Constraint Acquisition
You Can Chase but You Cannot Find

Sven Hartmann **Sebastian Link*** **Thu Trinh**

Department of Information Systems, Information Science Research Centre
Massey University, Palmerston North, New Zealand
E-mail: [S.Hartmann,S.Link]@massey.ac.nz

Abstract

We identify established tableaux techniques as an invaluable tool for semantic knowledge acquisition in the design process of relational databases.

Sample databases allow users and designers to judge, justify, convey and test their understanding of the semantics of the future database. In the case of integrity constraints such sample data can provide considerable assistance for deciding whether a constraint captures desirable information about the database or not. Since constraints can be particularly difficult to grasp in practice sample databases offer a convenient tool to confirm or reject the usefulness of potential candidate constraints.

We pinpoint the Chase and analytical tableau as two tableaux techniques that are able to automatically generate sample databases for large classes of integrity constraints. The Chase can be used for generating sample data that allows us to reject candidate constraints. However, analytical tableaux enable us to find all minimal sample databases which enable us to either accept or reject a candidate constraint.

1 Introduction

Database design is based on the assumption that the semantics of the underlying application has been completely captured. However, the complete acquisition of such knowledge presents many challenges. During the design process or in the lifetime of the database it only happens all too often that the knowledge about the database has been proven incomplete. Therefore, the acquisition of semantic knowledge about the application domain is absolutely crucial for the quality of the database.

Integrity constraints are conditions that every legal database instance is compelled to obey. They restrict databases to those considered meaningful for the application at hand. The acquisition and specification of integrity constraints is far from being trivial. This task does not only demand high abstraction abilities but also tends to be rather complex. For the correct utilisation of semantic information an advanced understanding of logics is required. Sometimes designers misunderstand integrity constraints and, consequently, interpret them badly. In addition, if designers and users work together it may be that they interpret the same constraint in different ways. Each of the participants in the requirements acquisition process has their own skills, experience and knowledge. In view of these problems and the importance of the sound and complete gathering of semantic information it is highly desirable to support the acquisition of semantic constraints.

In general, constraint acquisition can be a process of elimination: some participant suggests the significance of some integrity constraint φ for the future database, and according to the mutual feeling φ will be added to the set Σ of constraints that have already been specified explicitely or φ will be discarded (added to the set Υ of constraints that do not need to be specified). Any constraint φ that is implied by Σ has already been specified implicitely and can therefore be added to Υ. However, deciding implication may be a non-trivial task. At the end of the elimination process all potential candidates have been considered, and the set Σ contains all those constraints that will be explicitely specified. This process is illustrated in Table 1. The crucial question is how to deal with those candidates φ that are not implied by Σ? We need some kind of examples that allow us to better comprehend the consequences of not specifying φ.

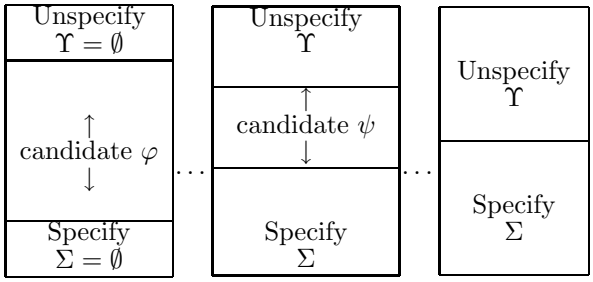


Table 1: Constraint Acquisition by Elimination

Armstrong databases constitute an invaluable tool for the acquisition of integrity constraints. They form one of the few instances in which example-based deduction can be used to obtain a sound and complete inference mechanism. In fact, an Armstrong database for a constraint set Σ satisfies an integrity constraint φ precisely if φ is implied by Σ. In this sense, Armstrong databases are user-friendly representations of constraint sets. In particular, every constraint φ not implied by Σ is violated by an Armstrong database for Σ. Hence, the consequences of not specifying φ can somewhere be identified as a violation in the Armstrong database. However, as such a database must violate all the constraints not implied by Σ Armstrong databases tend to have many tuples. For instance, the minimum size of an Armstrong re-

*This research was supported by Marsden Funding, Royal Society of New Zealand

lation for an arbitrary system of minimal keys over n attributes has lower bound $\frac{1}{n^2}\binom{n}{\lfloor \frac{n}{2} \rfloor}$ (Demetrovics & Gyepesi 1983, Katona & Tichler 2006). Hence, it may become rather difficult for the users and designers to focus on the specific candidate constraint under consideration. Moreover, if domains of certain attributes have a low cardinality, then Armstrong databases may not even exist.

In this paper we choose a weaker approach towards sample databases: for each candidate φ we generate sample databases that satisfies Σ and violate φ, if φ is not implied by Σ. In fact, we generate counterexample databases for the implication of Σ by φ. As was the case with Armstrong databases, our sample databases show explicitely the consequences of not specifying φ. Unlike Armstrong databases we are not concerned with violating all the other constraints that are not implied by Σ. The advantage of this approach is that our sample databases will be smaller than Armstrong databases, i.e., contain less database tuples. Designers and users benefit from small sample data, and their attention is focused on the particular candidate φ. If they agree that violations of φ are acceptable in future database instances, then φ can be discarded. In case that all violations of φ are unacceptable in future databases φ should be specified explicitely.

We consider the question of how to generate such counterexample databases automatically. We will not be able to deal with arbitrary classes of integrity constraints but will concentrate on some classes which are commonly found in database practice. The *Chase*-procedure is a popular technique to decide the implication of functional and join dependencies. We will demonstrate that the Chase is also highly useful for generating counterexample databases. Hence, this established tool can also be successfully employed during the constraint acquisition process. It turns out that the *Chase* does not always produce a minimal counterexample relation. This is illustrated in Example 1.1. Moreover, the *Chase* will only produce a single counterexample relation. If this relation is acceptable as a future database instance, then we cannot specify φ. However, if the relation is unacceptable, then we cannot conclude whether to specify φ or not.

Example 1.1. *Suppose we have a relation schema* FILM *with three attributes* Movie, Writer *and* Actor. *Let* Σ *consist of the multivalued dependency* Movie $\twoheadrightarrow$ Writer, *stating that movies determine their writers independently of their actors. Suppose we are thinking about specifying the candidate functional dependency* Movie $\rightarrow$ Writer, *stating that each movie determines its writer. We apply the chase to decide whether* φ *is implied by* Σ. *Therefore, we start with the following template*

Movie	*Writer*	*Actor*
a_M	a_W	a_A
a_M	b_1	b_2

with distinguished variables a_M, a_W, a_A *and nondistinguished variables* b_1, b_2. *The first tuples consists of distinguished variables only. The second tuple contains distinguished variables only in those columns that represent attributes occurring in the left-hand side of the FD under consideration. Since we are chasing by* Σ *the* JD*-rule enforces the following update on the template:*

Movie	*Writer*	*Actor*
a_M	a_W	a_A
a_M	b_1	b_2
a_M	a_W	b_2
a_M	b_1	a_A

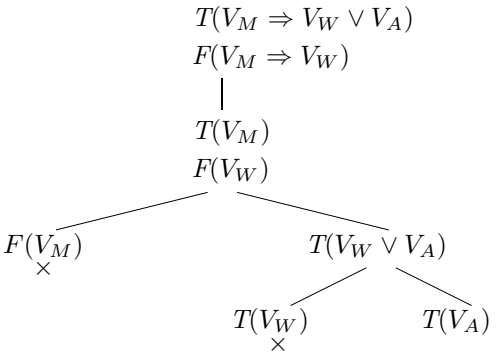
which is the final result of chasing φ *by* Σ. *Since the* Writer*-column contains variables different from* a_W *it follows that* Movie $\rightarrow$ Writer *is not implied by* Σ. *In fact, the relation above is a counterexample relation for the implication of* φ *by* Σ. *After some substitutions this database relation might be better represented as follows:*

