THE THEORY OF FUNCTIONAL AND SUBSET DEPENDENCIES OVER RELATIONAL EXPRESSIONS

Marco A. CASANOVA
Pontificia Universidade Catolica do RJ, 22453 Rio de Janeiro, RJ, Brasil

Communicated by M. Paul
Received 22 June 1981
Revised 1 June 1982

A formal system for reasoning about functional dependencies (FDs) and subset dependencies (SDs) defined over relational expressions is described. An FD e : X → Y indicates that Y is functionally dependent on X in the relation denoted by expression e; an SD e ⊂ f indicates that the relation denoted by e is a subset of that denoted by f. The system is shown to be sound and complete by resorting to the analytic tableaux method. Applications of the system include the problem of determining if a constraint of a subschema is implied by the constraints of the base schema and the development of database design methodologies similar to normalization.

Keywords: Formal languages, relational expressions, functional dependencies

1. Introduction

We describe in this paper a formal system S for reasoning about functional dependencies (FDs) and subset dependencies (SDs) defined over relational expressions. The class of FDs considered consists of statements of the form e : X → Y which assert that the Y-columns are functionally dependent on the X-columns in the relation defined by the relational expression e. Likewise, SDs are statements of the form e ⊂ f specifying that the relation defined by e is a subset of that defined by f.

The development of system S was motivated primarily by the subschema constraint problem, viz., whether a constraint of a subschema σ' is valid in any state of σ' constructed from a consistent state of the base schema σ. For example, suppose that σ' has relation names r₁, ..., rₙ defined by expressions e₁, ..., eₙ (only involving relation names of σ). Let rᵢ : X → Y be an FD of σ'. Then, rᵢ : X → Y is valid in any state of σ' constructed from a consistent state of σ via e₁, ..., eₙ iff cᵢ : X → Y is a logical consequence of the constraints of σ. This last assertion can then be resolved in S.

The subschema constraint problem was studied in [11,12,14], but not for the class of FDs and SDs over expressions. Formal systems for several classes of dependencies over relations were studied, e.g., in [1,10,3,21,15]. Other examples of the use of tableaux can be found in [20,16,18].

This paper is organized as follows. Section 2 defines FDs and SDs over relational expressions. Section 3 describes the formal system S and the analytic tableaux method. Section 4 discusses the soundness and the completeness of S. Finally, Section 5 discusses on-going research.

2. Functional and subset dependency languages

This section defines a family of formal languages that we call functional and subset dependency languages (FD-SD languages). An FD-SD language ℒ contains the following symbols:

1. relation names: for each n > 0, a nonempty set of n-ary relation names,
(2) constant symbols: a nonempty set of symbols, distinct from the above,

(3) the usual connectives and special symbols: \(\neg, \land, \lor, \Rightarrow, [ , ], \to, \in, =,\)

(4) the usual relation operators: \(\times, \cup, \setminus\).

An n-ary relational expression of \(\mathcal{L}\) is defined inductively as follows. Let \(\text{ATTR}(n)\) denote the set of sequences of distinct integers from the interval \([1, n]\):

(1) an n-ary relation name is an n-ary (atomic) expression;

(2) if \(e\) is an n-ary expression, \(T, U, V, X \in \text{ATTR}(n)\) and \(Z\) is a tuple of constants such that \(\|U\| = \|V\|\) and \(\|X\| = \|a\|\), then the projection \(e[T]\) is an \(|T|\)-ary expression and the restriction \(e[U = V]\) and selection \(e[X = a]\) are n-ary expressions;

(3) if \(e\) and \(f\) are m-ary and n-ary expressions, respectively, then the product \((e \times f)\) is an \((n + m)\)-ary expression and, if \(n = m\), the union \((e \cup f)\) and difference \((e - f)\) are n-ary expressions.

