16 (`n16_medB`, `n16_aids`, `n16_cancer`), N20 (`n20_medA`, `n20_medB`, `n20_cancer`) and N21 (`n21_medA`, `n21_medB`, `n21_aids`, `n21_cancer`). We separated them from the remaining 20 patient types that have a unique solution and examined whether the existence of multiple optima would slow down the SAT solver. That is, for the multiple optima case with one repetition we have 12 entries in *db*, 8 entries in *prior* and *pot_sec*, respectively, and $4 \cdot 5 = 20$ decision variables. We tested up to 600

[2] Again, in full TPTP syntax this is `fof(r1,axiom,'n1_medB'=>('n1_cancer'|'n1_flu')).` and `fof(r0,axiom,'n1_medA'=> ('n1_aids'|'n1_cancer')).`

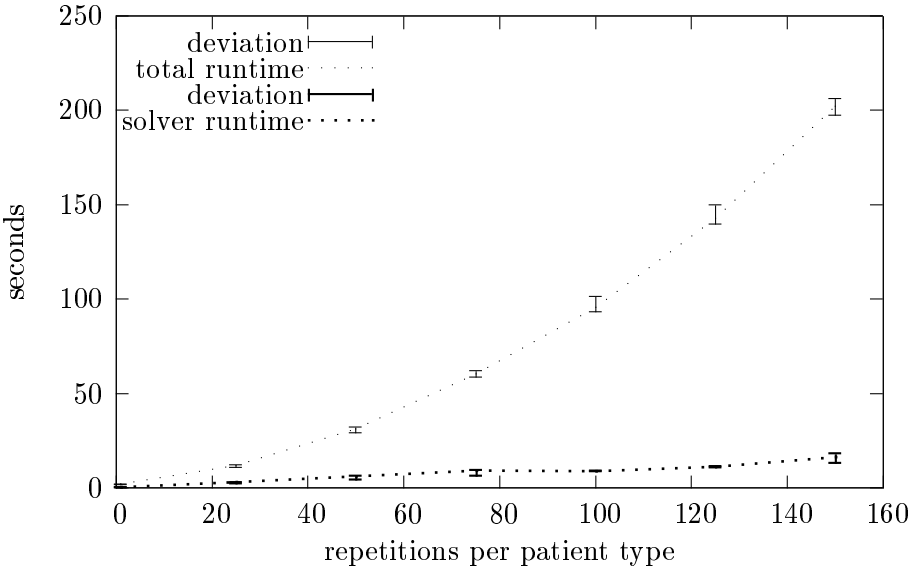


Figure 19: Performance of *pre*CQE for 24 patient types

| | total runtime (msec) | | | solver runtime (msec) | | | dec. | clauses | |
|---|---|---|---|---|---|---|---|---|---|
| rep. | min | max | avg. | min | max | avg. | vars. | soft | hard |
| 1 | 1832 | 2175 | 1930 | 178 | 208 | 184 | 120 | 120 | 96 |
| 25 | 10981 | 12246 | 11974 | 2214 | 3206 | 3092 | 3000 | 3000 | 2400 |
| 50 | 29333 | 32149 | 31304 | 4412 | 6360 | 6135 | 6000 | 6000 | 4800 |
| 75 | 58530 | 62026 | 60459 | 6503 | 9439 | 8991 | 9000 | 9000 | 7200 |
| 100 | 93275 | 101551 | 95792 | 8803 | 9001 | 8902 | 12000 | 12000 | 9600 |
| 125 | 139835 | 150095 | 142843 | 11000 | 11472 | 11171 | 15000 | 15000 | 12000 |
| 150 | 197389 | 206099 | 202067 | 13231 | 18253 | 16429 | 18000 | 18000 | 16800 |

Table 3: Performance of *pre*CQE for 24 patient types

repetitions of these four types; that is up to 7200 entries in *db*, 4800 entries in *prior* and *pot_sec*, respectively, and 12000 decision variables. Figures 20 shows the results of the test runs; Table 4 shows the measured running times as well as the number of decision variables and soft and hard clauses for the multiple optima case.

Then we tested the remaining 20 patient types separately. For one repetition there are 54 entries in *db*, and 40 entries in *prior* and *pot_sec*, respectively, with a total of $20 \cdot 5 = 100$ decision variables. We made tests with up to 300 repetitions; that is, 16200 entries in *db*, and 12000 entries in *prior* and *pot_sec*, respectively, and a total of 30000 decision variables. Figures 21 shows the results of the test runs; Table 5 shows the measured running times as well as the number of decision variables and soft and hard clauses for the unique optima case.

| | total runtime (msec) | | | solver runtime (msec) | | | dec. | clauses | |
|---|---|---|---|---|---|---|---|---|---|
| rep. | min | max | avg. | min | max | avg. | vars. | soft | hard |
| 1 | 1617 | 1945 | 1673 | 78 | 96 | 85 | 20 | 20 | 16 |
| 50 | 4100 | 4343 | 4185 | 1052 | 1070 | 1062 | 1000 | 1000 | 800 |
| 100 | 7477 | 7865 | 7611 | 2041 | 2290 | 2077 | 2000 | 2000 | 1600 |
| 150 | 11744 | 12721 | 11940 | 3033 | 3071 | 3047 | 3000 | 3000 | 2400 |
| 200 | 17111 | 17818 | 17429 | 4035 | 4094 | 4071 | 4000 | 4000 | 3200 |
| 250 | 23468 | 24071 | 23771 | 5121 | 5165 | 5138 | 5000 | 5000 | 4000 |
| 300 | 30835 | 31652 | 31194 | 6108 | 6186 | 6153 | 6000 | 6000 | 4800 |
| 350 | 39215 | 42328 | 40156 | 7179 | 7231 | 7207 | 7000 | 7000 | 5600 |
| 400 | 48748 | 50074 | 49332 | 8225 | 8305 | 8256 | 8000 | 8000 | 6400 |
| 450 | 59310 | 67434 | 60906 | 9180 | 9339 | 9259 | 9000 | 9000 | 7200 |
| 500 | 71220 | 73194 | 72087 | 10528 | 10673 | 10602 | 10000 | 10000 | 8000 |
| 550 | 84340 | 90991 | 86307 | 11627 | 12113 | 11747 | 11000 | 11000 | 8800 |
| 600 | 98902 | 103383 | 100476 | 13205 | 13306 | 13258 | 12000 | 12000 | 9600 |