Movie	*Writer*	*Actor*
The Cincinatti Kid	*R.Jessup*	*S. McQueen*
The Cincinatti Kid	*R.Lardner Jr.*	*E.G.Robinson*
The Cincinatti Kid	*R.Jessup*	*E.G.Robinson*
The Cincinatti Kid	*R.Lardner Jr.*	*S. McQueen*

This relation may substantiate the belief not to specify the FD above. However, if the mutual feeling suggests that only movies with a single writer will be stored in the database, then the counterexample above pinpoints this issue. □

The main contribution of this paper is the observation that propositional tableaux can be applied to generate *all* 2-tuple counterexample relations for the implication of φ by Σ where $\Sigma \cup \{\varphi\}$ constitutes a set of Boolean and multivalued dependencies. In fact, since the violation of any database dependency requires at least two distinct tuples the 2-tuple counterexample relations will be optimal. The idea of exploring tableaux for generating counterexamples is possible since the implication of Boolean and multivalued dependencies corresponds to the logical implication of certain fragments in Boolean propositional formulae.

Example 1.2. *Suppose* FILM, Σ *and* φ *are given as in Example 1.1. We now choose to represent the MVD in* Σ *as propositional formulae* $V_M \Rightarrow V_W \vee V_A$, *and* φ *as* $V_M \Rightarrow V_W$. *The following propositional tableau attempts to refute* $\Sigma \cup \{\neg\varphi\}$.



The right-most branch of this tableau is not closed, hence $\Sigma \cup \{\neg\varphi\}$ *is satisfiable, and the open branch defines a model for* $\Sigma \cup \{\neg\varphi\}$, *i.e., a counterexample truth evaluation for the implication of* φ *by* Σ. *We may now use this truth evaluation to obtain a two-tuple counterexample relation for the implication of the FD* φ *by the MVD in* Σ*:*

Movie	*Writer*	*Actor*
1	*1*	*1*
1	*0*	*1*

For instance, both tuples have the same entry in the Actor*-column since the truth value of the propositional variable* V_A, *associated with the attribute* Actor, *must be interpreted as* true *according to the open branch in the tableau above. As before, one may rename the values in the table to be more practical:*

Movie	*Writer*	*Actor*
The Godfather	*M.Puzzo*	*M.Brando*
The Godfather	*F.F.Coppola*	*M.Brando*

Again, the relation pinpoints the consequences of not specifying the functional dependency Movie → Writer*, but requires only two tuples.* □

Related Work. Armstrong databases have been well-studied in the past (Armstrong 1974, Beeri, Dowd, Fagin & Statman 1984, Demetrovics & Gyepesi 1983, Fagin 1982*a*, Fagin 1982*b*, Fagin & Vardi 1983, Gottlob & Libkin 1990, Grant & Jacobs 1982, Katona & Tichler 2006, Khardon, Mannila & Roth 1999, Mannila & Räihä 1986, Mannila & Räihä 1985, De Marchia & Petit 2007), but may contain too many tuples to draw the user's attention to the eligibility of some candidate constraint. Our sample databases are weaker than Armstrong databases in the sense that we only violate the candidate constraint in question (if possible) but not all the other constraints not implied by Σ.

The Chase was originally introduced as a method for testing the implication of data dependencies by a set of data dependencies (Maier, Mendelzon & Sagiv 1979), and has received considerable interest ever since (Beeri & Vardi 1984, Maier, Sagiv & Yannakakis 1981, Sciore 1982, Popa, Deutsch, Sahuguet & Tannen 2000, Sadri & Ullman 1982). To the authors' best knowledge nobody has observed yet the potential of the Chase to generate counterexample relations that are useful for rejecting potential candidate constraints. While the Chase can be applied to several classes of integrity constraints we will focus on functional and join dependencies here.

The correspondences between the implication of Boolean and multivalued dependencies and the logical implication of propositional formulae fragments is well-known (Sagiv, Delobel, Parker Jr. & Fagin 1981, Sagiv, Delobel, Parker Jr. & Fagin 1987). However, the observation to explore propositional tableaux techniques (Beth 1959, Hintikka 1955, Smullyan 1995) for generating all minimal counterexample relations is original. These 2-tuple relations present invaluable assistance for designers and users in deciding whether a candidate constraint captures desirable domain knowledge or not.

Organisation. We will summarise the necessary notions from the relational model of data in Section 2. Mainly, we will repeat the notions of functional, Boolean, multivalued and join dependencies. We will also briefly summarise the correspondences between the implication of Boolean and multivalued dependencies and the logical implication of propositional fragments. In Section 3 we demonstrate how the Chase can be used to reject candidate constraints that are either functional or join dependencies. The technique of analytical tableaux is summarised in Section 4. In Section 5 we apply this technique to reject or specify candidate constraints that are either Boolean or multivalued dependencies. Particular emphasis is put on the generation of all 2-tuple counterexample relations. We demonstrate the applicability of this technique by a detailed example in Section 6. We conclude in Section 7.

2 Preliminaries

Let $\mathfrak{A} = \{A_1, A_2, \ldots\}$ be a (countably) infinite set of symbols, called *attributes*. A *relation schema* is a finite set $R = \{A_1, \ldots, A_n\}$ of distinct *attributes* from $\mathfrak{A}$, which represent column names of a relation. Each attribute A_i of a relation schema is associated an infinite domain $dom(A_i)$ which represents the set of possible values that can occur in the column named A_i. If X and Y are sets of attributes, then we may write XY for $X \cup Y$. If $X = \{A_1, \ldots, A_m\}$, then we may write $A_1 \cdots A_m$ for X. In particular, we may write simply A to represent the singleton $\{A\}$. A *tuple* over $R = \{A_1, \ldots, A_n\}$ (R-tuple or simply tuple, if R is understood) is a function $t : R \to \bigcup_{i=1}^{n} dom(A_i)$ with $t(A_i) \in dom(A_i)$ for $i = 1, \ldots, n$. For $X \subseteq R$ let $t[X]$ denote the restriction of the tuple t over R on X, and $dom(X) = \prod_{A \in X} dom(A)$ the Cartesian product of the domains of attributes in X. A *relation* r over R is a finite set of tuples over R. The relation schema R is also called the domain $Dom(r)$ of the relation r over R. Let $r[X] = \{t[X] \mid t \in r\}$ denote the *projection* of the relation r over R on $X \subseteq R$. For $X, Y \subseteq R$, finite $r_1 \subseteq dom(X)$ and $r_2 \subseteq dom(Y)$ let $r_1 \bowtie r_2 = \{t \in dom(XY) \mid \exists t_1 \in r_1, t_2 \in r_2 \text{ with } t[X] = t_1[X] \text{ and } t[Y] = t_2[Y]\}$ denote the *natural join* of r_1 and r_2. Note that the 0-ary relation $\{()\}$ is the projection $r[\emptyset]$ of r on $\emptyset$ as well as left and right identity of the natural join operator.

2.1 Relational Dependencies

Functional dependencies (FDs) between sets of attributes have played a central role in the study of relational databases (Bernstein 1976, Biskup, Dayal & Bernstein 1979, Codd 1970), and seem to be central for the study of database design in other data models as well (Arenas & Libkin 2004, Levene & Loizou 1998, Hartmann & Link 2006, Tari, Stokes & Spaccapietra 1997, Weddell 1992, Wijsen 1999). The notion of a functional dependency is well-understood and the semantic interaction between these dependencies has been syntactically captured by Armstrong's well-known axioms (Armstrong 1974). A *functional dependency* (FD) (Codd 1972) on the relation schema R is an expression $X \to Y$ where $X, Y \subseteq R$. A relation r over R *satisfies* the FD $X \to Y$, denoted by $\models_r X \to Y$, if and only if every pair of tuples in r that agrees on each of the attributes in X also agrees on the attributes in Y. That is, $\models_r X \to Y$ if and only if $t_1[Y] = t_2[Y]$ whenever $t_1[X] = t_2[X]$ holds for any $t_1, t_2 \in r$.