An atomic formula of \(\mathcal{L}\) is either \(Z = b, e(a)\), a functional dependency \(e : X \to Y\) or a subset dependency \(e \subset f\), where \(a, b\) are n-ary tuples of constants, \(e, f\) are n-ary expressions and \(X, Y \in \text{ATTR}(n)\), \(n > 0\). A well-formed formula (wff) of \(\mathcal{L}\) is either an atomic formula or of the form \(\neg P, (P \land Q), (P \lor Q)\) or \((P \Rightarrow Q)\) where \(P, Q\) are wffs.

A structure \(I\) with domain \(D_1\) for \(\mathcal{L}\) is a function assigning to each n-ary relation name \(r\) of \(\mathcal{L}\) an n-ary relation \(I(r) \subseteq D_1^n\), \(n > 0\), and to each k-ary tuple of constants \(\bar{a}\), a k-ary tuple \(I(\bar{a}) \subseteq D_1^k\), \(k > 0\). I is extended to the relational expressions of \(\mathcal{L}\) as follows (note that I is already defined for the atomic expressions):

\[
\begin{align*}
(1) & \quad I(e[T]) = \{\bar{a} \mid \bar{a} \in I(e)\}, \\
(2) & \quad I(e[U = V]) = \{\bar{a} \mid \bar{a} \in I(e) \land \bar{a}_U = \bar{a}_V\}, \\
(3) & \quad I(e[X = \bar{a}]) = \{\bar{a} \mid \bar{a} \in I(e) \land \bar{a}_X = I(\bar{a})\}, \\
(4) & \quad I(e \times f) = \{\bar{a}\bar{b} \mid \bar{a} \in I(e) \land \bar{b} \in I(f)\}, \\
(5) & \quad I(e \cup f) = \{\bar{a} \mid \bar{a} \in I(e) \lor \bar{a} \in I(f)\}, \\
(6) & \quad I(e - f) = \{\bar{a} \mid \bar{a} \in I(e) \land \neg \bar{a} \in I(f)\}.
\end{align*}
\]

We now extend I to a Boolean valuation of the wffs of \(\mathcal{L}\) as follows:

\[
\begin{align*}
(1) & \quad I(\bar{a} = \bar{b}) = \text{true} \text{ if } I(\bar{a}) = I(\bar{b}), \text{ otherwise } I(\bar{a} = \bar{b}) = \text{false}; \\
(2) & \quad I(e(\bar{a})) = \text{true} \text{ if } I(\bar{a}) \in I(e), \text{ otherwise } I(e(\bar{a})) = \text{false}; \\
(3) & \quad I(e : X \to Y) = \text{true} \quad \forall \bar{a} \forall \bar{b} \quad (\bar{a} \in I(e) \land \bar{b} \in I(e) \land \bar{a}_X = \bar{b}_X \Rightarrow \bar{a}_Y = \bar{b}_Y)
\end{align*}
\]

3. A formal system for reasoning about functional and subset dependencies

Let \(\mathcal{L}\) be an FD-SD language. We introduce in this section a formal system \(S\), whose language is \(\mathcal{L}\), and a proof procedure for \(S\) such that a wff \(P\) of \(\mathcal{L}\) is logically implied by a set \(\varrho\) of wffs of \(\mathcal{L}\) iff \(P\) is a theorem of \(\varrho\) in \(S\). This result is proved in Section 4. Since the description of the rules of \(S\) depends on the proof procedure, we discuss it first.

The notion of a proof in \(S\) is a direct generalization of the analytic tableau method for Propositional Calculus [20]. It formalizes the following familiar strategy to prove that \(\varrho \vdash P\). Start with \(\varrho\) and \(\neg P\) and work out all possible cases. If every case leads to a contradiction, then \(\varrho \cup \{\neg P\}\) is unsatisfiable and, hence, \(\varrho \vdash P\). On the other hand, if the analysis of some case is exhausted without arriving at a contradiction, then \(\varrho \cup \{\neg P\}\) is satisfiable (this has to be proved, since it is not an immediate property of the system) and, hence, \(\varrho \vdash \neg P\) does not hold.

