On Finding an Inference-Proof Complete Database for Controlled Query Evaluation

Joachim Biskup and Lena Wiese

Universität Dortmund, 44221 Dortmund, Germany
{biskup,wiese}@ls6.cs.uni-dortmund.de
http://ls6-www.cs.uni-dortmund.de/issi/

Abstract. Controlled Query Evaluation (CQE) offers a logical framework to prevent a user of a database from inadvertently gaining knowledge he is not allowed to know. By modeling the user's a priori knowledge in an appropriate way, a CQE system can control not only plain access to database entries but also inferences made by the user. A dynamic CQE system that enforces inference control at runtime has already been investigated. In this article, we pursue a static approach that constructs an inference-proof database in a preprocessing step. The inference-proof database can respond to any query without enabling the user to infer confidential information. We illustrate the semantics of the system by a comprehensive example and state the essential requirements for an inference-proof and highly available database. We present an algorithm that accomplishes the preprocessing by combining SAT solving and "Branch and Bound".

Keywords: Controlled Query Evaluation, inference control, lying, confidentiality of data, complete database systems, propositional logic, SAT solving, Branch and Bound

1 Introduction and Related Work

Controlled query evaluation (cf. [1–3]) aims at preserving the confidentiality of some secret information in a sequence of queries to a database. Not only plain access to certain database entries is denied but also information that can be gained by logical reasoning is taken into consideration. This is what is usually called *inference control*. There are a lot of different approaches addressing inference control for example for statistical databases [9], distributed databases [5], relational databases with fuzzy values [15] and for XML documents [19]. In [11] the authors give a comprehensive overview of existing inference control techniques and state some of the fundamental problems in this area. Wang et al. [18] name two typical protection mechanisms (in their case for online analytical processing (OLAP) systems): *restriction* (deleting some values in the query result) and *perturbation* (changing some values in the query result). In general, any method for avoiding inferences has an effect on the accuracy of the returned

answers. Hence, there is a trade-off between *confidentiality* of secret information and *availability* of information; in order to protect the secret information, some (even non-secret) information may be modified. *Reliability* may also be reduced by protection mechanisms; the user may be unsure of whether he got a correct or a modified answer.
The above mentioned approaches are typically based on specialized data structures (relational data model, XML documents); Controlled Query Evaluation (CQE) however offers a flexible framework to execute inference control based on an arbitrary logic. In this paper, we construct an inference-proof database in the CQE framework considering the original database, a user's a priori knowledge and a set of secrets. In Section 2 we introduce the CQE framework and state the prerequisites assumed in this paper. In Section 3 we formalize the notion of an inference-proof and highly available database. Section 4 shows a transformation of our problem to SAT solving and presents an algorithm that computes an inference-proof database. A comprehensive example illustrates our approach in Subsections 2.1 and 4.3.

2 Controlled Query Evaluation

Basically, a system model for Controlled Query Evaluation consists of:

1. a database that contains some freely accessible information and some secret information
2. a single user (or a group of collaborating users, respectively) having a certain amount of information as a priori knowledge of the database and the world in general; the case that several different users independently query the database is not considered as the database cannot distinguish whether a group of users collaborates or not

The user sends queries to the database and the database sends corresponding answers to the user. To prevent the user from inferring confidential information from the answers and his assumed a priori knowledge, appropriate restriction or perturbation is done by the CQE system on the database side. In CQE on the one hand *refusal* is used as a means of restriction: to a critical query the database refuses to answer (i.e., just returns `mum`). On the other hand, *lying* is employed as a means of perturbation: the database returns a false value or declares the query answer as undefined although a value exists in the database. In this way, the CQE approach automates the enforcement of confidentiality: wanting to restrict access to some secret information, a database administrator just specifies which information has to be kept secret; then, the CQE system computes the inference-proof answers. No human being ever has to consider inferences potentially made by the user, because the CQE system computes these inferences and employs the above mentioned protection mechanisms such that only "safe" answers are returned to the user. However, the database should be as cooperative as possible: only a minimum of answers should be distorted to ensure the confidentiality of the secret information. CQE can be varied based on several different parameters:

– complete information systems are considered with a model-theoretic approach (see [2]) while incomplete information systems are treated with a proof-theoretic approach (see [4])
– the secret information can be represented by two distinct types of "confidentiality policies": either secrecies (no truth value may be known by the user – be it *true* or *false*) or potential secrets (the user may know that a secret is *false* or possibly undefined, but the user may not know that a secret is *true*)
– the user may or may not know what kind of information is to be kept secret; therefore the "user awareness" can be divided into the known policy and the unknown policy case
– as the protection mechanism (also called "enforcement method") either lying or refusal or a combination of both can be employed (see [3])