FDs are incapable of modelling many important properties that database users have in mind. Multivalued dependencies (MVDs) provide a more general notion and offer a response to the shortcomings of FDs. A *multivalued dependency* (MVD) (Delobel 1978, Fagin 1977) on R is an expression $X \twoheadrightarrow Y$ where $X, Y \subseteq R$. A relation r over R *satisfies* the MVD $X \twoheadrightarrow Y$, denoted by $\models_r X \twoheadrightarrow Y$, if and only if for all $t_1, t_2 \in r$ with $t_1[X] = t_2[X]$ there is some $t \in r$ with $t[XY] = t_1[XY]$ and $t[X(R - XY)] = t_2[X(R - XY)]$. Informally, the relation r satisfies $X \twoheadrightarrow Y$ when the value on X determines the set of values on Y independently of the set of values on $R - XY$. This actually suggests that the relation schema R is overloaded in the sense that it carries two independent facts XY and $X(R-XY)$. More precisely, it is shown in (Fagin 1977) that MVDs "provide a necessary and sufficient condition for a relation to be decomposable into two of its projections without loss of information (in the sense that the original relation is guaranteed to be the join of the two projections)". This means that $\models_r X \twoheadrightarrow Y$ if and only if $r = r[XY] \bowtie r[X(R - XY)]$. This characteristic of MVDs is fundamental to relational database design and 4NF (Fagin 1977).

Multivalued dependencies can be further generalised. A *join dependency* (JD) over a relation schema R is an experssions of the form $\bowtie\{X_1, \ldots, X_n\}$ where $X_1, \ldots, X_n \subseteq R$ and $\cup_{i=1}^{n} X_i = R$. An R-relation $r \subseteq dom(R)$ *satisfies* $\bowtie\{X_1, \ldots, X_n\}$ if $r = \pi_{X_1}(r) \bowtie \cdots \bowtie \pi_{X_n}(r)$ holds. In particular, r satisfies the MVD $X \twoheadrightarrow Y$ precisely if r satisfies the JD $\bowtie\{XY, X(R - XY)\}$. The basic motivation for join

dependencies stems from their usefulness in connection with relation decomposition.

The set of *Boolean dependencies* (BDs) over a relation schema R is the smallest set satisfying the following properties:

- every $A \in R$ is a Boolean dependency,
- if φ is a Boolean dependency, then $\neg\varphi$ is a Boolean dependency,
- if φ, ψ are Boolean dependencies, then $(\varphi \wedge \psi), (\varphi \vee \psi)$ and $(\varphi \Rightarrow \psi)$ are Boolean dependencies.

An R-relation r is said to *satisfy* the Boolean dependency φ over R if for all *distinct* $t_1, t_2 \in r$ the following holds:

- if $\varphi = A \in R$, then $t_1[A] = t_2[A]$,
- if $\varphi = \neg\psi$, then $\{t_1, t_2\}$ satisfies φ if and only if $\{t_1, t_2\}$ does not satisfy ψ,
- if $\varphi = (\varphi_1 \wedge \varphi_2)$, then $\{t_1, t_2\}$ satisfies φ if and only if $\{t_1, t_2\}$ satisfies φ_1 and $\{t_1, t_2\}$ satisfies φ_2,
- if $\varphi = (\varphi_1 \vee \varphi_2)$, then $\{t_1, t_2\}$ satisfies φ if and only if $\{t_1, t_2\}$ satisfies φ_1 or $\{t_1, t_2\}$ satisfies φ_2,
- if $\varphi = (\varphi_1 \Rightarrow \varphi_2)$, then $\{t_1, t_2\}$ satisfies φ if and only if $\{t_1, t_2\}$ satisfies φ_2 whenever $\{t_1, t_2\}$ satisfies φ_1.

Notice that the functional dependency $\{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\}$ is satisfied by r precisely when the Boolean dependency $(A_1 \wedge \cdots \wedge A_n) \Rightarrow (B_1 \wedge \cdots \wedge B_m)$ is satisfied by r. Hence, FDs are special cases of BDs. Notice, however, that MVDs are not BDs.

Let $\mathcal{C}$ denote a class of dependencies, $\Sigma \cup \{\varphi\}$ be a set of dependencies from $\mathcal{C}$ all defined on the same relation schema R. We say that Σ (finitely) implies φ, denoted by $\Sigma \models \varphi$ ($\Sigma \models_{\text{fin}} \varphi$) if and only if all (finite) $r \subseteq dom(N)$ that satisfy all dependencies in Σ also satisfy φ. Furthermore, Σ implies φ in the world of two-element instances if and only if all $r = \{t_1, t_2\} \subseteq dom(N)$ that satisfy all dependencies in Σ also satisfy φ. In this paper we deal with classes $\mathcal{C}$ of data dependencies for which finite and unrestricted implication coincide. The implication problem for a class $\mathcal{C}$ of dependencies is to decide whether for an arbitrary relation schema R and an arbitrary set $\Sigma \cup \{\varphi\}$ of dependencies on R from $\mathcal{C}$ the following holds: $\Sigma \models \varphi$.

2.2 Correspondences to Propositional Logic

We assume familiarity with basic notions from classical Boolean propositional logic (Enderton 2001). We will be particularly interested in propositional formulae of the form

$$(U_1 \wedge \cdots \wedge U_n) \Rightarrow (V_1 \wedge \cdots \wedge V_m)$$

and

$$(U_1 \wedge \cdots \wedge U_n) \Rightarrow ((V_1 \wedge \cdots \wedge V_m) \vee (W_1 \wedge \cdots \wedge W_k))$$

where U_i, V_j, W_l denote propositional variables. We assume that the conjunction of 0 propositional variables is *true*, that negation $\neg$ binds stronger than any other Boolean connectives, and that conjunction $\wedge$ and disjunction $\vee$ bind stronger than implication $\Rightarrow$. The formulae above become $U_1 \wedge \cdots \wedge U_n \Rightarrow V_1 \wedge \cdots \wedge V_m$ and $U_1 \wedge \cdots \wedge U_n \Rightarrow (V_1 \wedge \cdots \wedge V_m) \vee (W_1 \wedge \cdots \wedge W_k)$, respectively. The satisfaction of a propositional formula φ' by a truth assignment θ is denoted by $\models_\theta \varphi'$.

Let $\phi : R \rightarrow \mathcal{V}$ denote a bijection between the attributes of R and the set $\mathcal{V}$ of Boolean propositional variables. We will now extend this bijection to classes of data dependencies over the relation schema R and (fragments of) Boolean propositional formulae over $\mathcal{V}$.