Reasoning by cases is captured by using rules of the following type:

\[
\begin{array}{c}
\varrho_1, \varrho_2, \ldots, \varrho_n, \\
\varrho_i \rightarrow \varrho_i
\end{array}
\]

where \(\varrho_i\) and \(\varrho_{ij}\) \((1 \leq j \leq n_i)\) are finite sets of wffs. Intuitively, \(\varrho_i\) means that from \(\varrho_i\) we can derive all wffs in \(\varrho_{ij}\) for some \(j \in [1, n_i]\). We call \(\varrho_i\) the antecedent of \(\varrho_i\) and \(\varrho_{11}, \ldots, \varrho_{in_i}\) the consequents of \(\varrho_i\). A proof by case analysis can be organized
as a tree, where the sons of a node correspond to branching cases. A proof terminates when each branch either contains a contradiction or cannot be extended further without repetition. These observations are formalized as follows (by a branch of a tree we mean a path from the root to a leaf).

**Definition 3.1.** (a) The set of analytic tableaux for a set \( \mathfrak{P} \) of wffs consists of trees whose nodes are sets of wffs. It is defined inductively as follows:

(i) The tree whose only node is \( \mathfrak{P} \) is an analytic tableau for \( \mathfrak{P} \);

(ii) Suppose that \( \tau \) is an analytic tableau for \( \mathfrak{P} \) and let \( \lambda \) be a leaf of \( \tau \). Then, the tree obtained by extending \( \tau \) by the following operation is also an analytic tableau for \( \mathfrak{P} \): if there is a rule \( \mathfrak{R}_i \) with antecedent \( \mathfrak{P}_i \) and consequents \( \mathfrak{P}_{i1}, \ldots, \mathfrak{P}_{in} \), such that all wffs in \( \mathfrak{P}_i \) occur in the branch ending in \( \lambda \), then \( n_i \) distinct sons \( \lambda_1, \ldots, \lambda_{n_i} \) may simultaneously be adjoined to \( \lambda \), where \( \lambda_j \in \mathfrak{P}_{ij} \) \( (1 \leq j \leq n_i) \).

(b) A set \( \mathfrak{K} \) of wffs is a Hintikka set iff no wff and its negation are in \( \mathfrak{K} \) and if there is a rule \( \mathfrak{R}_i \) with antecedent \( \mathfrak{P}_i \) and consequents \( \mathfrak{P}_{i1}, \ldots, \mathfrak{P}_{in} \), such that all wffs in \( \mathfrak{P}_i \) occur in the branch ending in \( \lambda \), then \( n_i \) distinct sons \( \lambda_1, \ldots, \lambda_{n_i} \) may simultaneously be adjoined to \( \lambda \), where \( \lambda_j \in \mathfrak{P}_{ij} \) \( (1 \leq j \leq n_i) \).

(c) A branch of a tableau is closed iff it contains a wff and its negation, otherwise it is open.

(d) A branch of a tableau is complete iff the union of all its nodes is a Hintikka set.

(e) A tableau is closed iff every branch is closed.

(f) A tableau is complete iff each branch is either closed or complete.

(g) A proof of a wff \( P \) from a set of wffs \( \mathfrak{P} \) is a closed tableau for \( \mathfrak{P} \cup \{\neg P\} \). In this case, \( P \) is a theorem of \( \mathfrak{P} \) in \( S \) (written \( \mathfrak{P} \vdash P \)).

**Product rules**

\[
\begin{align*}
-\text{PT} &. \quad -e(x \times f)(\bar{a}), e(\bar{a}_{\{1, n_1\}}), f(\bar{a}_{\{n+1, n+m\}}) \\
\text{PT} &. \quad e(x \times f)(\bar{a}), e(\bar{a}_{\{1, n_1\}}) \quad f(\bar{a}_{\{n+1, n+m\}})
\end{align*}
\]

\(\bar{a}, \bar{b}\) are any tuples of constants and \( e \) is \( n \)-ary and \( f \) is \( m \)-ary.

**Union rules**

\[
\begin{align*}
\text{-UN} &. \quad -(e \cup f)(\bar{a}), e(\bar{a}), f(\bar{a}) \\
\text{UN} &. \quad (e \cup f)(\bar{a}), e(\bar{a}) \quad f(\bar{a})
\end{align*}
\]

\(\bar{a}\) is any tuple of constants.