Table 4: Performance of *pre*CQE for multiple optima

| | total runtime (msec) | | | solver runtime (msec) | | | dec. | clauses | |
|---|---|---|---|---|---|---|---|---|---|
| rep. | min | max | avg. | min | max | avg. | vars. | soft | hard |
| 1 | 1761 | 2135 | 1839 | 140 | 168 | 156 | 100 | 100 | 80 |
| 50 | 21973 | 22788 | 22195 | 3700 | 3747 | 3721 | 5000 | 5000 | 4000 |
| 100 | 68106 | 71299 | 69415 | 7286 | 7465 | 7353 | 10000 | 10000 | 8000 |
| 150 | 140329 | 151528 | 143251 | 10863 | 11248 | 11066 | 15000 | 15000 | 12000 |
| 200 | 236707 | 249837 | 241230 | 14570 | 14978 | 14858 | 20000 | 20000 | 16000 |
| 250 | 373767 | 462364 | 419816 | 19012 | 31093 | 22250 | 25000 | 25000 | 20000 |
| 300 | 514403 | 575354 | 528843 | 23046 | 28447 | 23833 | 30000 | 30000 | 24000 |

Table 5: Performance of *pre*CQE for unique optima

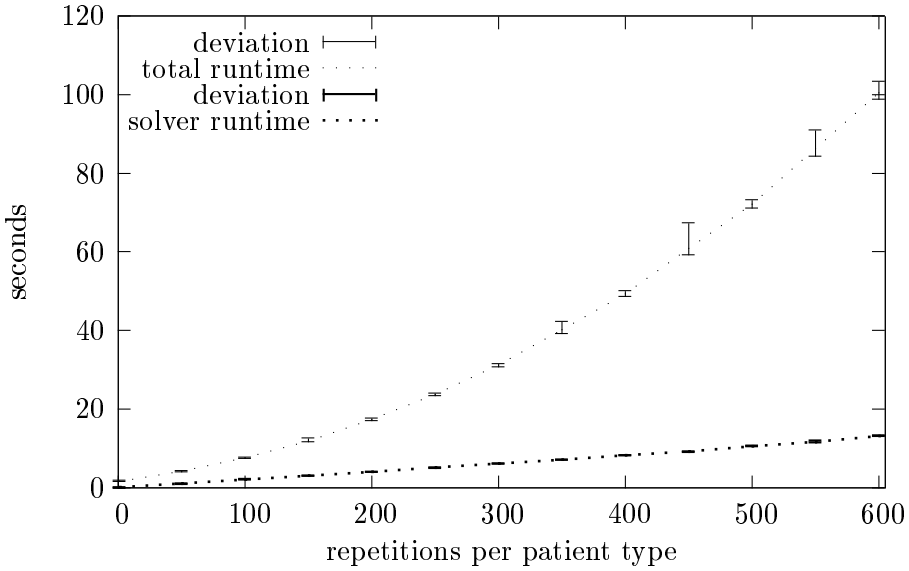


Figure 20: Performance of *pre*CQE for multiple optima

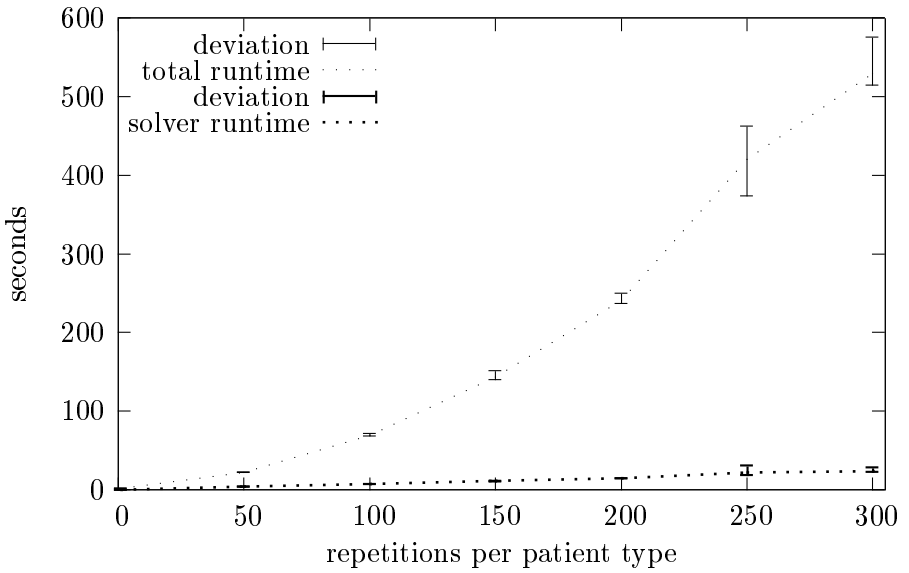


Figure 21: Performance of *pre*CQE for unique optima

In a next testing step we introduced an explicit availability policy; that is, we supplied a set *avail* with two entries for each patient type:

$$avail = \{\texttt{n1\_medA}, \texttt{n1\_medB}, \texttt{n2\_medA}, \texttt{n2\_medB}, \ldots\}$$

In the *pre*CQE implementation, they are first of all evaluated according to *db* (that

is, $eval^*(avail)(db)$ is computed). The resulting formulas are translated as described in Section 17.2 into clauses augmented with an auxiliary propositional variable that are added to the hard constraints in $C$ and into auxiliary constraints with an intermediate weight, as well.
In the simplest case with one repetition per patient type we thus have 120 decision variables again with lowest weight 1. As there are 48 formulas in *avail*, we have 48 auxiliary constraints with weight 121. Finally there are $48 + 48 + 48 = 144$ hard constraints with weight $120 + (48 \cdot 121) + 1 = 5929$. That is, when satisfying all hard constraints, the solution has a weight of at least 853776. We experienced problems with these high weight values, because after 55 repetitions of patient types, we faced an integer overflow: the computed solution suddenly had a negative weight. This was also the case with the PMSAT input format, although the weights of the hard clauses should be ignored and not be included in the optimization process in this case. The results of the test runs can be found in Figure 22 and Table 6 for up to 50 repetitions of the 24 patient types; that is, we tested up to 6000 clauses with lowest weight, 2400 auxiliary clauses and 7200 hard clauses.

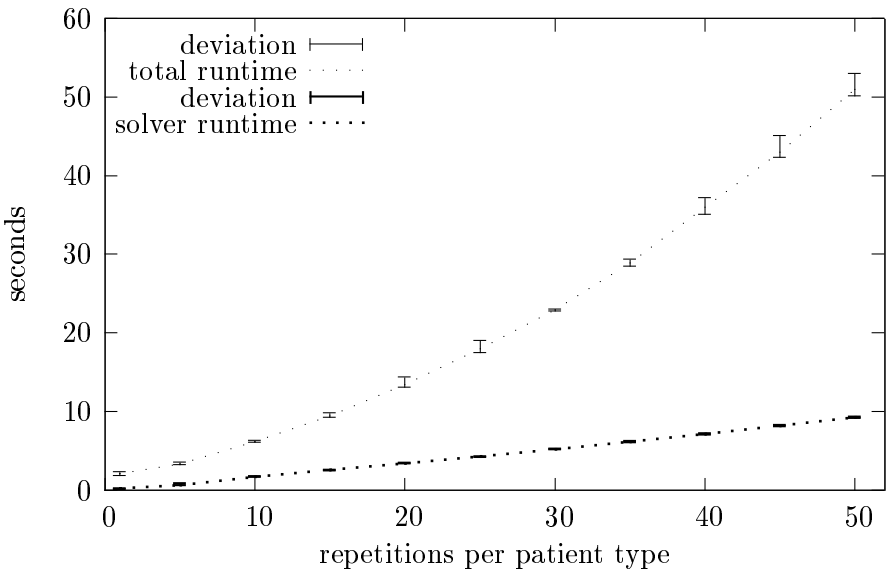


Figure 22: Performance of *pre*CQE with availability policy

We then examined the performance of a reduced set of patient types. We removed the patient types with `medA`-entries, such that only 13 were left: we kept N1, N2, N3, N4, N5, N6, N7, N14, N15, N16, N17, N18, N19 and their corresponding entries in *db*, *prior*, *pot_sec* and *avail*. The results can be found in Figure 23 and Table 7. Quite unexpectedly, we were able to repeat these 13 patient types much more often than the full 24 patient type set. Probably only the search with the full set led to the integer overflow, while for the reduced set this was not the case.