First, consider the FD φ: $X \rightarrow Y$ on R where $X = \{X_1, \ldots, X_n\}$ and $Y = \{Y_1, \ldots, Y_k\}$. Define $\Phi(\varphi)$ to be the propositional formula

$$\varphi' = \phi(X_1) \wedge \cdots \wedge \phi(X_n) \Rightarrow \phi(Y_1) \wedge \cdots \wedge \phi(Y_k) \quad .$$

Secondly, consider the MVD φ: $X \twoheadrightarrow Y$ on R where $X = \{X_1, \ldots, X_n\}$, $Y = \{Y_1, \ldots, Y_k\}$ and $R - XY = \{Z_1, \ldots, Z_m\}$. In this case, define $\Phi(\varphi)$ to be the Boolean propositional formula

$$\varphi' = \phi(X_1) \wedge \cdots \wedge \phi(X_n) \Rightarrow (\phi(Y_1) \wedge \cdots \wedge \phi(Y_k)) \vee (\phi(Z_1) \wedge \cdots \wedge \phi(Z_m)) \quad .$$

Finally, we recursively define the mapping of Boolean dependencies φ to their equivalent Boolean propositional formulae $\Phi(\varphi) = \varphi'$. If $\varphi = A$ is an attribute of R, then let $\varphi' = \phi(A)$. The rest of the mapping is straightforward:

- for $\varphi = \neg\sigma$ we have $\varphi' = \neg\sigma'$,
- for $\varphi = (\varphi_1 \vee \varphi_2)$ we have $\varphi' = (\varphi_1' \vee \varphi_2')$,
- for $\varphi = (\varphi_1 \wedge \varphi_2)$ we have $\varphi' = (\varphi_1' \wedge \varphi_2')$, and
- for $\varphi = (\varphi_1 \Rightarrow \varphi_2)$ we have $\varphi' = (\varphi_1' \Rightarrow \varphi_2')$.

If Σ is a set of dependencies on R, then let $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$ denote the corresponding set of propositional formulae over $\mathcal{V}$.

Theorem 2.1 (Sagiv, Delobel, Stott Parker jr., Fagin). *Let $\Sigma \cup \{\varphi\}$ be a set of functional and multivalued dependencies on the relation schema R, and let $\Sigma' \cup \{\varphi'\}$ denote the set of propositional formulae that correspond to $\Sigma \cup \{\varphi\}$. Then the following are equivalent:*

- *Σ implies φ,*
- *Σ implies φ in the world of two-element instances,*
- *Σ' logically implies φ'.* □

The major proof argument shows a one-to-one correspondence between 2-tuple counterexample relations r for the implication of φ by Σ and a special truth assignment θ_r showing that φ' is not logically implied by Σ'. In fact, for $r = \{t_1, t_2\}$ and all $A \in R$ we have $t_1[A] = t_2[A]$ precisely if $\theta_r(\phi(A)) = \textit{true}$. That is, the two tuples t_1, t_2 of the counterexample relation r agree on precisely those attribute whose corresponding variable is evaluated to *true* under θ_r.

Example 2.1. *Let* FILM={Movie,Writer,Actor}, *and* Σ = {Movie $\twoheadrightarrow$ Writer}. *Furthermore, let φ be* Movie $\rightarrow$ Writer. *The 2-tuple relation r*

Movie	*Writer*	*Actor*
The Godfather	*M.Puzzo*	*M.Brando*
The Godfather	*F.F.Coppola*	*M.Brando*

shows that φ is not implied by Σ. Let us now look at the implication of φ by Σ from a logical point of view. Hence, we have the three propositional variables V_M, V_W and V_A, and $\Sigma' = \{V_M \Rightarrow (V_W \vee V_A)\}$ and $\varphi' = V_M \Rightarrow V_A$. The truth assignment θ_r with $\theta_r(V)$ = true *precisely when $V \in \{V_M, V_A\}$ satisfies $\models_{\theta_r} \Sigma'$ but not $\models_{\theta_r} \varphi'$. Notice the correspondence between r and θ_r: the two tuples agree on precisely those attributes whose corresponding variables are evaluated to* true. □

Theorem 2.1 has been extended to Boolean dependencies and propositional formulae.

Theorem 2.2 (Sagiv, Delobel, Stott Parker jr., Fagin). *Let $\Sigma \cup \{\varphi\}$ be a set of Boolean dependencies on the relation schema R, and let $\Sigma' \cup \{\varphi'\}$ denote the set of propositional formulae that correspond to $\Sigma \cup \{\varphi\}$. Then the following are equivalent:*

- *Σ implies φ,*
- *Σ implies φ in the world of two-element instances,*
- *Σ' logically implies φ'.* □

Notice that these results are not extendable to either join or embedded multivalued dependencies (Sagiv et al. 1981). We will demonstrate later on how these correspondences can be applied to support the acquisition of constraints, i.e., Boolean and multivalued dependencies.

3 The Chase

A *tableau* (Maier et al. 1979) is a two-dimensional matrix in which columns correspond to attributes. The rows of a tableau consist of variables of the following types:

1. *distinguished variables*, usually denoted by subscripted a's, and
2. *nondistinguished variables*, usually denoted by subscripted b's.

A variable cannot appear in more than one column, and in each column there is exactly one distinguished variable.

A JD $\bowtie\{X_1, \ldots, X_n\}$ has a corresponding tableau T as follows. For each X_j, tableau T has a row t_j with distinguished variables in all the X_j columns and distinct nondistinguished variables in the rest of the columns. We can also view a tableau as a relation over the domain of distinguished and nondistinguished variables. Note that rows $t_1, \ldots, t_n$ of T are joinable on $X_1, \ldots, X_n$, and the resulting row t consists only of distinguished variables. That is, $t[X_i] = t_i[X_i]$ holds for all $i = 1, \ldots, n$.

Example 3.1. *Consider the relation schema*

$$\text{Family}=\{Name, Child, Child\_DoB, Pet\}.$$

The corresponding tableau for the JD

$$\bowtie \{\{Name, Child\}, \{Name, Child\_DoB, Pet\}\}$$

looks as follows:

Name	*Child*	*Child_DoB*	*Pet*
a_N	a_C	b_1	b_2
a_N	b_3	a_D	a_P

where a_N, a_C, a_D, a_P denote the distinguished variables and b_1, b_2, b_3 denote the nondistinguished variables. □

Let Σ be a set of FDs and JDs. Each dependency in Σ has an associated rule that can be applied to any tableau T as follows.

1. *FD*-rules. An FD $X \to Y$ in Σ has an associated rule for equating variables of T as follows. Suppose that rows t_1 and t_2 of T agree in all X-columns but disagreee in an A-column, where A is an attribute in Y. If one of t_1 and t_2 has a distinguished variable in its A-column, then rename the two rows so that t_1 is that row. The FD-rule for $X \to Y$ replaces all occurrences of the variable appearing in the A-column of t_2 with the variable appearing in the A-column of t_1.
2. *JD*-rules. A JD $\bowtie\{Y_1, \ldots, Y_m\}$ in Σ has an associated rule for adding rows to T as follows. If rows $t_1, \ldots, t_m$ of T are joinable on $Y_1, \ldots, Y_m$ with a result t and t is not already in T, then t is added to T.

Each one of the above rules transforms a tableau T to another tableau T'. The rules can be applied repeatedly to a tableau T only a finite number of times, and the result is unique (up to renaming of nondistinguished variables) (Maier et al. 1979). The *chase* of T by Σ, denoted by $chase_\Sigma(T)$, is the tableau obtained by applying the rules associated with Σ to T until no rule can be applied anymore. Let φ be a JD with a corresponding tableau T_φ. The JD φ is a consequence of Σ if and only if $chase_\Sigma(T_\varphi)$ contains a row consisting of distinguished variables (Maier et al. 1979).