**Difference rules**

\[
\begin{align*}
\text{-DI} &. \quad -(e - f)(\bar{a}), e(\bar{a}) \quad f(\bar{a}) \\
\text{DI} &. \quad (e - f)(\bar{a}) \quad e(\bar{a}) \quad -f(\bar{a})
\end{align*}
\]

\(\bar{a}\) is any tuple of constants.

**Equality rules**

\[
\begin{align*}
\text{ES} &. \quad \bar{a} = \bar{b} \quad \bar{a} = \bar{b} \\
\text{ER} &. \quad \bar{a} = \bar{b} \quad \bar{a} = \bar{b} \\
\text{ET} &. \quad \bar{a} = \bar{b}, \bar{b} = \bar{c} \quad \bar{a} = \bar{c}
\end{align*}
\]

**A-rules and B-rules**

**A-rules.**

\[
\begin{align*}
\frac{A}{A_1, A_2} \\
\frac{B}{B_1 | B_2}
\end{align*}
\]

where \( A, A_1, A_2 \) and \( B, B_1, B_2 \) are given by Table 1.

Intuitively, the rules of \( S \) capture patterns of reasoning commonly used in mathematics. Rule -FD, for example, captures the following pattern: "... Assume \( \neg e : X \rightarrow Y \). Then, there must be two tuples in the value of \( e \) agreeing on \( X \), but not on \( Y \). Let \( \bar{a} \) and \( \bar{b} \) denote these tuples ..." (where \( \bar{a} \) and \( \bar{b} \) have not been previously used). Rules FD, SD and SD are similarly explained. The rules for
Table 1

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A_1</th>
<th>A_2</th>
<th>B</th>
<th>B_1</th>
<th>B_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P ∧ Q</td>
<td>P</td>
<td>Q</td>
<td>¬(P ∧ Q)</td>
<td>¬P</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>¬(P ∨ Q)</td>
<td>¬P</td>
<td>¬Q</td>
<td>P ∨ Q</td>
<td>P</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>¬(P → Q)</td>
<td>P</td>
<td>¬Q</td>
<td>P → Q</td>
<td>¬P</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>¬¬P</td>
<td>P</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

relational expressions directly reflect previous definitions. Finally, the equality, the A-rules and the B-rules are quite familiar, except for rules EP and ¬EP, which follow if we interpret \( \bar{a} = \bar{b} \) as

\[
\bigwedge_{i=1}^{n} a_i = b_i.
\]

At a more formal level the rules of S may be viewed as derived rules of first-order logic. This remark does not imply that S is superfluous, since S shortens proofs by hiding quantifiers and certain standard derivations pertaining to FDs and SDs.

As an example of our claim we derive rules ¬SD and SD (ignoring certain technical difficulties raised by relational expressions, which can be circumvented along the lines of [5; 9, Chapter 4]). Consider the definition of \( e \subseteq f \) cast as follows:

(i) \( e \subseteq f \equiv \forall a (e(a) \Rightarrow f(a)) \).

Then, (i) is equivalent to the conjunction of

(ii) \( \forall a (e \subseteq f \land e(a) \Rightarrow f(a)) \),

(iii) \( \exists a (e \neq f \Rightarrow e(\bar{a}) \land \neg f(\bar{a})) \).

By applying the usual rules of first-order logic (based on tableaux [20]), (ii) generates SD and (iii) generates ¬SD.

We now describe the rules of S. By a tuple of new constants we mean a tuple of constants that do not occur in the tableau constructed thus far. If \( t = (t_1, \ldots, t_n, t_{n+1}, \ldots, t_{n+m}) \), then \( t_{[1:n]} \) denotes \( (t_1, \ldots, t_n) \) and \( t_{[n+1:]} \) denotes \( (t_{n+1}, \ldots, t_{n+m}) \).