| | total runtime (msec) | | | solver runtime (msec) | | | dec. | clauses | | |
|---|---|---|---|---|---|---|---|---|---|---|
| rep. | min | max | avg. | min | max | avg. | vars. | low | aux. | hard |
| 1 | 1878 | 2386 | 2097 | 181 | 229 | 206 | 120 | 120 | 48 | 144 |
| 5 | 3285 | 3551 | 3334 | 617 | 896 | 660 | 600 | 600 | 240 | 720 |
| 10 | 6091 | 6363 | 6154 | 1716 | 1756 | 1729 | 1200 | 1200 | 480 | 1440 |
| 15 | 9268 | 9837 | 9460 | 2530 | 2604 | 2571 | 1800 | 1800 | 720 | 2160 |
| 20 | 13124 | 14417 | 13408 | 3397 | 3524 | 3431 | 2400 | 2400 | 960 | 2880 |
| 25 | 17522 | 19044 | 17952 | 4240 | 4324 | 4289 | 3000 | 3000 | 1200 | 3600 |
| 30 | 22778 | 23064 | 22937 | 5178 | 5302 | 5232 | 3600 | 3600 | 1440 | 4320 |
| 35 | 28483 | 29429 | 28879 | 6086 | 6279 | 6156 | 4200 | 4200 | 1680 | 5040 |
| 40 | 35083 | 37233 | 35992 | 7081 | 7255 | 7168 | 4800 | 4800 | 1920 | 5760 |
| 45 | 42315 | 45121 | 42962 | 8101 | 8288 | 8209 | 5400 | 5400 | 2160 | 6480 |
| 50 | 50189 | 53000 | 50925 | 9172 | 9383 | 9260 | 6000 | 6000 | 2400 | 7200 |

Table 6: Performance of *pre*CQE with availability policy

| | total runtime (msec) | | | solver runtime (msec) | | | dec. | clauses | | |
|---|---|---|---|---|---|---|---|---|---|---|
| rep. | min | max | avg. | min | max | avg. | vars. | low | aux. | hard |
| 1 | 1744 | 2102 | 1841 | 142 | 162 | 146 | 65 | 65 | 26 | 78 |
| 25 | 8068 | 8308 | 8166 | 2051 | 2118 | 2077 | 1625 | 1625 | 650 | 1950 |
| 50 | 19028 | 20650 | 19423 | 4009 | 4121 | 4076 | 3250 | 3250 | 1300 | 3900 |
| 75 | 35061 | 37300 | 35706 | 5981 | 6266 | 6132 | 4875 | 4875 | 1950 | 5850 |
| 100 | 54295 | 63201 | 57153 | 5712 | 8375 | 8002 | 6500 | 6500 | 2600 | 7800 |
| 150 | 107971 | 117968 | 113187 | 8695 | 12533 | 12017 | 9750 | 9750 | 3900 | 11700 |
| 200 | 187700 | 195847 | 190946 | 11601 | 16924 | 16131 | 13000 | 13000 | 5200 | 15600 |
| 250 | 277757 | 296878 | 289068 | 15119 | 21247 | 20257 | 16250 | 16250 | 6500 | 19500 |
| 300 | 397551 | 425732 | 407416 | 18031 | 26073 | 24910 | 19500 | 19500 | 7800 | 23400 |
| 350 | 537890 | 562223 | 548090 | 22343 | 31778 | 30152 | 22750 | 22750 | 9100 | 27300 |

Table 7: Performance of *pre*CQE without `medA`-entries

| | total runtime (msec) | | | solver runtime (msec) | | | dec. | clauses | | |
|---|---|---|---|---|---|---|---|---|---|---|
| rep. | min | max | avg. | min | max | avg. | vars. | low | aux. | hard |
| 1 | 1941 | 2934 | 2182 | 225 | 337 | 264 | 120 | 120 | 48 | 120 |
| 25 | 18283 | 23445 | 20439 | 4630 | 6530 | 5362 | 3000 | 3000 | 1200 | 3000 |
| 50 | 50486 | 58167 | 52449 | 9651 | 12052 | 9966 | 6000 | 6000 | 2400 | 6000 |
| 75 | 101453 | 105935 | 103266 | 15904 | 16270 | 16115 | 9000 | 9000 | 3600 | 9000 |
| 100 | 164611 | 175434 | 170920 | 18185 | 23374 | 22623 | 12000 | 12000 | 4800 | 12000 |
| 125 | 252737 | 276016 | 260537 | 28020 | 32737 | 31255 | 15000 | 15000 | 6000 | 15000 |
| 150 | 351488 | 380984 | 367471 | 32437 | 40160 | 39087 | 18000 | 18000 | 7200 | 18000 |