In this paper we focus on a *complete* information system in *propositional logic* with a security policy of *potential secrets* and a *known* policy. Additionally, we only use *lying* as a protection mechanism. Thus, in this paper a CQE system is based on the following:

– a finite alphabet $\mathcal{P}$ of propositional variables; formulas can be built from the variables with the connectives $\neg$, $\vee$ and $\wedge$;[1] formulas contain "positive literals" (variables) and "negative literals" (negations of variables)
– a database $db \subset \mathcal{P}$ that represents an interpretation I of the propositional variables: for each $A \in \mathcal{P}$, if $A \in db$, then I assigns A the value *true* (written as $I(A) = 1$), else I assigns A the value *false* (written as $I(A) = 0$); this means that we have a complete database: to each query the database returns either *true* or *false*
– the user's queries to the database as formulas Φ over $\mathcal{P}$
– a security policy *pot_sec* as a set of formulas over $\mathcal{P}$ of "potential secrets"; these potential secrets represent the secret information; the semantics is that for each formula $\Psi \in$ *pot_sec*, if Ψ evaluates to *true* according to *db* then the user may not know this, but the user may believe that the formula evaluates to *false* (that is, the negation of Ψ evaluates to *true* according to *db*).
– the user's a priori knowledge as a set of formulas over $\mathcal{P}$ called *prior*; *prior* may contain general knowledge (like implications over $\mathcal{P}$) or knowledge of *db* (like semantic constraints)

There are some prerequisites our system has to fulfill. In this paper, we presume that:

(a) [**consistent knowledge**] *prior* is consistent and the user cannot be made believe inconsistent information at any time
(b) [**monotone knowledge**] the user cannot be "brainwashed" and forced to forget some part of his knowledge
(c) [**single source of information**] the database *db* is the user's only source of information (besides *prior*)
(d) [**unknown secrets**] for all $\Psi \in$ *pot_sec*: $prior \not\models \Psi$

[1] Two consecutive negations cancel each other out: $\neg\neg A \equiv A$

(e) **[implicit closure under disjunction]** the user may not know (a priori) that the disjunction of the potential secrets is true:

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

We require (d) because if the user already knows a potential secret, we obviously cannot protect the secret anymore. Moreover, in the case of lying as the only protection mechanism, we even have to be more strict: requirement (e) is necessary, because otherwise the system could run into a situation where even a lie reveals a secret. To illustrate this, assume $pot\_sec = \{\alpha, \beta\}$ (for formulas α and β that are both *true* according to *db*) and $prior = \{\alpha \vee \beta\}$; to the query $\Phi = \alpha$ the CQE system would return the lie $\neg\alpha$, but this would enable the user to conclude that β were true (and he is not allowed to know this); thus, we require *prior* to fulfill Equation (1). This line of reasoning also demands that the CQE system lie to every query entailing the disjunction of some potential secrets. See [1, 3] for more information.

2.1 An Example System

The following example shall clarify the system design. Let us imagine that we have a database with Alice's medical records. The curious user Mallory wants to find out whether she is incapable of working or has financial problems. We use the alphabet

$$\mathcal{P} = \{\texttt{cancer}, \texttt{brokenArm}, \texttt{brokenLeg}, \texttt{highCosts}, \texttt{lowWorkCpcty}\}.$$

Poor Alice is badly ill and her medical records (as a set of literals) look like this:

$$record = \{\texttt{cancer}, \texttt{brokenArm}, \texttt{brokenLeg}\}.$$

It is generally known that cancer leads to high medical costs and low work capacity and that a broken arm leads to low work capacity and a broken leg to high medical costs. Expressing these implications as formulas, we have the general knowledge

$$\begin{aligned} genknowl = \{ \quad & \neg\texttt{cancer} \vee \texttt{highCosts}, \neg\texttt{cancer} \vee \texttt{lowWorkCpcty}, \\ & \neg\texttt{brokenArm} \vee \texttt{lowWorkCpcty}, \neg\texttt{brokenLeg} \vee \texttt{highCosts}\}. \end{aligned}$$