**FD-rules**

\[
\neg e : X \rightarrow Y
\]

\[
e(\bar{a}), e(\bar{b}), \bar{a}_X = \bar{b}_X, \neg \bar{a}_Y = \bar{b}_Y
\]

\( \bar{a}, \bar{b} \) are tuples of new constants,

\[
e(\bar{a}), e(\bar{b}), \bar{a}_X = \bar{b}_X, e : X \rightarrow Y
\]

\[
\bar{a}_Y = \bar{b}_Y
\]

\( \bar{a}, \bar{b} \) are any tuples of constants.

**SD-rules**

\[
\neg SD. \quad \neg \neg e \subseteq f
\]

\[
e(\bar{a}), \neg f(\bar{a})
\]

\( \bar{a} \) is a tuple of new constants,

\[
SD. \quad e(\bar{a}), e \subseteq f
\]

\[
f(\bar{a})
\]

\( \bar{a} \) is any tuple of constants.

**Projection rules**

\[
\neg PR. \quad \neg e[X](\bar{a}), e(\bar{b})
\]

\[
\neg \bar{b}_X = \bar{a}
\]

\( \bar{a}, \bar{b} \) are any tuples of constants,

\[
PR. \quad e[X](\bar{a})
\]

\[
e(\bar{b}), \bar{b}_X = \bar{a}
\]

\( \bar{a} \) is any tuple of constants and \( \bar{b} \) is a tuple of new constants.

**Restriction rules**

\[
\neg RE. \quad \neg e[X = Z](\bar{a}), e(\bar{a})
\]

\[
\neg \bar{a}_X = \bar{a}_Z
\]

\( \bar{a} \) is any tuple of constants.

**Selection rules**

\[
\neg SE. \quad \neg e[X = \bar{d}](\bar{a}), e(\bar{a})
\]

\[
\neg \bar{a}_X = \bar{d}
\]

\( \bar{a} \) is any tuple of constants.

**Example 3.2.** We exhibit a formal proof in S of the second half of Theorem 1 of [17]. This result essentially says that, given a partition of the columns of a relation name \( r \) into \( X, Y, Z \), if \( r : X \rightarrow Y \) or \( r : X \rightarrow Z \) hold, then the join of \( r[XY] \) and \( r[XZ] \) on \( X \) is a subset of \( r \). Using the definition of join in terms of product and restriction, we formalize
Table 2

1. \( r : X \rightarrow Y \lor r : X \rightarrow Z \) with \( \neg((r[XY] \times r[XZ])[X = X'])[XYZ'] \subseteq r \)
2. \( ((r[XY] \times r[XZ])[X = X'])[XYZ'] \) \( (\bar{a}, \bar{b}, \bar{c}) \) \( \neg r(\bar{a}, \bar{b}, \bar{c}) \)
3. \( ((r[XY] \times r[XZ])[X = X'])[\bar{a}, \bar{b}, \bar{a}', \bar{c}] \) \( \neg r(\bar{a}, \bar{b}, \bar{a}', \bar{c}) \)
4. \( r[XY] \times r[XZ] \) \( (\bar{a}, \bar{b}, \bar{a}', \bar{c}) \) \( \bar{a} = \bar{a}' \)
5. \( r[XY](\bar{a}, \bar{b}) \) \( r[XZ](\bar{a}', \bar{c}) \)
6. \( r(\bar{a}, \bar{b}, \bar{c}) \) \( r(\bar{a}', \bar{b}', \bar{c}) \)
7. \( r : X \rightarrow Z \) \( 8. \) \( r : X \rightarrow Y \)
8. \( \bar{c} = \bar{c}' \)
9. \( \bar{c} = \bar{c}' \) \( 10. \) \( \bar{b} = \bar{b}' \)
10. \( \bar{b} = \bar{b}' \)
11. \( r(\bar{a}, \bar{b}, \bar{c}) \) \( 12. \) \( r(\bar{a}, \bar{b}, \bar{c}) \)

\( X \) \( X \)

We indicate the tree structure spatially, so that 7. and 8. should be understood as sons of 6., and 10. as the only son of 8., for example.

the above assertion as the following wff (call it Q):

\[ r : X \rightarrow Y \lor r : X \rightarrow Z \Rightarrow ((r[XY] \times r[XZ])[X = X'])[XYZ'] \subseteq r \] (1)

where \( X', Z' \) are obtained by adding \( k \) to each element of \( X, Z \), respectively, if \( r \) is a \( k \)-ary relation name.