Table 8: Performance of *pre*CQE with conjunctive secrets

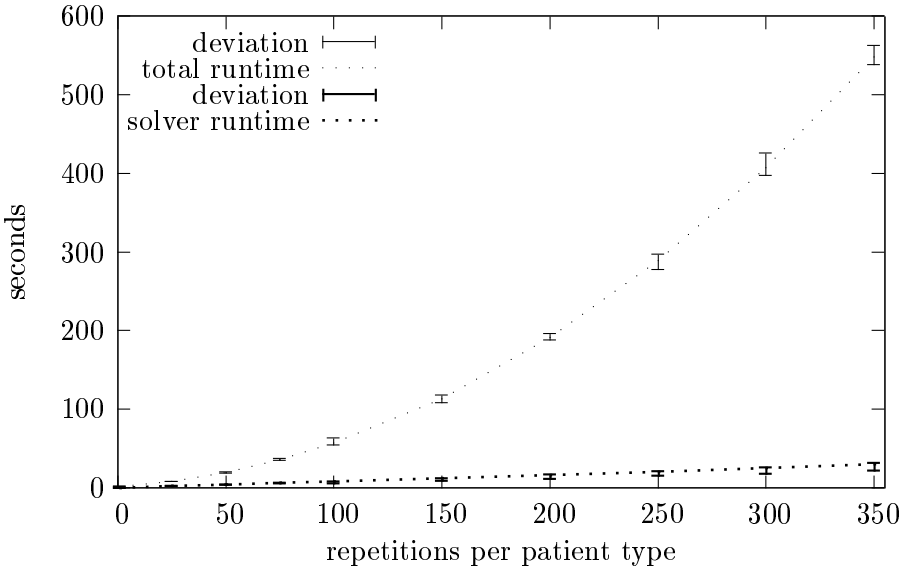


Figure 23: Performance of *pre*CQE without `medA`-entries

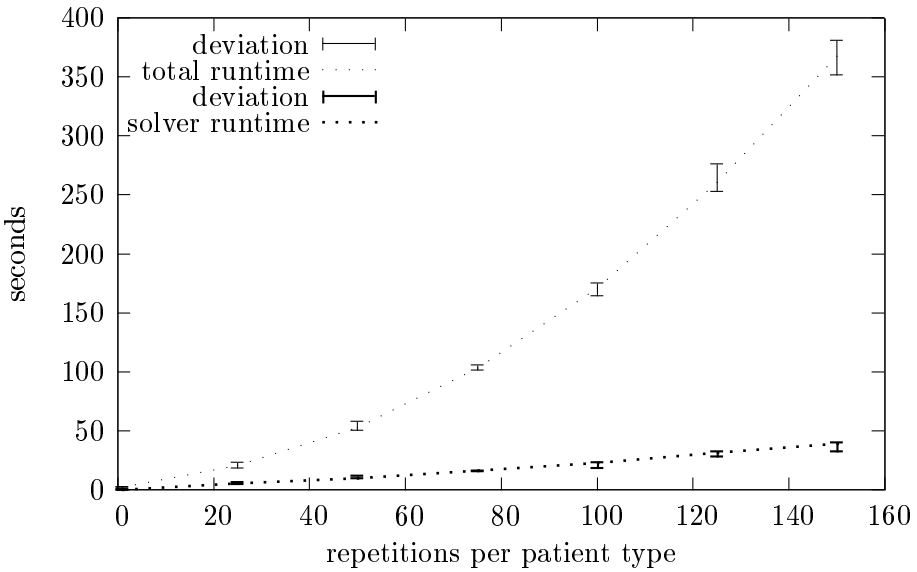


Figure 24: Performance of *pre*CQE with conjunctive secrets

Lastly, we made a test with the full set of 24 patient types but we changed the potential secrets into a conjunctive format:

$$pot\_sec = \{\texttt{n1\_aids} \wedge \texttt{n1\_cancer}, \texttt{n2\_aids} \wedge \texttt{n2\_cancer}, \ldots\}$$

This means that for every patient it is allowed to know if the patient has either aids

or cancer but it is not allowed to know that a patient has both aids and cancer at the same time. This offers a greater set of possible solutions and the SAT solver is forced to make more decision steps. The results are detailed in Figure 23 and Table 7. Yet, as the amount of formulas in *pot_sec* is half of what it was before – only one entry per patient type – the number of hard clauses is reduced: for one repetition we have 120 low level constraints, 48 auxiliary constraints and $48 + 48 + 24 = 120$ hard constraints.

Further tests that expand the setting (for example introducing a third medicine medC and adapting *pot_sec* and *prior* accordingly) were started but not yet thoroughly analyzed.

## 19.2 Cascading Constraints Tests

The Medical Record Tests contained independent subproblems in the sense that for each patient a satisfaction of the constraints could be reached without affecting the entries of other patients. Hence, as a second class of test cases we looked for constraints that are more involved and where the search for a solution requires several splitting and backtracking steps because the clauses share variables and thus are interconnected. Moreover we wanted to test formulas with increasing length. We perceived this to be a lot more challenging task for the SAT solvers.

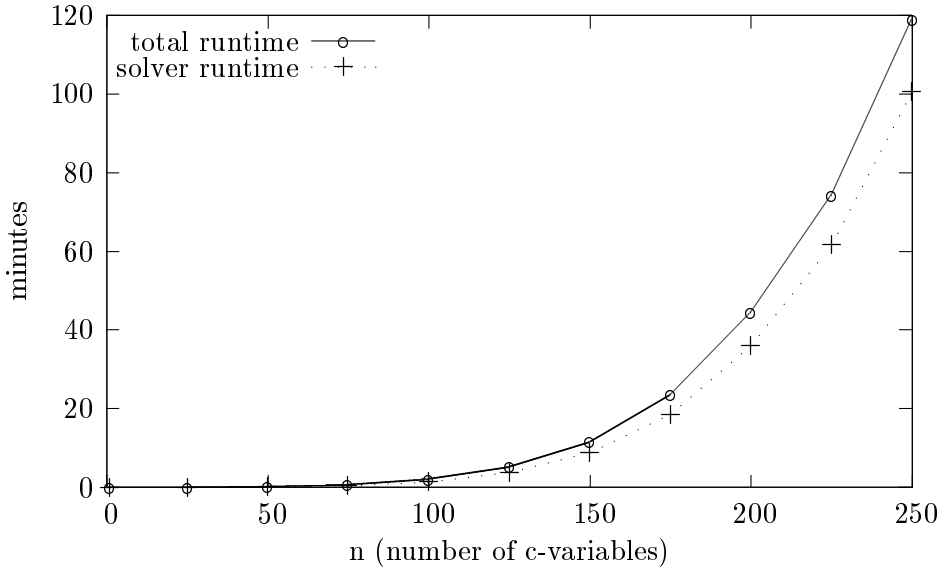


Figure 25: Performance of *pre*CQE with cascading constraints

We adapted one example taken from [Win90] (page 36) which originally was conceived in the setting of deletion of constraints – roughly speaking – in order to make the set of constraints satisfiable after an update. We on the contrary were look-

| $n$ | total runtime (msec.) | solver runtime (msec.) | dec. vars. | soft clauses | hard clauses |
|---|---|---|---|---|---|
| 25 | 4249 | 876 | 625 | 625 | 648 |
| 50 | 11914 | 4922 | 2500 | 2500 | 2548 |
| 75 | 39555 | 22439 | 5625 | 5625 | 5698 |
| 100 | 125470 | 80787 | 10000 | 10000 | 10098 |
| 125 | 316619 | 221291 | 15625 | 15625 | 15748 |
| 150 | 695302 | 532373 | 22500 | 22500 | 22648 |
| 175 | 1418657 | 1112157 | 30625 | 30625 | 30798 |
| 200 | 2663476 | 2162016 | 40000 | 40000 | 40198 |
| 225 | 4456381 | 3694283 | 50625 | 50625 | 50848 |
| 250 | 7144284 | 6038346 | 62500 | 62500 | 62748 |