The database *db* contains the medical record *record* and is compliant with the general knowledge *genknowl*; thus, in this example we have

$$db = \mathcal{P}.$$

Mallory just knows what is also generally known:

$$prior = genknowl.$$

Now we can specify the potential secrets. As a first example we have just one formula consisting of one literal:

$$pot\_sec_1 = \{\texttt{lowWorkCpcty}\}.$$

To the query $\Phi = \texttt{cancer}$ the database should now return the lie *false* (as otherwise Mallory would conclude from his a priori knowledge that Alice has low work capacity). The same applies to the queries `brokenArm` and `lowWorkCpcty`. However, to the queries `brokenLeg` and `highCosts` the database should return the correct answer *true*.
A conjunction of two literals means that Mallory may know one of the literals but may not know both at the same time. As an example, consider the secret

$$pot\_sec_2 = \{\texttt{highCosts} \wedge \texttt{lowWorkCpcty}\}.$$

To one of the queries $\Phi_1 = \texttt{brokenArm}$ and $\Phi_2 = \texttt{brokenLeg}$ the database can return the correct answer *true*; however, to the other query the database has to return the lie *false* (otherwise both high medical costs and low work capacity can be concluded).
The meaning of a disjunction of two literals implies that Mallory may know neither of the literals. Consider

$$pot\_sec_3 = \{\texttt{highCosts} \vee \texttt{lowWorkCpcty}\}.$$

To every possible atomic query $\Phi \in \mathcal{P}$ the database has to return the lie *false* (as otherwise either high medical costs or low work capacity or both can be concluded).
The intended semantics of *pot_sec* can inductively be extended to non-singleton sets of arbitrary formulas.

As can be seen from the above examples, confidentiality of secret information is considered more important than a correct and reliable answer. Secret information has to be kept secret even at the risk of returning inaccurate information.

3 Constructing an Inference-Proof Database

Given a database *db*, a security policy *pot_sec* in the form of potential secrets, and the user's a priori knowledge *prior* as described in the previous section, we now want to construct a database db' that is inference-proof with respect to every possible sequence of queries the user may come up with. We demand the following for db' to be fulfilled:

i. [**inference-proof**] db' does not satisfy any of the potential secrets: $db' \not\models \Psi$ for every $\Psi \in pot\_sec$
ii. [**complete**] db' is complete (as is *db*)
iii. [**highly available**] db' contains as few lies as possible; we want to remove, add or change only a minimum of entries (with respect to the original database *db*): db' shall contain a lie only if otherwise a potential secret would be endangered

iv. [**consistent**] db' is consistent in itself and also consistent with *prior* as the user's a priori knowledge is fixed and we cannot make the user believe inconsistent information

As for the inference-proofness and completeness features (i. and ii.), we want db' to represent an interpretation I' that assigns a value to every propositional variable in $\mathcal{P}$. Hence the database db' must return an answer (*true* or *false*) to every query – also to a query containing a potential secret. As we consider here the *known policy* case, the user knows that he gets the answer *false* when querying a potential secret (because he is not allowed to know that a potential secret is true). So concluding from completeness and the user's awareness of the security policy, as one property of the database db' we have:

$$db' \models \neg\Psi \text{ for every } \Psi \in pot\_sec \tag{2}$$

We define the set of formulas $Neg(pot\_sec) := \{\neg\Psi | \Psi \in pot\_sec\}$ and try to find an interpretation I' that satisfies all formulas in $Neg(pot\_sec)$ in order for db' to fulfill (2).

Now, let us turn to the availability feature (iii.).[2] To have a measure for the availability of db' we define a distance between an interpretation I and an interpretation J with respect to a set of propositional variables $\mathcal{V}$ as follows: $dist_{\mathcal{V}}(I, J) := ||\{A | A \in \mathcal{V}, I(A) \neq J(A)\}||$. That is, we count all variables in $\mathcal{V}$ having in one interpretation a value distinct from the value in the other interpretation. As we want to maximize availability, we have to minimize the distance of the new interpretation I' with respect to the original interpretation I and all variables in $\mathcal{P}$: $dist_{\mathcal{P}}(I, I') \longrightarrow min$.

The consistency feature (iv.) means that we want to find an interpretation I' such that all formulas in *prior* are satisfied.

All in all we conclude that I' has to be an interpretation (for the variables in $\mathcal{P}$) that has minimal distance to the original interpretation I and satisfies the set of formulas $prior \cup Neg(pot\_sec)$.
Under the requirements (a)–(e) given in Section 2, such a satisfying interpretation always exists. To prove this, first of all note that requirement (e) implies that *pot_sec_disj* is not a tautology. Combining requirements (a) and (e), we conclude that *prior* is consistent with the set $Neg(pot\_sec)$. Thus, there exists at least one interpretation I' satisfying $prior \cup Neg(pot\_sec)$.