Example 3.2. *Consider the relation schema*

$$\text{Family}=\{Name, Child, Child\_DoB, Pet\}.$$

Let Σ consist of the JD

$$\bowtie\{\{\text{Name}, \text{Child}, \text{Child\_DoB}\}, \{\text{Name}, \text{Pet}\}\}$$

and the FD

$$\text{Child} \to \text{Child\_DoB}.$$

Suppose we want to decide whether the JD φ

$$\bowtie\{\{\text{Name}, \text{Child}\}, \{\text{Name}, \text{Child\_DoB}, \text{Pet}\}\}$$

is implied by Σ. As in Example 3.1 the tableau T_φ is

Name	*Child*	*Child_DoB*	*Pet*
a_N	a_C	b_1	b_2
a_N	b_3	a_D	a_P

The JD*-rule can be applied to T_φ to generate the following template:*

Name	*Child*	*Child_DoB*	*Pet*
a_N	a_C	b_1	b_2
a_N	b_3	a_D	a_P
a_N	a_C	b_1	a_P
a_N	b_3	a_D	b_2

which equates to $\text{chase}_\Sigma(T_\varphi)$*. It follows that φ is not implied by Σ as* $\text{chase}_\Sigma(T_\varphi)$ *does not contain the row (a_N, a_C, a_D, a_P). In fact,* $\text{chase}_\Sigma(T_\varphi)$ *is a counterexample relation for the implication of φ by Σ. After some interaction this relation can be better presented as follows:*

Name	*Child*	*Child_DoB*	*Pet*
Bundy	*Kelly*	*15.05.1982*	*Peggy*
Bundy	*Bud*	*26.02.1977*	*Buck*
Bundy	*Kelly*	*15.05.1982*	*Buck*
Bundy	*Bud*	*26.02.1977*	*Peggy*

One can easily observe that the set of Child*-values are not determined by the* Name*-value independently of the* Pet*-value and the* Child_DoB*-value. Since the sample database appears to be realistic the JD φ does not need to be specified.* □

The FD $X \to A$ has the following corresponding tableau T_X which has two rows: t_d consists of distinguished variables only, and t_X has distinguished variables in the X-columns and distinct nondistinguished variables elsewhere. The FD $X \to A$ is a consequence of Σ if and only if $chase_\Sigma(T_X)$ has only distinguished variables in the A-column.

Example 3.3. *Consider the relation schema*

$$\text{Family}=\{Name, Child, Child\_DoB, Pet\}.$$

Let Σ consist of the JD

$$\bowtie\{\{\text{Name}, \text{Child}, \text{Child\_DoB}\}, \{\text{Name}, \text{Pet}\}\}$$

and the FDs

$$\text{Child} \rightarrow \text{Child\_DoB} \text{ and } \text{Child} \rightarrow \text{Pet}.$$

Suppose we want to decide whether the FD φ

$$\text{Name} \rightarrow \text{Pet}$$

is implied by Σ. Hence, we start with the template T_{Name}.

Name	*Child*	*Child_DoB*	*Pet*
a_N	a_C	a_D	a_P
a_N	b_1	b_2	b_3

An application of the JD*-rule leads to the template*

Name	*Child*	*Child_DoB*	*Pet*
a_N	a_C	a_D	a_P
a_N	b_1	b_2	b_3
a_N	a_C	a_D	b_3
a_N	b_1	b_2	a_P

The FD*-rule can be applied twice to this template and results in the final template* $\text{chase}_\Sigma(T_{\text{Name}})$.

Name	*Child*	*Child_DoB*	*Pet*
a_N	a_C	a_D	a_P
a_N	b_1	b_2	a_P

Since $\text{chase}_\Sigma(T_{\text{Name}})$ *has only the distinguished variable a_P in the* Pet*-column it follows that φ is indeed implied by Σ. Consequently, φ is already implicitely specified by Σ.* □

A *Chase*-counterexample suffices to reject φ if this counterexample is acceptable as part of a future database instance. However, a single *Chase* counterexample may not justify the specification of φ.

4 Propositional Tableaux

We will summarise the extremely elegant and efficient proof procedure for propositional logic that is known as *analytic tableaux*. The technique goes back to Beth (Beth 1959), Hintikka (Hintikka 1955) and its idea derives from Gentzen (Gentzen 1935). For a comprehensive view on first-order logic from an analytical tableaux point of view we recommend (Smullyan 1995).

For conveniance we introduce symbols "T" and "F" to our object language, and define *signed* formulae as an expression TX or FX where X is an unsigned propositional formula. Informally, we read TX as "X is true" and FX as "X is false". The truth value of TX is the same as that of X; the truth value of FX is the same as that of $\neg X$. By the *conjugate* of a signed formula we mean the result of changing "T" to "F" or "F" to "T" (thus the conjugate of TX is FX; the conjugate of FX is TX).

We now summarise the rules that applied to the formulae in a tableau and expand the tableau at a leaf. The conclusions of a rule are appended either vertically or horizontally at a leaf whenever the premise of the exansion rule matches a formula that appears anywhere on the path from the root to that leaf. We distinguish between an α-expansion and a β-expansion:

$$\begin{array}{c}\alpha\\ |\\ \alpha_1\\ \alpha_2\end{array}$$

(α-expansion)

$$\begin{array}{ccc}&\beta&\\ &/\ \backslash&\\ \beta_1 & & \beta_2\end{array}$$

(β-expansion)

In an α-expansion both conclusions α_1 and α_2 are appended on top of one another, whereas in a β-expansion the tree branches into β_1 and β_2. Table 2 shows the different signed formulae for an α-expansion. We note that under any truth assignment,

α	α_1	α_2
$T(X \wedge Y)$	TX	TY
$F(X \vee Y)$	FX	FY
$F(X \Rightarrow Y)$	TX	FY
$T(\neg X)$	FX	FX
$F(\neg X)$	TX	TX

Table 2: α-expansion

α is *true* precisely if α_1 and α_2 are both *true*. Table 3 shows the different signed formulae for a β-expansion. Under any truth assignment, β is *true* precisely if at

β	β_1	β_2
$F(X \wedge Y)$	FX	FY
$T(X \vee Y)$	TX	TY
$T(X \Rightarrow Y)$	FX	TY

Table 3: β-expansion

least one of the pair β_1 and β_2 is *true*.

An *analytical tableau* is a marked (by signed propositional formulae), unordered, finite tree that is inductively defined as follows:

1. The tree consisting of a single path

$$\begin{array}{c}F_1\\ F_2\\ \vdots\\ F_n\end{array}$$

   is a tableau for $\{F_1, \ldots, F_n\}$.

2. If T is a tableau for $\{F_1, \ldots, F_n\}$, and T' results from T by applying an expansion rule, then T' is also a tableau for $\{F_1, \ldots, F_n\}$.

A *branch* of a tableau is a maximal path, i.e., a path from the root to a leaf. A branch is said to be *closed* if it contains a signed variable and its conjugate. A tableau is called *closed* precisely when all its branches are closed. A branch ϑ of a tableau is *complete* if for every α which occurs in ϑ, both α_1 and α_2 occur in ϑ, and for every β which occurs in ϑ, at least one of β_1, β_2 occurs in ϑ. We call a tableau *completed* if every branch is either closed or completed.

Theorem 4.1. *Every complete open branch of a tableau for $\{T\sigma \ : \ \sigma \in \Sigma\} \cup \{F\varphi\}$ defines a truth assignment θ that satisfies Σ and violates φ.* □

We will now summarise the constructive proof of Theorem 4.1 (for a full proof see for instance (Smullyan 1995)) which has interesting consequences for us. Let S be the set of signed formulae in a complete open branch ϑ of a tableau. Then the set satisfies the following three conditions for every α and β:

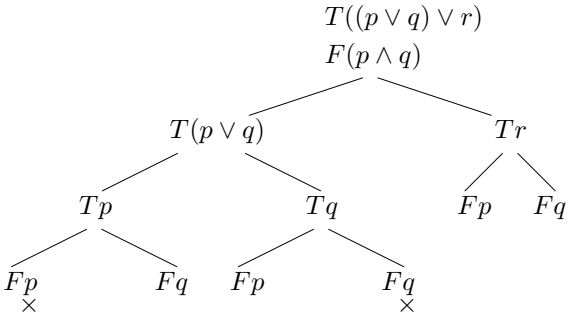
1. No signed variable and its conjugate are both in S.
2. If $\alpha \in S$, then $\alpha_1 \in S$ and $\alpha_2 \in S$.
3. If $\beta \in S$, then $\beta_1 \in S$ or $\beta_2 \in S$.