Table 9: Performance of *pre*CQE with cascading constraints

ing for a satisfiable set of constraints where however the detection of a W-PMSAT model is quite complicated. Consider the following input sets:

$$\begin{aligned} db &= \{\texttt{c1}, \texttt{c2}, \texttt{c3}\} \\ pot\_sec &= \{\texttt{c3}\} \\ prior &= \{\texttt{c3<=>((~v3\_1|~v3\_2|~v3\_3)\& c2)}, \\ &\quad\ \texttt{c2<=>((~v2\_1|~v2\_2|~v2\_3)\& c1)}\} \end{aligned}$$

We see that $Neg(pot\_sec) = \{\neg\texttt{c3}\}$, which is a unit constraint. The first *prior*-formula is violated when the unit constraint is satisfied; it can be satisfied by adding all three variables `v3_1`, `v3_2` and `v3_3` to $db$, or removing `c2` from $db$. In the latter case, the second *prior*-formula is violated; it can be satisfied by adding all three variables `v2_1`, `v2_2` and `v2_3` to $db$, or removing `c1` from $db$. Obviously, the solution where all `c`-variables are removed (that is $db' = \emptyset$), is distortion-minimal with distortion distance 3, while the other two solutions are not – they have distance 4 and 5, respectively. The challenging task for the SAT solver is, that it also has to search on the `v`-variables until the unique solution is found. Note that the first *prior* formula is transformed into the following five clauses in the TPTP conversion step; that is, CNF conversion is done automatically by the TPTP library:

```
~c3|~v1_1|~v1_2|~v1_3 , ~c3|c2 , c3|~c2|v1_1 , c3|~c2|v1_2 , c3|~c2|v1_3
```

Analogously, the second *prior* formula is transformed into:

```
~c2|~v2_1|~v2_2|~v2_3 , ~c2|c1 , c2|~c1|v2_1 , c2|~c1|v2_2 , c2|~c1|v2_3
```

We expanded this example as follows: whenever there are $n$ `c`-variables, in every *prior*-formula there are $n$ distinct `v`-variables. There are $n-1$ such formulas in *prior*; thus in total there are $n^2$ decision variables. The number of clauses after

CNF conversion is $n^2$ as a soft constraint for each variable, 1 for the unit constraint in *pot_sec*, and $(n+2)(n-1) = n^2 + n - 2$ for the formulas in *prior*(that is, we have $(n+2)(n-1) = n^2 + n - 1$ hard constraints).

## Summary of Part V

This part showed how the special propositional case can be interpreted as a SAT solving problem. We proposed several alternative encodings; the W-PMSAT problem was the most appropriate to emulate *pre*CQE. The presented prototype makes use of current SAT solver technology and uses the TPTP library which is standard in automated theorem proving. Two sets of test runs showed that the preprocessing approach is feasible for a large number of database entries.

## References

[ABK00] Marcelo Arenas, Leopoldo E. Bertossi, and Michael Kifer. Applications of annotated predicate calculus to querying inconsistent databases. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *First International Conference on Computational Logic, Proceedings*, volume 1861 of *Lecture Notes in Computer Science*, pages 926–941. Springer, 2000.

[ADB06] Ofer Arieli, Marc Denecker, and Maurice Bruynooghe. Distance-based repairs of databases. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *JELIA Logics in Artificial Intelligence, Proceedings*, volume 4160 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2006.

[ADB07] Ofer Arieli, Marc Denecker, and Maurice Bruynooghe. Distance semantics for database repair. *Annals of Mathematics and Artificial Intelligence*, 50(3–4):389–415, 2007.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AK08] Foto N. Afrati and Phokion G. Kolaitis. Answering aggregate queries in data exchange. In Maurizio Lenzerini and Domenico Lembo, editors, *Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Proceedings*, pages 129–138. ACM, 2008.

[Bau00] Peter Baumgartner. FDPLL - a first order Davis-Putnam-Logeman-Loveland procedure. In David A. McAllester, editor, *17th International Conference on Automated Deduction, Proceedings*, volume 1831 of *Lecture Notes in Computer Science*, pages 200–219, 2000.

[BB01] Joachim Biskup and Piero A. Bonatti. Lying versus refusal for known potential secrets. *IEEE Transactions on Data & Knowledge Engineering*, 38(2):199–222, 2001.

[BB04a] Joachim Biskup and Piero A. Bonatti. Controlled query evaluation for enforcing confidentiality in complete information systems. *International Journal of Information Security*, 3(1):14–27, 2004.

[BB04b] Joachim Biskup and Piero A. Bonatti. Controlled query evaluation for known policies by combining lying and refusal. *Annals of Mathematics and Artificial Intelligence*, 40(1-2):37–62, 2004.

[BB04c] Loreto Bravo and Leopoldo E. Bertossi. Consistent query answering under inclusion dependencies. In Hanan Lutfiyya, Janice Singer, and Darlene A. Stewart, editors, *Conference of the Centre for Advanced Studies on Collaborative research, Proceedings*, pages 202–216. IBM, 2004.

[BB07] Joachim Biskup and Piero A. Bonatti. Controlled query evaluation with open queries for a decidable relational submodel. *Annals of Mathematics and Artificial Intelligence*, 50(1-2):39–77, 2007.

[BBFL08] Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 33(4-5):407–434, 2008.

[BBWW07] Joachim Biskup, Dominique Marc Burgard, Torben Weibert, and Lena Wiese. Inference control in logic databases as a constraint satisfaction problem. In Patrick Drew McDaniel and Shyam K. Gupta, editors, *Third International Conference on Information Systems Security, Proceedings*, volume 4812 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2007.

[BC03] Leopoldo E. Bertossi and Jan Chomicki. Query answering in inconsistent databases. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, *Logics for Emerging Applications of Databases*, pages 43–83. Springer, 2003.

[BCFP03] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security*, 6(1):71–127, 2003.

[BEL08] Joachim Biskup, David W. Embley, and Jan-Hendrik Lochner. Reducing inference control to access control for normalized database schemas. *Information Processing Letters*, 106(1):8–12, 2008.

[BEST98] François Bry, Norbert Eisinger, Heribert Schütz, and Sunna Torge. SIC: Satisfiability checking for integrity constraints. In *6. Workshop on Deductive Databases and Logic Programming, Proceedings*, volume 22 of *GMD Report*, pages 25–36, 1998.

[BFJ00] Alexander Brodsky, Csilla Farkas, and Sushil Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE Transactions on Knowledge & Data Engineering*, 12(6):900–919, 2000.

[BGG01] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer, 2001.

[BH80] W. W. Bledsoe and Larry M. Hines. Variable elimination and chaining in a resolution-based prover for inequalities. In Wolfgang Bibel and Robert A. Kowalski, editors, *5th Conference on Automated Deduction, Proceedings*, volume 87 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 1980.

[Bis00] Joachim Biskup. For unknown secrecies refusal is better than lying. *IEEE Transactions on Data & Knowledge Engineering*, 33(1):1–23, 2000.

[BKS95] Piero A. Bonatti, Sarit Kraus, and V. S. Subrahmanian. Foundations of secure deductive databases. *IEEE Transactions on Data & Knowledge Engineering*, 7(3):406–422, 1995.