4 A "Branch and Bound"-SAT-solver

In order to find interpretation I', we combine SAT-solving (for the completeness and satisfiability requirements) with "Branch and Bound" (for the minimization requirement). The database db' representing I' will be inference-proof by construction, as we describe in the following.

[2] Availability in this context is defined as "containing as much correct information as possible"; in contrast, reliability is defined as "the user knows that the answer he got from db' is correct".

SAT-solvers 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 DLL-algorithm (see [7, 6]). It implements:

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": if in all clauses only the positive literal or only the negative literal of a variable occurs (a so-called "pure literal"), then the literal can be evaluated 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 value is assigned to a variable, the set of clauses can be simplified by subsumption (if a clause contains a literal that is evaluated to *true*, remove the whole clause) or resolution (if a clause contains a literal that is evaluated to *false*, remove this literal from the clause but keep the remaining clause).

"Branch and Bound" (B&B, for short) is a method for finding solutions to an optimization problem. It offers the features "branching" (dividing the problem into adequate subproblems), "bounding" (efficiently computing local lower and upper bounds for subproblems), and "pruning" (discarding a subproblem due to a bad bound value). For a minimization problem a global upper bound is maintained stating the currently best value. A B&B-algorithm may have a superpolynomial 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.

4.1 The Algorithm

First of all, we consider the case where $prior \cup Neg(pot\_sec)$ is a set of formulas in conjunctive normal form (CNF); the general case will be treated in Section 4.2. We define the set of clauses $C_{decision}$ as

$$C_{decision} := \bigcup_{\Psi \in prior \cup Neg(pot\_sec)} \texttt{clauses}(\Psi)$$

where $\texttt{clauses}(\Psi)$ is a set of clauses representing $\Psi \in prior \cup Neg(pot\_sec)$. $C_{decision}$ is the input to our SAT-solving algorithm. Let $\mathcal{P}_{decision} \subset \mathcal{P}$ be the set of all variables occurring in $C_{decision}$; these are the "decision variables". Our SAT-solver finds an interpretation $I'_{decision}$ for all decision variables; all other variables get assigned the same truth value as before: $I'(A) := I(A)$ if $A \in \mathcal{P} \setminus \mathcal{P}_{decision}$. Thus, we have $dist_{\mathcal{P} \setminus \mathcal{P}_{decision}}(I, I') = 0$. We find an interpretation $I'_{decision}$ for $\mathcal{P}_{decision}$ satisfying $C_{decision}$ with minimal distance to I by employing a branch and bound algorithm. Listings 1, 2, 3 and 4 show the four functions "initialization", "best-first-search splitting", "boolean constraint propagation", and "simplification of clauses" of our algorithm. In the following we describe each function in detail.

1. **Initialization** for the root node r
   1.1. $I_r(A) := \mathit{undefined}$ for all $A \in \mathcal{P}_{decision}$
   1.2. $lb_r := 0$; $ub_r := \mathit{depth}_{max}$; $ub_{global} := ub_r$
   1.3. $C_r := C_{decision}$
   1.4. `BCP(`I_r,C_r,lb_r,ub_r`)`

Listing 1. Initialization for root node r

B&B on the set $\mathcal{P}_{decision}$ yields a binary tree; its maximal depth is the cardinality of $\mathcal{P}_{decision}$: $\mathit{depth}_{max} = ||\mathcal{P}_{decision}||$. The binary structure of the tree is created as follows. In every node v a "splitting variable" $A \in \mathcal{P}_{decision}$ is selected; we refer to this variable by $\mathit{splitvar}(v)$. Then, a left and a right child node v_{left} and v_{right} are constructed; in one of the child nodes A is set to *true* and in the other child node A is set to *false*. This is the splitting step of the DLL-algorithm as well as the branching step of B&B. We conduct a "best-first search" with our B&B algorithm: in the left child node we assign $\mathit{splitvar}(v)$ the same truth value as in I (this choice yields better local bounds and we process the left child node first; see Listing 2, line 2.3.), and in the right child node we assign $\mathit{splitvar}(v)$ the opposite truth value $\overline{I(\mathit{splitvar}(v))}$ (which yields worse local bounds; see line 2.6.). A splitting step is pictured in Figure 1.