Sets S obeying these three conditions are known as *Hintikka sets* which are satisfiable. In fact, we can define a truth assignment that makes every element of S true. Therefore, we assign each variable p, which occurs in at least one element of S, a truth value as follows:

1. If $Tp \in S$, then give p the value *true*.
2. If $Fp \in S$, then give p the value *false*.
3. If neither Tp nor Fp is an element in S, then give p the value *true* or *false* at will.

It can be shown that every element of S is true under this truth assignment.

Example 4.1. *Suppose we want to refute* $T((p \vee q) \vee r)$ *and* $F(p \wedge q)$. *Then we obtain the following tableau:*



The left-most open branch contains Tp *and* Fq *but no signed variable for* r. *Hence, the truth value of* r *can be chosen at will, i.e., we obtain two truth assignments* $\theta_1(r) = \text{true}$, $\theta_2(r) = \text{false}$ *and* $\theta_1(p) = \theta_2(p) = \text{true}$ *and* $\theta_1(q) = \theta_2(q) = \text{false}$. □

Completed tableaux do not only allow us to read off a single truth assignment that satisfies Σ and violates φ, but *all* models of $\Sigma \cup \{\neg\varphi\}$.

Example 4.2. *Consider the complete tableau from Example 4.1. The following truth assignments can be read off the tableau.*

p	q	r
true	*false*	*true*
true	*false*	*false*
false	*true*	*true*
false	*true*	*false*
false	*false*	*true*

These form the set of all truth assignments that satisfy $(p \vee q) \vee r$, *but violate* $p \wedge q$. □

5 Assisting Constraint Acquisition

We will use this section to describe how analytical tableaux can be utilised in the constraint acquisition process. Suppose that so far we have gathered a certain set Σ of those constraints that should be specified (initially $\Sigma = \emptyset$). Some user or designer may suggest that another constraint φ captures important semantic information about the database. That is, φ becomes the new candidate constraint. Using the correspondences to logic from Subsection 2.2 we translate $\Sigma \cup \{\varphi\}$ into its corresponding set $\Sigma' \cup \{\varphi'\}$ of propositional formulae. Subsequently, we attempt to refute $\{T(\sigma') \mid \sigma' \in \Sigma'\} \cup \{F(\varphi')\}$. If our completed tableau is closed we can conclude that φ' is already implicitely specified by Σ'. Utilising the correspondences once again we can conclude that φ is implied by Σ. Consequently, φ does not need to be specified. In the other case our completed tableau is open and φ' is not implied by Σ'. In this case we read off *all* truth assignments that satisfy Σ' and violate φ'. We will then use every truth assignment θ to generate a 2-tuple counterexample relation r_θ for the implication of φ by Σ. The first tuple of r_θ consists of 1's only. Let V_A denote the propositional variable that corresponds to the attribute A, i.e., $\phi(A) = V_A$. The second tuple has the truth value of V_A as entry in column A. That is, the two tuples of r_θ agree on precisely those attributes whose corresponding variables are evaluated to true by θ. At this stage the designer may substitute the 1's and 0's by actual values from the domains of the attributes, accordingly. Note that this does not result in any problems as any reasonable attribute can be assumed to have at least two distinct values.

Example 5.1. *Reconsider Example 1.2. The only open branch of that tableau defines the truth assignment* $\theta(V_M) = 1$, $\theta(V_W) = 0$ *and* $\theta(V_A) = 1$. *The 2-tuple counterexample relation* θ_r *is*

Movie	*Writer*	*Actor*
1	*1*	*1*
1	*0*	*1*

The first tuple consists of 1*s only while the second tuple essentially stores the truth values from the corresponding variables. The designer may decide to substitute* The Godfather *for the* 1 *in the* Movie*-column,* M.Puzzo *for the* 1 *and* F.F.Coppola *for the* 0 *in the* Writer*-column, and* M.Brando *for the* 1 *in the* Actor*-column. Thus, we obtain*

Movie	*Writer*	*Actor*
The Godfather	*M.Puzzo*	*M.Brando*
The Godfather	*F.F.Coppola*	*M.Brando*

In this way we can generate 2-tuple relations from the special truth assignments. □

After this process the participants of the acquisition process inspect all of the relations. Each of these relations conforms to the rules represented by Σ and violates the rule represented by φ. If there is at least one such relation that is acceptable, then the specification of φ is too restrictive for the future database. Consequently, we do not specify φ. In the remaining case we cannot find any acceptable 2-tuple violation of φ. This means that we cannot find any acceptable counterexample relation at all because any such relation would include at least one unacceptable 2-tuple counterexample subrelation. That means, that every relation that satisfies Σ and violates φ is unacceptable to us. In other words, it is unacceptable to satisfy Σ without satisfying φ. Since φ is not implied by Σ we must add φ to the set of specified constraints.

Our techniques may become even more powerful when integrated into more general approaches. For instance, the work in (Albrecht, Buchholz, Düsterhöft & Thalheim 1995) describes a general framework for constraint acquisition combining several techniques such as natural language processing, dialogues, heuristics and sample data. However, in their approach sample data is not generated automatically but entered by the designer. As demonstrated tableaux techniques can help generating all typical sample data such that candidate constraints cannot only be rejected but also be accepted.

6 An Example

In this section we will present an example that demonstrates the support of our technique for acquiring constraints.

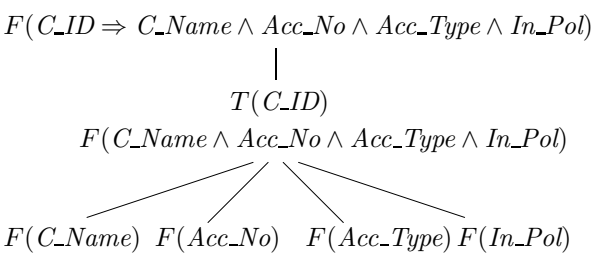
Assume we are given a relation schema BANK with the following attributes: *C_ID*, *C_Name*, *Acc_No*, *Acc_Type*, and *In_Pol*. The intention of the schema is the following: customers have an id and a name, bank accounts with a no and a certain type, and insurances with an insurance policy.

In the constraint acquistion process the first task might be to determine a key for BANK, i.e., to determine those attributes whose values suffice to identify rows in a database instance.

A first guess might be that *C_ID* forms a reasonable key. Hence, we consider the candidate FD

$$C\_ID \rightarrow C\_Name, Acc\_No, Acc\_Type, In\_Pol.$$

The tableau for the corresponding formulae looks as follows:



Instead of reading off truth assignments for all the variables we only look at truth assignments defined by those variables that actually appear in a branch. This is justified as we might replace the FD above by 4 separate FDs each with a single attribute on the right. Consequently, after some intervention by the designer we derive the following counterexamples:

C_ID	C_Name
001	Sylvester
001	Tweety

C_ID	Acc_No
001	0553-0331410-38
001	0553-0331410-40

C_ID	Acc_Type
001	Cheque
001	Savings

C_ID	In_Pol
001	House
001	Contents

The table in the top left illustrates an instance where the same customer id is associated with different customer names. This is unacceptable since customers can be distinguished by their id. Since the example above is essentially the only counterexample for the validity of $C\_ID \rightarrow C\_Name$ (up to renaming) but this counterexample is unacceptable we specify this FD.