[BL07] Joachim Biskup and Jan-Hendrik Lochner. Enforcing confidentiality in relational databases by reducing inference control to access control. In Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta, editors, *10th International Conference on Information Security, Proceedings*, volume 4779 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2007.

[Bla96] Simon Blackburn. *The Oxford Dictionary of Philosophy.* Oxford University Press, 1996. Entries on "Inference" and "Logic".

[BT98] François Bry and Sunna Torge. A deduction method complete for refutation and finite satisfiability. In Jürgen Dix, Luis Fariñas del Cerro, and Ulrich Furbach, editors, *JELIA Logics in Artificial Intelligence, Proceedings*, volume 1489 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998.

[BT05] Peter Baumgartner and Cesare Tinelli. The model evolution calculus with equality. In Robert Nieuwenhuis, editor, *20th International Conference on Automated Deduction, Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 392–408. Springer, 2005.

[BT08] Peter Baumgartner and Cesare Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172(4-5):591–632, 2008.

[BW04] Joachim Biskup and Torben Weibert. Refusal in incomplete databases. In Csilla Farkas and Pierangela Samarati, editors, *18th Annual IFIP WG 11.3 Conference on Data and Applications Security, Proceedings*, pages 143–157. Kluwer, 2004.

[BW06] Joachim Biskup and Lena Wiese. On finding an inference-proof complete database for controlled query evaluation. In Ernesto Damiani and Peng Liu, editors, *20th Annual IFIP WG 11.3 Conference on Data*

*and Applications Security, Proceedings*, volume 4127 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2006.

[BW07] Joachim Biskup and Torben Weibert. Confidentiality policies for controlled query evaluation. In Steve Barker and Gail-Joon Ahn, editors, *21st Annual IFIP WG 11.3 Conference on Data and Applications Security, Proceedings*, Lecture Notes in Computer Science, pages 1–13. Springer, 2007.

[BW08a] Joachim Biskup and Torben Weibert. Keeping secrets in incomplete databases. *Internation Journal of Information Security*, 7(3):199–217, 2008.

[BW08b] Joachim Biskup and Lena Wiese. Preprocessing for controlled query evaluation with availability policy. *Journal of Computer Security*, 16(4):477–494, 2008.

[CC94] Laurence Cholvy and Frédéric Cuppens. Providing consistent views in a polyinstantiated database. In Joachim Biskup, Matthew Morgenstern, and Carl E. Landwehr, editors, *Database Security, VIII: Status and Prospects, Proceedings of the IFIP WG11.3 Working Conference on Database Security*, volume A-60 of *IFIP Transactions*, pages 277–296. North-Holland, 1994.

[CC06] Yu Chen and Wesley W. Chu. Database security protection via inference detection. In Sharad Mehrotra, Daniel Dajun Zeng, Hsinchun Chen, Bhavani M. Thuraisingham, and Fei-Yue Wang, editors, *IEEE International Conference on Intelligence and Security Informatics, Proceedings*, volume 3975 of *Lecture Notes in Computer Science*, pages 452–458. Springer, 2006.

[CCGL02] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. In Anne Banks Pidduck, John Mylopoulos, Carson C. Woo, and M. Tamer Özsu, editors, *14th International Conference on Advanced Information Systems Engineering, Proceedings*, volume 2348 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2002.

[CG01] Frédéric Cuppens and Alban Gabillon. Cover story management. *Data & Knowledge Engineering*, 37(2):177–201, 2001.

[CL73] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[CLR03] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Twenty-Second ACM SIGACT-SIGMOD-SIGART Sym-*

*posium on Principles of Database Systems, Proceedings*, pages 260–271. ACM, 2003.

[CM00] LiWu Chang and Ira S. Moskowitz. An integrated framework for database privacy protection. In Bhavani M. Thuraisingham, Reind P. van de Riet, Klaus R. Dittrich, and Zahir Tari, editors, *Fourteenth Annual Working Conference on Database Security, Proceedings*, volume 201 of *IFIP Conference Proceedings*, pages 161–172. Kluwer, 2000.

[CM05] Jan Chomicki and Jerzy Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In Leopoldo E. Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 119–150. Springer, 2005.

[CN09] Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity.* Cambridge University Press, 2009. To be published by the Perspectives in Logic series of the Association for Symbolic Logic.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[CS03] Koen Claessen and Niklas Sörensson. New techniques that improve mace-style finite model finding. In *CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications, Proceedings*, 2003.

[CW94] Seng-Cho Timothy Chou and Marianne Winslett. A model-based belief revision system. *Journal of Automated Reasoning*, 12(2):157–208, 1994.

[DdVLS99] Steven Dawson, Sabrina De Capitani di Vimercati, Patrick Lincoln, and Pierangela Samarati. Minimal data upgrading to prevent inference and association. In *Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Proceedings*, pages 114–125. ACM Press, 1999.

[DdVLS02] Steven Dawson, Sabrina De Capitani di Vimercati, Patrick Lincoln, and Pierangela Samarati. Maximizing sharing of protected information. *Journal of Computer and System Sciences*, 64(3):496–541, 2002.

[Dem92] Robert Demolombe. Syntactical characterization of a subset of domain-independent formulas. *Journal of the ACM*, 39(1):71–94, 1992.

[Den82] Dorothy E. Denning. *Cryptography and Data Security.* Addison-Wesley, 1982.

[Den87] Dorothy E. Denning. Lessons learned from modeling a secure multilevel relational database system. In Carl E. Landwehr, editor, *Database Security: Status and Prospects, Results of the IFIP WG 11.3 Initial Meeting*, pages 35–43. North-Holland, 1987.

[DF02] Josep Domingo-Ferrer, editor. *Inference Control in Statistical Databases, From Theory to Practice*, volume 2316 of *Lecture Notes in Computer Science*. Springer, 2002.

[DFF06] Josep Domingo-Ferrer and Luisa Franconi, editors. *Privacy in Statistical Databases, CENEX-SDC Project International Conference, Proceedings*, volume 4302 of *Lecture Notes in Computer Science*. Springer, 2006.

[DFT04] Josep Domingo-Ferrer and Vicenç Torra, editors. *Privacy in Statistical Databases: CASC Project International Workshop, Proceedings*, volume 3050 of *Lecture Notes in Computer Science*. Springer, 2004.

[DH94] Harry S. Delugach and Thomas H. Hinke. Using conceptual graphs to represent database inference security analysis. *Journal of Computing and Information Technology*, 2(4):291–307, 1994.

[DH96] Harry S. Delugach and Thomas H. Hinke. Wizard: A database inference analysis and detection system. *IEEE Transactions on Data & Knowledge Engineering*, 8(1):56–66, 1996.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[DS83] Dorothy E. Denning and Jan Schlörer. Inference controls for statistical databases. *IEEE Computer*, 16(7):69–82, 1983.