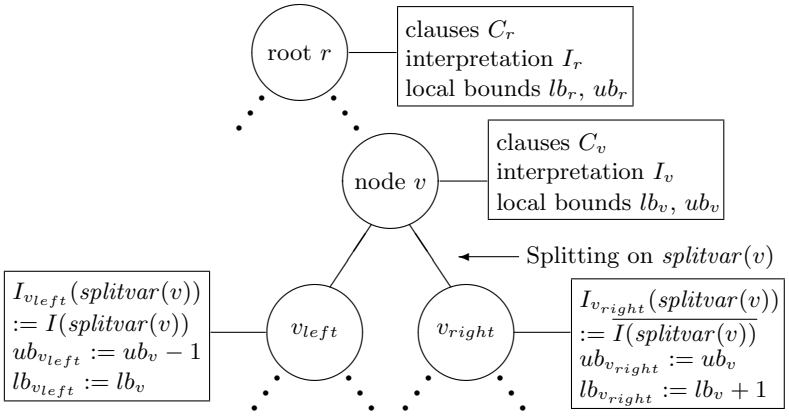

Fig. 1. A splitting step

Each node v has its own set of clauses C_v. The child nodes get each a simplified set of clauses constructed from C_v by subsumption and resolution (see lines 2.5.1. and 2.8.1. and Listing 4). Initially, for the root node r the set of clauses is $C_r = C_{decision}$ (see Listing 1, line 1.3.).
Before splitting takes place in node v, we carry out BCP: unit clauses are repeatedly eliminated from C_v until there are none left (see Listing 3, line 3.2.). Variables in unit clauses are assigned a value without splitting on these variables; thus, BCP reduces the number of branches in the tree.

2. SPLIT(I_v,C_v,lb_v,ub_v): **Best-First-Search-Splitting** on $A \in \mathcal{P}_{decision}$
   2.1. generate two child nodes v_{left} and v_{right}
   2.2. copy I_v into $I_{v_{left}}$ and $I_{v_{right}}$
   2.3. $I_{v_{left}}(A) := I(A)$; $ub_{v_{left}} := ub_v - 1$; $lb_{v_{left}} := lb_v$
   2.4. if ($lb_{v_{left}} > ub_{global}$) { GOTO line 2.6. }
   2.5. else
      2.5.1. $C_{v_{left}}$:= SIMP($I_{v_{left}}$,C_v,ub_v)
      2.5.2. BCP($I_{v_{left}}$,$C_{v_{left}}$,$lb_{v_{left}}$,$ub_{v_{left}}$)
   2.6. $I_{v_{right}}(A) := \overline{I(A)}$; $ub_{v_{right}} := ub_v$; $lb_{v_{right}} := lb_v + 1$
   2.7. if ($lb_{v_{right}} > ub_{global}$) { RETURN }
   2.8. else
      2.8.1. $C_{v_{right}}$:= SIMP($I_{v_{right}}$,C_v,ub_v)
      2.8.2. BCP($I_{v_{right}}$,$C_{v_{right}}$,$lb_{v_{right}}$,$ub_{v_{right}}$)

Listing 2. Best-First-Search Splitting

3. BCP(I_v,C_v,lb_v,ub_v): **Boolean Constraint Propagation** in node v
   3.1. C_v^{unit} := set of unit clauses of C_v
   3.2. while ($C_v^{unit} \neq \emptyset$)
      3.2.1. foreach clause $[l] \in C_v^{unit}$ (with $l = A$ or $l = \neg A$ for an $A \in \mathcal{P}_{decision}$)
         3.2.1.1. remove $[l]$ from C_v^{unit}; set $I_v(A)$ such that $I_v \models l$
         3.2.1.2. if ($I_v(A) == I(A)$) $\{ub_v - -\}$
         3.2.1.3. else $\{lb_v + +\}$
      3.2.2. if ($lb_v > ub_{global}$) { RETURN }
      3.2.3. else
         3.2.3.1. C_v := SIMP(I_v,C_v,ub_v); C_v^{unit} := set of unit clauses of C_v
   3.3. if ($v == r$ and $ub_{global} > ub_r$) $\{ub_{global} := ub_r\}$
   3.4. if ($C_v \neq \emptyset$) {SPLIT(I_v,C_v,lb_v,ub_v)}