In Table 2 we offer closed tableaux as a proof that \( Q \) is indeed a tautology (starting with the result of applying an A-rule to the negation of (1)).

Example 3.3. In this example we prove that the following wff,

\[ e : SN \rightarrow SNAME, SCITY, STATUS \] (2)

is a theorem of the following set of wffs,

\[ \text{SUPPLIER} : SN \rightarrow SNAME, SCITY, \]
\[ \text{CS} : \text{CITY} \rightarrow \text{STATUS}, \] (3)

We offer the tableau in Table 3 as a proof.

4. Soundness and completeness of system S

We discuss in this section the soundness and
completeness of $S$. Soundness means that

$$
\forall P \vdash P \implies \neg \vdash P
$$

holds and completeness signifies that the converse holds. Since

$$
\forall P \vdash P \iff \neg P_1 \land \cdots \land P_n \implies P
$$

and

$$
\forall P \vdash P \iff \neg P_1 \land \cdots \land P_n \implies P,
$$

where $\forall = \{P_1, \ldots, P_n\}$, we may assume without loss of generality that $\forall$ is empty. (We tacitly assume that $\forall$ is always finite.) We also assume that the set of constants of the language $\mathcal{L}$ used by $S$ is infinite (which assures that we do not run out of constants during a proof).

The soundness of $S$ follows trivially. (All proofs appear in [4].)

**Theorem 4.1.** $S$ is sound.

To prove the completeness of $S$ we have to show that if $P$ is a tautology, then there is a closed tableau for $\neg P$ (i.e., that $\vdash P$). We actually prove that if $P$ is a tautology, then every complete tableau for $\neg P$ closes. Or, equivalently, that if there is a complete open tableau for $\neg P$, then $\neg P$ is satisfiable and, hence, $P$ is not a tautology. This result is obtained as follows. Recall that a tableau $\tau$ is complete and open iff every open branch $\beta$ of $\tau$ forms a Hintikka set. We prove that, in fact, any Hintikka set is satisfiable. Hence, $\beta$ is satisfiable and, since $\beta$ starts with $\neg P$, so is $\neg P$.

**Lemma 4.2.** Any Hintikka set is satisfiable.

In order to use Lemma 4.2 to obtain a completeness proof for $S$ we must guarantee that every branch of a tableau that does not close eventually becomes a Hintikka set. But the procedure given in Definition 3.1(a) permits constructing tableaux with infinite open branches which are not Hintikka sets. This follows because: (i) rules may be applied redundantly to introduce wffs already derived, (ii) rules $\neg FD$, $\neg SD$ and $PR$ may be repeatedly applied to generate wffs that differ only on the tuples of constants used, and (iii) rule $ES$ may always be applied using any tuple of constants. These problems are avoided by refining the procedure for constructing tableaux so that rules are nonredundantly applied in a cyclical pattern.

In what follows we assume that the 29 rules of $S$ are numbered from 0 to 28. Let $\forall$ and $\emptyset$ be sets of wffs occurring in branches $\beta$ and $\gamma$ (not necessarily distinct) of a tableau $\tau$, respectively. We say that $\forall$ is *higher than* $\emptyset$ iff some wff in $\forall$ occurs in a node of $\beta$ with level higher than the level of any node of $\gamma$ containing a wff of $\emptyset$. Likewise, let $\bar{a}$ and $\bar{b}$ be tuples of constants occurring in wffs of $\beta$ and $\gamma$, respectively. We say that $\bar{a}$ is *higher than* $\bar{b}$ iff $\bar{a}$ occurs in a node of $\beta$ with level higher than the level of any node of $\gamma$ containing an occurrence of $\bar{b}$.