[DS04] James P. Delgrande and Torsten Schaub. Two approaches to merging knowledge bases. In José Júlio Alferes and João Alexandre Leite, editors, *9th European Conference on Logics in Artificial Intelligence, Proceedings*, volume 3229 of *Lecture Notes in Computer Science*, pages 426–438. Springer, 2004.

[DS07] James P. Delgrande and Torsten Schaub. A consistency-based framework for merging knowledge bases. *Journal of Applied Logic*, 5(3):459–477, 2007.

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *8th International Conference on Theory and Applications of Satisfiability Testing, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[EFGL08] Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. Repair localization for query answering from inconsistent databases. *ACM Transactions on Database System*, 33(2):10:1–10:51, 2008.

[FBJ06] Csilla Farkas, Alexander Brodsky, and Sushil Jajodia. Unauthorized inferences in semistructured databases. *Information Sciences*, 176(22):3269–3299, 2006.

[Fit96] Melvin Fitting. *First-order Logic and Automated Theorem Proving.* Springer, 1996. Second Edition.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[FTE01] Csilla Farkas, Tyrone S. Toland, and Caroline M. Eastman. The inference problem and updates in relational databases. In *Fifteenth Annual Working Conference on Database and Application Security, Proceedings*, volume 215 of *IFIP Conference Proceedings*, pages 181–194. Kluwer, 2001.

[GA07] Igor Gammer and Eyal Amir. Solving satisfiability in ground logic with equality by efficient conversion to propositional logic. In Ian Miguel and Wheeler Ruml, editors, *7th International Symposium on Abstraction, Reformulation, and Approximation, Proceedings*, volume 4612 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2007.

[GH08] Bernardo Cuenca Grau and Ian Horrocks. Privacy-preserving query answering in logic-based information systems. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *18th European Conference on Artificial Intelligence, Patras, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 40–44. IOS Press, 2008.

[GS02] Martin Grohe and Luc Segoufin. On first-order topological queries. *ACM Transactions on Computational Logic*, 3(3):336–358, 2002.

[GT91] Allen Van Gelder and Rodney W. Topor. Safety and translation of relational calculus queries. In *ACM Transactions on Database Systems*, volume 16, pages 235–278. ACM, 1991.

[HLO08] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSAT: An efficient Weighted Max-SAT Solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[HRCS02] John N. Hooker, G. Rago, V. Chandru, and A. Shrivastava. Partial instantiation methods for inference in first-order logic. *Journal of Automated Reasoning*, 28(5):371–396, 2002.

[HS91] Richard Hull and Jianwen Su. On the expressive power of database queries with intermediate types. *Journal of Computer and System Sciences*, 43(1):219–267, 1991.

[HS92] Klaus-Uwe Höffgen and Hans-Ulrich Simon. Robust trainability of single neurons. In *Fifth Annual ACM Conference on Computational Learning Theory, Proceedings*, pages 428–439, 1992.

[HS94] Richard Hull and Jianwen Su. Domain independence and the relational calculus. *Acta Informatica*, 31(6):513–524, 1994.

[HS96] John Hale and Sujeet Shenoi. Analyzing FD inference in relational databases. *IEEE Transactions on Data & Knowledge Engineering*, 18(2):167–183, 1996.

[HTS94] John Hale, Jody Threet, and Sujeet Shenoi. A practical formalism for imprecise inference control. In Joachim Biskup, Matthew Morgenstern, and Carl E. Landwehr, editors, *Database Security, VIII: Status and Prospects, Proceedings*, volume A-60 of *IFIP Transactions*, pages 139–156. North-Holland, 1994.

[JM95a] Sushil Jajodia and Catherine Meadows. Inference problems in multilevel secure databasemanagement systems. *Information Security - An Integrated Collection of Essays*, pages 570–584, 1995.

[JM95b] Sushil Jajodia and Catherine Meadows. Solutions to the polyinstantiation problem. *Information Security - An Integrated Collection of Essays*, pages 493–530, 1995.

[Jon97] Peter Jonsson. Tight lower bounds on the approximability of some NPO PB-complete problems. *Linköping Electronic Articles in Computer and Information Science*, 2(4), 1997. `http://www.ep.liu.se/ea/cis/1997/004/`.

[JS91] Sushil Jajodia and Ravi S. Sandhu. Toward a multilevel secure relational data model. In James Clifford and Roger King, editors, *ACM SIGMOD International Conference on Management of Data, Proceedings*, pages 50–59. ACM Press, 1991.

[Kan94] Viggo Kann. Polynomially bounded minimization problems that are hard to approximate. *Nordic Journal of Computing*, 1(3):317–331, 1994.

[LDS+90] Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley. The SeaView security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, 1990.

[LHdG08] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.

[LLV07] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. $t$-closeness: Privacy beyond $k$-anonymity and $l$-diversity. In *23rd International Conference on Data Engineering, Proceedings*, pages 106–115. IEEE Computer Society, 2007.

[LZWJ02] Yingjiu Li, Sencun Zhu, Lingyu Wang, and Sushil Jajodia. A privacy-enhanced microaggregation method. In Thomas Eiter and Klaus-Dieter Schewe, editors, *Second International Symposium on Foundations of Information and Knowledge Systems, Proceedings*, volume 2284 of *Lecture Notes in Computer Science*, pages 148–159. Springer, 2002.

[MGKV06] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. $l$-diversity: Privacy beyond $k$-anonymity. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *22nd International Conference on Data Engineering, Proceedings*, pages 24–36. IEEE Computer Society, 2006.

[MS07] Gerome Miklau and Dan Suciu. A formal analysis of information disclosure in data exchange. *Journal of Computer and System Sciences*, 73(3):507–534, 2007.

[MSS88] Subhasish Mazumdar, David W. Stemple, and Tim Sheard. Resolving the tension between integrity and security using a theorem prover, proceedings. In Haran Boral and Per-Åke Larson, editors, *ACM SIGMOD International Conference on Management of Data*, pages 233–242. ACM Press, 1988.

[Nic82] Jean-Marie Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.

[Per94] Günther Pernul. Database security. *Advances in Computers*, 38:1–72, 1994.

[Qia94] Xiaolei Qian. Inference channel-free integrity constraints in mulitlevel relational databases. In *IEEE Symposium on Security and Privacy, Proceedings*, pages 158–173, 1994.

[QL92] Xiaolei Qian and Teresa F. Lunt. Tuple-level vs element-level classification. In *Database Security, VI: Status and Prospects. Results of the IFIP WG 11.3 Workshop on Database Security*, volume A-21 of *IFIP Transactions*, pages 301–316. North-Holland, 1992.

[RMSR04] Shariq Rizvi, Alberto O. Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In

Gerhard Weikum, Arnd Christian König, and Stefan Deßloch, editors, *ACM SIGMOD International Conference on Management of Data, Proceedings*, pages 551–562. ACM, 2004.

[Ros06] Riccardo Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In Stijn Vansummeren, editor, *Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Proceedings*, pages 356–365. ACM, 2006.