Listing 3. Boolean Constraint Propagation

Each node v represents an interpretation I_v containing all variable assignments occurring on the path from the root node r to the node v. Initially, for the root node r all values are undefined (see Listing 1, line 1.1.).
In each node v also the local lower bound lb_v and the local upper bound ub_v are defined. The global upper bound is called ub_{global}. We compute the bounds using the availability distance defined above; this is the bounding step of B&B. The lower bound lb_v is the number of variables that are assigned in I_v a value distinct from the value in the original interpretation I; the upper bound ub_v is lb_v plus the number of variables that are still undefined in I_v; *if* the final interpretation $I'_{decision}$ can be found in this branch, it has at least distance lb_v and at most distance ub_v from I: $lb_v \leq dist_{\mathcal{P}_{decision}}(I, I'_{decision}) \leq ub_v$. Initially, the bounds are $lb_r = 0$ and $ub_r = depth_{max}$; ub_{global} is initialized to $depth_{max}$, too (see line 1.2.).
During BCP the bounds are accordingly adjusted (see lines 3.2.1.2. and 3.2.1.3.). After BCP in the root node r, the global upper bound can already be decre-

4. `SIMP(`I_v,C_v,ub_v`)`: **Simplification** of a set of clauses C_v given interpretation I_v
   4.1. initialize set of clauses $C_{return} := \emptyset$
   4.2. `foreach` clause c in C_v
      4.2.1. `foreach` literal l in c:
         4.2.1.1. `if (`$I_v \models l$ `)` {**Subsumption:** `CONTINUE` (ignore c)}
         4.2.1.2. `else if (`$I_v \not\models l$ `)` {**Resolution:** remove l from c}
      4.2.2. `if (`c empty clause`)` { `RETURN` (I_v not satisfying)}
      4.2.3. `else` {add c to C_{return}}
   4.3. `if (`$C_{return} == \emptyset$ and $ub_{global} > ub_v$`)` $\{ub_{global} := ub_v\}$
   4.4. `return` C_{return}

Listing 4. Simplification of clauses

mented (see line 3.3.); every satisfying assignment $I'_{decision}$ has to satisfy the unit clauses in the root node. Subsequently, ub_{global} may only be adjusted when a complete and satisfying interpretation I_v is found and $ub_v < ub_{global}$ (see line 4.3.).
During splitting in an arbitrary node v, in the left child node only the upper bound has to be decremented, while in the right child node only the lower bound has to be incremented (see lines 2.3. and 2.6.).
For the pruning of B&B to take place, there are two possible conditions:

1. [**bad lower bound**] the local lower bound of node v is worse than the current global upper bound: $lb_v > ub_{global}$; we have already found a better solution and we are not able to expand I_v to an interpretation for all decision variables with minimal distance
2. [**unsatisfiability**] while constructing I_v we encountered an inconsistency: we are not able to expand I_v to an interpretation for all decision variables that satisfies $C_{decision}$

The lower bound condition is checked in lines 2.4., 2.7. and 3.2.2. after new lower bounds have been calculated. The unsatisfiability condition is checked in line 4.2.2. during simplification: if an empty clause is generated, the current interpretation is not satisfying.

In the subsumption step it may happen that variables "disappear" from the clauses: there are no clauses left that contain the variables. Thus, it may be the case that in the final interpretation $I'_{decision}$ there are still some undefined variables (they are often referred to as "don't care variables"). However, these variables had no influence on the satisfiability of the clauses; these variables are removed from $P_{decision}$ and thus are assigned the same truth value as in the original interpretation.

Having found $I'_{decision}$ satisfying $C_{decision}$ and having minimal distance to I, we can construct our inference-proof database db' as follows:

$$\text{for all } A \in \mathcal{P}_{decision}: \qquad A \in db' \text{ iff } I'_{decision}(A) = 1$$
$$\text{for all } A \in \mathcal{P} \setminus \mathcal{P}_{decision}: A \in db' \text{ iff } A \in db \text{ (that is } I(A) = 1)$$

If $||\mathcal{P}_{decision}|| \ll ||\mathcal{P}||$, it may be more efficient to just maintain a small separate database $db_{decision}$, in this special case including possibly negated entries:

$$\text{for all } A \in \mathcal{P}_{decision} \begin{cases} A \in db_{decision} & \text{iff } I'_{decision}(A) = 1 \\ \neg A \in db_{decision} & \text{iff } I'_{decision}(A) = 0 \end{cases}$$

$db_{decision}$ returns the truth values for all variables from $\mathcal{P}_{decision}$. The original database db returns the truth values of variables from $\mathcal{P} \setminus \mathcal{P}_{decision}$.