The table in the top right illustrates the case where two account numbers are associated with the same customer id, i.e., the same customer may have different accounts. As this occurs in practice all the time, this counterexample is acceptable. Accordingly, we do not specify the FD $C\_ID \rightarrow Acc\_No$.

Similar situations are illustrated in the bottom tables. The same customer may have different account types, and different insurance policies. Hence, we do not specify neither $C\_ID \rightarrow Acc\_Type$ nor $C\_ID \rightarrow In\_Pol$.

In particular, we do not specify $\{C\_ID\}$ as a key.

We may now proceed by checking the suitability of all remaining attributes of BANK as a unary key. In a similar way that was described for C_ID it turns out that no attribute is suitable. When checking *Acc_No* we can derive, in particular, the following counterexample relation:

Acc_No	Acc_Type
0553-0331410-38	Cheque
0553-0331410-38	Savings

In this instance different account types are associated with the same account. This is unacceptable, and we therefore decide to specify the FD $Acc\_No \rightarrow Acc\_Type$.

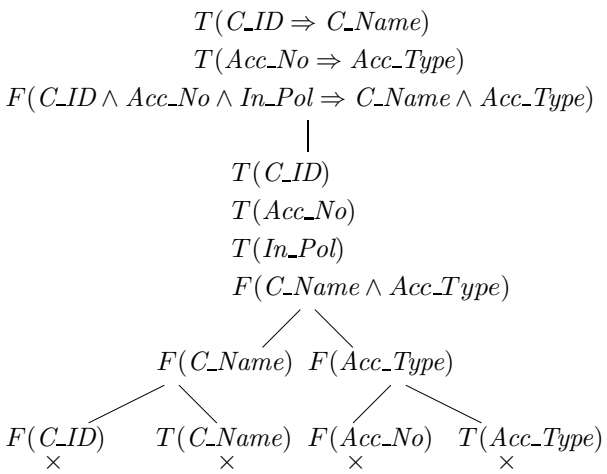
From this we might conclude that $\{C\_ID, Acc\_No, In\_Pol\}$ forms a reasonable minimal key. In order to support this assumption we can try to decide whether the FD

$$C\_ID, Acc\_No, In\_Pol \rightarrow C\_Name, Acc\_Type$$

should be specified on top of

$$\Sigma = \{C\_ID \rightarrow C\_Name, Acc\_No \rightarrow Acc\_Type\}.$$

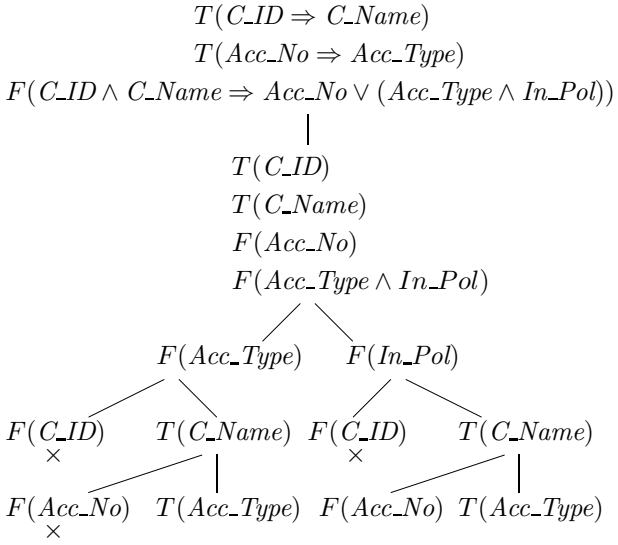
We utilise the following tableau to read off any potential counterexamples.



Indeed, the tableau is closed which confirms our assumption that the key is already implied by Σ. Earlier we had detected that the customer information does not uniquely identify the account information since a customer may have multiple accounts. However, for each customer her/his accounts are independent from the insurance policies. As a first approximation one may ask whether the MVD

$$C\_ID, C\_Type \twoheadrightarrow Acc\_No$$

provides a useful semantic constraint together with those that have already been specified. Hence, we obtain the following tableau.



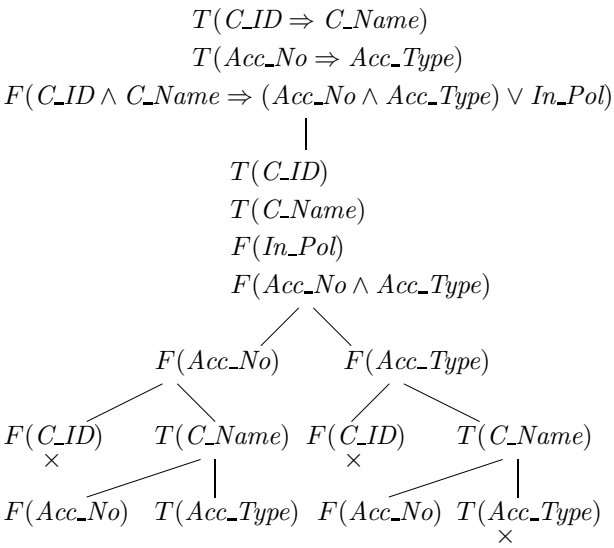
The open branches of this tableau allow us to read off the following counterexample relations

C_ID	C_Name	Acc_No	Acc_Type	In_Pol
001	Sylvester	0331410	Cheque	Life
001	Sylvester	1240501	Savings	House

C_ID	C_Name	Acc_No	Acc_Type	In_Pol
001	Sylvester	0331410	Cheque	Life
001	Sylvester	1240501	Cheque	House

C_ID	C_Name	Acc_No	Acc_Type	In_Pol
001	Sylvester	0331410	Cheque	Life
001	Sylvester	1240501	Savings	Life

The first two relations from the top seem unacceptable: the customer has not only two different accounts, but also two different policies. However, the bottom relation is completely acceptable: the customer has two different accounts, each of a different type, and only one insurance policy. This shows that the MVD above does not fully capture that the account information is independent of the insurance policies. Hence, we decide not to specify the MVD above. However, the account type is linked to the account no, i.e., both of them cannot be independent of one another. Consequently, we ask now whether the MVD $C\_ID, C\_Type \twoheadrightarrow Acc\_No, Acc\_Type$ represents useful semantic information.



The open branches of the tableau define the following counterexample relations:

C_ID	C_Name	Acc_No	Acc_Type	In_Pol
001	Sylvester	0331410	Cheque	Life
001	Sylvester	1240501	Savings	House

C_ID	C_Name	Acc_No	Acc_Type	In_Pol
001	Sylvester	0331410	Cheque	Life
001	Sylvester	1240501	Cheque	House

Both of them are unacceptable. Hence, there is no counterexample relation that is acceptable. Consequently, we specify the MVD above. At this point we have acquired the following constraint set Σ

- $C\_ID \rightarrow C\_Name$
- $Acc\_No \rightarrow Acc\_Type$
- $C\_ID, C\_Type \twoheadrightarrow Acc\_No, Acc\_Type$.

Based on Σ we may derive the following faithful 4NF-decomposition of BANK:

- C_INFO={C_ID,C_Name} with key {C_ID},
- C_ACCOUNT={C_ID,Acc_No} with key C_ACCOUNT,
- A_INFO={Acc_No,Acc_Type} with key {Acc_No}, and
- C_INSURANCE={C_ID,In_Pol} with key C_INSURANCE.

7 Conclusion

Knowledge about the semantics of the database is crucial for its quality. Semantic constraints form the basis for database normalisation, restructuring operations and query optimisation. The acquistion of semantic constraints is a difficult task for many database designers.

We have identified two different tableaux techniques that can be utilised for rejecting and accepting candidate constraints. So far, the Chase has been primarily used for query optimisation and deciding the implication of certain classes of dependencies. However, it is also useful for generating sample data. If the participants of the acquisition process agree that the violation of a candidate constraint is acceptable, then the candidate can be rejected.