The refined procedure for constructing tableaux works as follows. Let $\tau$ be an open tableau. Let $\forall_i$ be the last rule that was considered for application in $\tau$. Consider now for application rule $\forall_j$, $j = i + 1$ mod 29:

*Case 1. $\forall_j$ is not $ES$.*

Let $\forall$ be a set of wffs occurring in an open branch $\beta$ of $\tau$ that was never used as antecedent of $\forall_j$. If no such set $\forall$ exists, do not apply $\forall_j$. Otherwise, assume that no other set $\emptyset$ of wffs with the same property as $\forall$ is higher than $\forall$. Let $\lambda$ be the node lowest in $\beta$ that contains some wff in $\forall$. Then, extend all branches of $\tau$ that contain $\lambda$ by applying $\forall_j$. $\forall$ will never be used again as antecedent of $\forall_j$. This completes Case 1.

*Case 2. $\forall_j$ is $ES$.*

Let $\bar{a}$ be a tuple of constants occurring in an open branch $\beta$ of $\tau$ that was never used by $ES$. If no such $\bar{a}$ exists, do not apply $ES$. Otherwise, assume that no other tuple $\bar{b}$ with the same property as $\bar{a}$ is higher than $\bar{a}$. Let $\lambda$ be the node highest in $\beta$ that contains an occurrence of $\bar{a}$. Then, extend all branches of $\tau$ that contain $\lambda$ by applying $ES$ using $\bar{a} = \bar{a}$ as consequent. This concludes Case 2.

The procedure stops either when $\tau$ is closed or when all 29 rules of $S$ were unsuccessfully considered for application. By a *finished systematic tableau* we mean a tableau constructed by the refined procedure which is either infinite or else finite but cannot be extended further by the refined procedure.
We can now state the Completeness Theorem for system S.

**Theorem 4.3** (a) Every open branch of every finished systematic tableau is a Hintikka set.
(b) If a wff P is a tautology, then every finished systematic tableau starting with \(-P\) must close.
(c) System S is complete.

We conclude this section with some observations about the decidability of the tautology problem for an FD-SD language \(\mathcal{L}\) (i.e., the problem of determining whether a wff of \(\mathcal{L}\) is a tautology). This problem is undecidable because it can be trivially reduced to the equivalence problem for relational expressions of \(\mathcal{L}\), which is undecidable by Theorem 6 of [14]. The equivalence problem for relational expressions of \(\mathcal{L}\) is to determine whether two arbitrary expressions \(e\) and \(f\) of \(\mathcal{L}\), with the same arity, have the same value on any structure of \(\mathcal{L}\) (denoted by \(e \equiv f\)). Since \(e \equiv f\) holds iff \(e \subseteq f \land f \subseteq e\), the tautology problem for SDs alone is undecidable. By a more elaborate technique, the tautology problem for FDs alone can also be reduced to the equivalence problem.

By the above observations there is no procedure that receives as input any wff of any FD-SD language and always stops with ‘YES’, if P is a tautology, and ‘NO’, otherwise. In particular, the procedure described in this section may fail to stop even if P involves only FDs over relations and SDs over projections (just consider \(r[A] \subseteq r[B] \land r : A \rightarrow B \Rightarrow r[B] \subseteq r[A]\). which is true in every finite structure, but not a tautology).

### 5. Conclusions and directions for future research

This paper described a formal system for FDs and SDs over expressions and an associated proof procedure based on the analytic tableaux method. The simplicity of the system, as compared to that in [14] for FDs alone, derives from the explicit use of tuples of constants and equality. In fact, our earlier experience indicates that, without this device, FDs and SDs have to be generalized into complicated dependencies, least searching for a complete system becomes a very difficult, if not hopeless task.

The analytic tableaux method proved to be quite attractive and easy to use manually. However, it would be reasonable inefficient to implement the method as we described. This is complicated by the fact that the tautology problem for FD-SD languages is undecidable. Hence, fast decision procedures or at least efficient heuristics for reducts of the full problem should be sought.

Finally, we observe that the completeness result of Section 4 generalizes to other types of dependencies, if the induction carried out to prove Lemma 4.2 still holds.

### References