4.2 Some Remarks on Further Techniques

There exist several adaptations of the basic DLL-algorithm to non-CNF formulas. For example, Ganai et al. [13] introduce hybrid SAT for boolean circuits where only newly added ("learned") clauses are in CNF. Propagation in circuits is supported by a lookup table and "watched literals" (see also [21]). Giunchiglia et al. [14] apply renaming of subformulas by adding new variables (thus constructing an equisatisfiable formula in CNF for a non-CNF input formula) but propose to split only on the original ("independent") variables. Thiffault et al. [17] represent the non-CNF input formula as a directed acyclic graph and base propagation on "watch children" and "don't care watch parents". With such techniques our B&B-algorithm can be extended to accept non-CNF formulas. Moreover, a vast number of optimization techniques and splitting heuristics have been proposed for the basic DLL-algorithm; for example, subsumption removal [20], reduction of the number of clauses [10] or elimination of variables [16]. Such techniques can be employed to speed up the search process.

Our availability distance may not be the only optimization criterion; the bounding can easily be extended by other measures to guide the search and determine the quality of a solution. Equally, a preference relation on propositions can be employed so that lying for a lower-ranked proposition is preferred to lying for a higher-ranked proposition.

Lastly, let us note that in our algorithm pure literals cannot be removed, that is, the DLL-"affirmative-negative rule" cannot be applied: setting pure literals to *true* may lead to an interpretation that does not have minimal distance. Anyway, detection of pure literals is expensive (see for example [21]) and often omitted in SAT-solver implementations.

4.3 An Example Run

Let us come back to our example. We consider db, *prior* and pot_sec_1. The set $prior \cup Neg(pot\_sec_1)$ is a set of CNF-formulas and we have as a set of clauses (clauses are written with square brackets):

$$C_{decision} = \{[\neg\texttt{brokenArm}, \texttt{lowWorkCpcty}], [\neg\texttt{brokenLeg}, \texttt{highCosts}],$$
$$[\neg\texttt{cancer}, \texttt{highCosts}], [\neg\texttt{cancer}, \texttt{lowWorkCpcty}], [\neg\texttt{lowWorkCpcty}]\}.$$

Figure 2 shows the tree created by our algorithm. We need just one splitting step; all other assignments are determined by BCP.

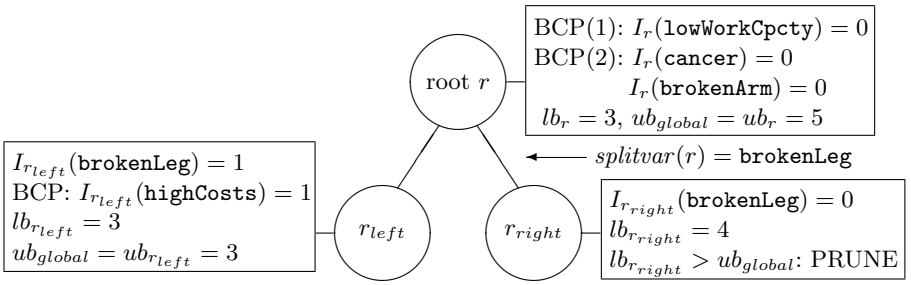

Fig. 2. An example run

In root r we have two BCP steps: the first one on $\{[\neg\texttt{lowWorkCpcty}]\}$ and the second one on $\{[\neg\texttt{cancer}], [\neg\texttt{brokenArm}]\}$. This yields the interpretation $I_r(\texttt{cancer}) = I_r(\texttt{brokenArm}) = I_r(\texttt{lowWorkCpcty}) = 0$ and from this we compute the bounds $lb_r = 3$ and $ub_{global} = ub_r = 5$.
Next we split on $splitvar(r) = \texttt{brokenLeg}$: we construct the child nodes r_{left} and r_{right} and begin with $I_{r_{left}}(\texttt{brokenLeg}) = 1$ (best-first search). We have $lb_{r_{left}} = 3$ and $ub_{r_{left}} = 4$ and simplification produces the new unit clause $[\texttt{highCosts}]$. BCP on this clause results in $lb_{r_{left}} = 3$ and $ub_{r_{left}} = 3$; we have found a complete and satisfying interpretation in this branch and we set $ub_{global} = ub_{r_{left}} = 3$.
We now treat r_{right} and set $I_{r_{right}}(\texttt{brokenLeg}) = 0$, $lb_{r_{right}} = 4$ and $ub_{r_{right}} = 5$. We find that $lb_{r_{right}} > ub_{global}$ and this branch is pruned.
Thus, the optimal solution is $I'(\texttt{brokenLeg}) = I'(\texttt{highCosts}) = 1$ on the one hand and $I'(\texttt{cancer}) = I'(\texttt{brokenArm}) = I'(\texttt{lowWorkCpcty}) = 0$ on the other hand; the transformed database is $db' = \{\texttt{brokenLeg}, \texttt{highCosts}\}$ with distance 3 to the original database.