Boolean and multivalued dependencies are semantic constraints that are very common in database practice. Their semantic implication is equivalent to the logical implication of propositional formulae fragments. Based on these correspondences we have demonstrated how analytical tableaux can be used to generate all minimal sample relations that satisfy all the constraints acquired so far but violate the current candidate. If a single of these sample relations is acceptable as part of any future database instance, then the candidate must be rejected. If, otherwise, it is always unacceptable to violate the candidate whenever the other constraints are satisfied, then the candidate should be specified.

It is future work to combine our techniques with already existing tools (Albrecht et al. 1995), and to study the exact time-complexities for generating our sample data with respect to certain classes of dependencies and certain tableaux formalisms.

References

Albrecht, M., Buchholz, E., Düsterhöft, A. & Thalheim, B. (1995), An informal and efficient approach for obtaining semantic constraints using sample data and natural language processing., *in* 'Semantics in Databases', Vol. 1358 of *LNCS*, Springer, pp. 1–28.

Arenas, M. & Libkin, L. (2004), 'A normal form for XML documents', *Trans. Database Syst.* **29**(1), 195–232.

Armstrong, W. W. (1974), 'Dependency structures of database relationships', *Information Processing* **74**, 580–583.

Armstrong, W. W., Nakamura, Y. & Rudnicki, P. (2002), 'Armstrong's axioms', *Journal of formalized Mathematics* **14**.

Beeri, C. & Bernstein, P. A. (1979), 'Computational problems related to the design of normal form relational schemata', *ACM Trans. Database Syst.* **4**(1), 30–59.

Beeri, C., Dowd, M., Fagin, R. & Statman, R. (1984), 'On the structure of armstrong relations for functional dependencies', *J. ACM* **31**(1), 30–46.

Beeri, C. & Vardi, M. (1984), 'A proof procedure for data dependencies', *J. ACM* **31**(4), 718–741.

Bernstein, P. (1976), 'Synthesizing third normal form relations from functional dependencies', *Trans. Database Syst.* **1**(4), 277–298.

Bernstein, P. A. & Goodman, N. (1980), What does Boyce-Codd normal form do?, *in* 'Proceedings of the 6th International Conference on Very Large Data Bases', IEEE Computer Society, pp. 245–259.

Beth, E. (1959), *The Foundations of Mathematics*, North Holland.

Biskup, J., Dayal, U. & Bernstein, P. (1979), Synthesizing independent database schemas, *in* 'SIGMOD Conference', pp. 143–151.

Codd, E. F. (1970), 'A relational model of data for large shared data banks', *Commun. ACM* **13**(6), 377–387.

Codd, E. F. (1972), Further normalization of the database relational model, *in* 'Courant Computer Science Symposia 6: Data Base Systems', Prentice-Hall, pp. 33–64.

De Marchia, F. & Petit, J.-M. (2007), 'Semantic sampling of existing databases through informative armstrong databases', *Inf. Syst.* **32**(3), 446–457.

Delobel, C. (1978), 'Normalisation and hierarchical dependencies in the relational data model', *Trans. Database Syst.* **3**(3), 201–222.

Demetrovics, J. & Gyepesi, G. (1983), 'A note on minimal matrix rpresentation of closure operations', *Combinatorica* **2**, 177–179.

Enderton, H. (2001), *A mathematical introduction to logic: Second Edition*, Academic Press.

Fagin, R. (1977), 'Multivalued dependencies and a new normal form for relational databases', *Trans. Database Syst.* **2**(3), 262–278.

Fagin, R. (1982*a*), Armstrong databases, Technical Report RJ3440(40926), IBM Research Laboratory, San Jose, California, USA.

Fagin, R. (1982*b*), 'Horn clauses and database dependencies', *J. ACM* **29**(4), 952–985.

Fagin, R. & Vardi, M. (1983), 'Armstrong databases for functional and inclusion dependencies', *Inf. Process. Lett.* **16**(1), 13–19.

Gentzen, G. (1935), 'Untersuchungen über das logische Schließen', *Mathematische Zeitschrift* **39**, 176–210,405–431.

Gottlob, G. & Libkin, L. (1990), 'Investigation on armstrong relations, dependency inference, and excluded functional dependencies', *Acta Cybern.* **9**(4), 385–402.

Grant, J. & Jacobs, B. (1982), 'On the family of generalized dependency constraints', *J. ACM* **29**(4), 986–997.

Hara, C. & Davidson, S. (1999), Reasoning about nested functional dependencies, *in* 'Proceedings of the 18th SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', ACM, pp. 91–100.

Hartmann, S. & Link, S. (2006), 'On a problem of Fagin concerning multivalued dependencies in relational databases', *Theor. Comput. Sci.* **353**(1-3), 53–62.

Hintikka, K. (1955), 'Form and content in quantification theory', *Acta Philosophica Fenica* **8**, 7–55.

Katona, G. & Tichler, K. (2006), Some contributions to the minimum representation problem of key systems, *in* 'FoIKS', Vol. 3861 of *LNCS*, Springer, pp. 240–257.

Khardon, R., Mannila, H. & Roth, D. (1999), 'Reasoning with examples: Propositional formulae and database dependencies', *Acta Inf.* **36**(4), 267–286.

Levene, M. & Loizou, G. (1998), 'Axiomatisation of functional dependencies in incomplete relations', *Theor. Comput. Sci.* **206**(1-2), 283–300.

Maier, D., Mendelzon, A. & Sagiv, Y. (1979), 'Testing implications of data dependencies', *ACM Trans. Database Syst.* **4**(4), 455–469.

Maier, D., Sagiv, Y. & Yannakakis, M. (1981), 'On the complexity of testing implications of functional and join dependencies', *J. ACM* **28**(4), 680–695.

Mannila, H. & Räihä, K.-J. (1985), Test data for relational queries, *in* 'Proceedings of the 5th SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', ACM, pp. 217–223.

Mannila, H. & Räihä, K.-J. (1986), 'Design by example: An application of armstrong relations', *J. Comput. Syst. Sci.* **33**(2), 126–141.

Popa, L., Deutsch, A., Sahuguet, A. & Tannen, V. (2000), A chase too far?, *in* 'SIGMOD Conference', ACM, pp. 273–284.

Sadri, F. & Ullman, J. (1982), 'Template dependencies: A large class of dependencies in relational databases and its complete axiomatization', *J. ACM* **29**(2), 363–372.

Sagiv, Y., Delobel, C., Parker Jr., D. S. & Fagin, R. (1981), 'An equivalence between relational database dependencies and a fragment of propositional logic', *J. ACM* **28**(3), 435–453.

Sagiv, Y., Delobel, C., Parker Jr., D. S. & Fagin, R. (1987), 'Correction to "an equivalence between relational database dependencies and a fragment of propositional logic"', *J. ACM* **34**(4), 1016–1018.

Sciore, E. (1982), 'A complete axiomatization of full join dependencies', *J. ACM* **29**(2), 373–393.

Smullyan, R. (1995), *First-order Logic*, Dover Publications.

Tari, Z., Stokes, J. & Spaccapietra, S. (1997), 'Object normal forms and dependency constraints for object-oriented schemata', *Trans. Database Syst.* **22**, 513–569.

Weddell, G. (1992), 'Reasoning about functional dependencies generalized for semantic data models', *Trans. Database Syst.* **17**(1), 32–64.

Wijsen, J. (1999), 'Temporal FDs on complex objects', *Trans. Database Syst.* **24**(1), 127–176.

Zaniolo, C. (1976), Analysis and Design of Relational Schemata for Database Systems, PhD thesis, UCLA, Tech. Rep. UCLA-ENG-7769.