[SdJvdR83] George L. Sicherman, Wiebren de Jonge, and Reind P. van de Riet. Answering queries without revealing secrets. *ACM Transactions on Database Systems*, 8(1):41–59, 1983.

[SF04] Andrei Stoica and Csilla Farkas. Ontology guided XML security engine. *Journal of Intelligent Information Systems*, 23(3):209–223, 2004.

[SJ90] R. Sandhu and S. Jajodia. Restricted polyinstantiation or how to close signaling channels without duplicity. In *Third RADC Workshop on Multilevel Database Security, Proceedings*, 1990.

[SJ92] Ravi S. Sandhu and Sushil Jajodia. Polyinstantation for cover stories. In Yves Deswarte, Gérard Eizenberg, and Jean-Jacques Quisquater, editors, *Second European Symposium on Research in Computer Security, Proceedings*, volume 648 of *Lecture Notes in Computer Science*, pages 307–328. Springer, 1992.

[SÖ91] Tzong-An Su and Gultekin Özsoyoglu. Controlling FD and MVD inferences in multilevel relational database systems. *IEEE Transactions on Data & Knowledge Engineering*, 3(4):474–485, 1991.

[SP04] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. Niver: Non increasing variable elimination resolution for preprocessing SAT instances. In *7th International Conference on Theory and Applications of Satisfiability Testing, Proceedings*, 2004.

[ST99] Alexei P. Stolboushkin and Michael A. Taitslin. Finite queries do not have effective syntax. *Information and Computation*, 153(1):99–116, 1999.

[Sta03] Jessica Staddon. Dynamic inference control. In Mohammed Javeed Zaki and Charu C. Aggarwal, editors, *8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, Proceedings*, pages 94–100, 2003.

[Sti94] Mark E. Stickel. Elimination of inference channels by optimal upgrading. In *IEEE Symposium on Security and Privacy, Proceedings*, pages 168–174, 1994.

[Sut07] Geoff Sutcliffe. TPTP, TSTP, CASC, etc. In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Computer Science - Theory and Applications, Proceedings*, volume 4649 of *Lecture Notes in Computer Science*, pages 6–22. Springer, 2007.

[SW92] Kenneth Smith and Marianne Winslett. Entity modeling in the MLS relational model. In Li-Yan Yuan, editor, *18th International Conference on Very Large Data Bases, Proceedings*, pages 199–210. Morgan Kaufmann, 1992.

[Swe97] Latanya Sweeney. Datafly: A system for providing anonymity in medical data. In *Database Security XI: Status and Prospects, Proceedings*, volume 113 of *IFIP Conference Proceedings*, pages 356–381. Chapman & Hall, 1997.

[Swe02] Latanya Sweeney. $k$-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[Tad08] Cornelia Tadros. Erzeugen einer inferenzsicheren und optimal verfügbaren Datenbank mit gewichtetem Max-SAT. Technische Universität Dortmund, Fakultät für Informatik, Lehrstuhl 6, October 2008.

[TFE05] Tyrone S. Toland, Csilla Farkas, and Caroline M. Eastman. Dynamic disclosure monitor ($D^2$Mon): An improved query processing solution. In Willem Jonker and Milan Petkovic, editors, *Second VLDB Workshop on Secure Data Management, Proceedings*, volume 3674 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 2005.

[Wei08] Torben Weibert. *A Framework for Inference Control in Incomplete Logic Databases.* PhD thesis, Technische Universität Dortmund, 2008.

[Wil97] Mary-Anne Williams. Anytime belief revision. In *Fifteenth International Joint Conference on Artificial Intelligence, Proceedings*, pages 74–81. Morgan Kaufmann, 1997.

[Win90] Marianne Winslett. *Updating Logical Databases.* Cambridge University Press, 1990.

[WLWJ03] Lingyu Wang, Yingjiu Li, Duminda Wijesekera, and Sushil Jajodia. Precisely answering multi-dimensional range queries without privacy breaches. In Einar Snekkenes and Dieter Gollmann, editors, *8th European Symposium on Research in Computer Security, Proceedings*, volume 2808 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2003.

[WS04] David P. Woodruff and Jessica Staddon. Private inference control. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel,

editors, *ACM Conference on Computer and Communications Security, Proceedings*, pages 188–197, 2004.

[YL98a] Raymond W. Yip and Karl N. Levitt. Data level inference detection in database systems. In *11th IEEE Computer Security Foundations Workshop, Proceedings*, pages 179–189, 1998.

[YL98b] Raymond W. Yip and Karl N. Levitt. The design and implementation of a data level database inference detection system. In *Database Security XII: Status and Prospects, IFIP TC11 WG 11.3 Twelfth International Working Conference on Database Security, Proceedings*, volume 142 of *IFIP Conference Proceedings*, pages 253–266. Kluwer, 1998.

[YL04] Xiaochun Yang and Chen Li. Secure XML publishing without information leakage in the presence of data inference. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *30th International Conference on Very Large Data Bases, Proceedings*, pages 96–107. Morgan Kaufmann, 2004.

[Zha05] Lintao Zhang. On subsumption removal and on-the-fly CNF simplification. In Fahiem Bacchus and Toby Walsh, editors, *8th International Conference on Theory and Applications of Satisfiability Testing, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 482–489. Springer, 2005.

[ZJB08] Lei Zhang, Sushil Jajodia, and Alexander Brodsky. Simulatable binding: Beyond simulatable auditing. In Willem Jonker and Milan Petkovic, editors, *5th VLDB Workshop on Secure Data Management*, volume 5159 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2008.

## List of Figures

## List of Tables

## List of Definitions

## List of Theorems, Lemmas, Propositions, Corollaries and Remarks

## Full Contents

## Index

## Danksagung

Mein besonderer Dank gilt Joachim Biskup für seine fachliche und menschliche Begleitung und Anleitung während des gesamten Prozesses der Promotion.
Danken möchte ich auch Gabriele Kern-Isberner als Zweitgutachterin und Heiko Krumm und Hubert Wagner für ihre Bereitschaft als Mitglieder der Prüfungskommission das Promotionsverfahren zum Abschluss zu bringen; ich lege sehr großen Wert auf ihr fachliches Urteil.
Cornelia Tadros bin ich zu großem Dank verpflichtet für ihre herausragenden Leistungen bei Implementierung und Test des Prototypen.
In den ersten beiden Jahren der Promotionszeit war ich Stipendiatin im DFG-Graduiertenkolleg "Mathematische und ingenieurwissenschaftliche Methoden für sichere Datenübertragung und Informationsvermittlung"; dieses Stipendium hat mir einen gelungenen Einstieg in das wissenschaftliche Arbeiten ermöglicht.
Schließlich ein dickes DANKE! an all die Menschen, die im Großen und im Kleinen zum Gelingen dieser Doktorarbeit beigetragen haben; ohne euch wäre diese Arbeit nicht so weit gediehen!