5 Conclusion and Future Work

We presented an algorithm to preprocess an inference-proof database based on a user's a priori knowledge and a specification of secret information. While the worst-case runtime is exponential, there is a good chance to find an acceptable (or even optimal) solution in a smaller amount of time. Even if we stop the algorithm prematurely and accept a suboptimal solution, it is a solution that is definitely inference-proof; it might only be suboptimal with respect to availability: there may be more lies in the database than necessary.

Future work shall investigate how this approach can be adapted to other CQE parameters, namely, the unknown policy case, refusal as a restriction method and incomplete databases. Furthermore, this method shall be expanded to other logics (for example first-order logic). In a wider setting, we want to connect CQE to other established research areas, for example linear or constraint programming [12] and theory merging [8]. Moreover, a comparison of the CQE system to existing approaches for general purpose databases (see e.g. [11]) has already been initiated.

One of the fundamental problems inherent to every inference control system still remains: It is difficult – if not impossible – to appropriately model the user's knowledge in the system. Apart from that, there are other interesting questions regarding the user's knowledge in the approach presented here. For example, if there is one transformed database db'_1 for a fixed $prior_1$, is it possible to reuse (parts of) db'_1 to construct a db'_2 for a different $prior_2$? Similarly, if we allow the user's knowledge to change at runtime due to input from external sources, is it possible to adjust db' to the new situation? These topics will be covered in the near future.

Acknowledgements

This work was funded by the German Research Council (DFG) under a grant for the Research Training Group (Graduiertenkolleg) "Mathematical and Engineering Methods for Secure Data Transfer and Information Mediation" organized at Ruhr-Universität Bochum, Universität Dortmund, Universität Essen and Fernuniversität Hagen.

References

1. Joachim Biskup and Piero A. Bonatti. Lying versus refusal for known potential secrets. *Data & Knowledge Engineering*, 38(2):199–222, 2001.
2. 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.
3. 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.
4. 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.
5. LiWu Chang and Ira S. Moskowitz. A study of inference problems in distributed databases. In Ehud Gudes and Sujeet Shenoi, editors, *16th Annual IFIP WG 11.3 Conference on Data and Applications Security, Proceedings*, pages 191–204. Kluwer, 2002.
6. Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
7. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.
8. 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.
9. Josep Domingo-Ferrer, editor. *Inference Control in Statistical Databases, From Theory to Practice*, volume 2316 of *Lecture Notes in Computer Science*. Springer, 2002.

10. 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.
11. Csilla Farkas and Sushil Jajodia. The inference problem: A survey. *SIGKDD Explorations*, 4(2):6–11, 2002.
12. Thom Frühwirth and Slim Abdennadher. *Essentials of Constraint Programming*. Springer, 2003.
13. Malay K. Ganai, Pranav Ashar, Aarti Gupta, Lintao Zhang, and Sharad Malik. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *39th Design Automation Conference, Proceedings*, pages 747–750. ACM, 2002.
14. Enrico Giunchiglia and Roberto Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In Evelina Lamma and Paola Mello, editors, *6th Congress of the Italian Association for Artificial Intelligence, Proceedings*, volume 1792 of *Lecture Notes in Computer Science*, pages 84–94. Springer, 2000.
15. John Hale and Sujeet Shenoi. Analyzing fd inference in relational databases. *Data & Knowledge Engineering*, 18(2):167–183, 1996.
16. 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, Online Proceedings*, 2004.
17. Christian Thiffault, Fahiem Bacchus, and Toby Walsh. Solving non-clausal formulas with DPLL search. In *7th International Conference on Theory and Applications of Satisfiability Testing, Online Proceedings*, 2004.
18. 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.
19. 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.
20. 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.
21. Lintao Zhang and Sharad Malik. The quest for efficient boolean satisfiability solvers. In Andrei Voronkov, editor, *18th International Conference on Automated Deduction, Proceedings*, volume 2392 of *Lecture Notes in Computer Science*, pages 295–313. Springer, 2002.