On improving dependency implication algorithms

Wai Yin Mok\textsuperscript{a,}\textsuperscript{*}, David W. Embley\textsuperscript{b,1}

\textsuperscript{a} Department of Mathematical Sciences, The University of Akron, Akron, OH, USA
\textsuperscript{b} Brigham Young University, Provo, UT, USA

Received 9 December 1996; revised 17 September 1997
Communicated by G.R. Andrews

Keywords: Algorithms, Databases, Dependency basis, Join dependencies, Multivalued dependencies; Functional dependencies; Relational database theory

1. Introduction

Let $\Sigma$ be a finite set of join dependencies (JDs), multivalued dependencies (MVDs) and functional dependencies (FDs). To test whether $\Sigma$ implies an MVD $X \rightarrow Y$ or an FD $X \rightarrow A$, the procedure in [3] first generates the dependency basis $\text{DEP}(X)$ of $X$ with respect to $\Sigma$. Then, $X \rightarrow Y$ if and only if $Y$ is the union of some elements in $\text{DEP}(X)$, and $X \rightarrow A$ if and only if $A \in \text{DEP}(X)$, and there is an FD $V \rightarrow W$ in $\Sigma$ with $A \subseteq W - V$.

In this paper, we present two algorithms, Algorithms 1 and 2, that test the same implication problems. However, these algorithms do not generate $\text{DEP}(X)$ with respect to $\Sigma$, but rather directly determine whether $X \rightarrow Y$ or $X \rightarrow Y$ without computing the basis for all right-hand sides $Z$ for $X \rightarrow Z$ or $X \rightarrow Z$ as in [3]. As part of proving these algorithms correct, we also prove a result that is interesting in its own right.

We organize this paper as follows. In Section 2, we present the basic definitions and some results in the literature that we need. In Section 3, we prove a theorem whose corollary is the basis of our algorithms. In Section 4, we present Algorithm 1 that tests if $\Sigma$ implies an FD. In Section 5, we present Algorithm 2 that tests if $\Sigma$ implies an MVD. We conclude in Section 6.

2. Preliminaries

$U$ is a finite set of attributes and $\Sigma$ is a set of JDs, MVDs and FDs. All JDs, MVDs and FDs are formed from the attributes in $U$. We denote a JD by $*[R_1, \ldots, R_p]$ where $\bigcup_{i=1}^{p} R_i = U$.

A JD $J = *[R_1, \ldots, R_p]$ may imply numerous MVDs. However, every MVD implied by $J$ is also implied by $\text{MVD}(J)$ where $\text{MVD}(J)$ is the set of MVDs of the form $*[\bigcup_{i \in S_1} R_i, \bigcup_{i \in S_2} R_i]$, where $S_1 \cup S_2 = \{1, \ldots, p\}$ and $S_1 \cap S_2 = \emptyset$ [4].

In this paper we use two inference rules from the literature on relational dependency theory, namely, the subset rule for MVDs in [1], and C2 in [4]. The subset rule states that if $X \rightarrow Y$, $V \rightarrow W$ and $Y \cap V = \emptyset$, then $X \rightarrow (Y \cap W)$ and $X \rightarrow (Y - W)$. C2 states that if $X \rightarrow Y$, $V \rightarrow W$, $Y \cap V = \emptyset$, and $W \subseteq Y$, then $X \rightarrow W$.

Essential to our algorithms are the maximal connected components of a JD. Since for every JD, there
is a corresponding hypergraph \[2\], we use the terminology for hypergraphs to define what a maximal connected component of a JD is. The hypergraph of a JD \[\{R_1, \ldots, R_p\}\] has as its set of nodes those attributes that appear in one or more of the \(R_i\)'s; its set of edges is \(\{R_1, \ldots, R_p\}\). A path from attribute \(A\) to attribute \(B\) is a sequence of \(k \geq 1\) edges \(E_1, \ldots, E_k\) such that (1) \(A \in E_1\), (2) \(B \in E_k\), and (3) \(E_i \cap E_{i+1}\) is nonempty for \(1 \leq i < k\). Two attributes are connected if there is a path from one to the other. A nonempty set of attributes is a connected component if every pair in the set is connected. A connected component \(S\) is maximal for a JD \(\{R_1, \ldots, R_p\}\) if there does not exist an attribute \(A \in \cup_{i=1}^{p} R_i\) such that \(SA\) is a connected component.

If we remove a set \(Z\) of attributes from a JD \(J = \{R_1, \ldots, R_p\}\), we obtain another JD \(J' = \{R_1 - Z, \ldots, R_p - Z\}\). If \(C_1, \ldots, C_m\) are the maximal connected components of \(J\), then by Theorem 3 in [2], \(J\) implies the MVDs \(Z \rightarrow C_i, 1 \leq i \leq m\). Since \(J\) implies \(Z \rightarrow C_i\), MVD(\(J\)) also implies \(Z \rightarrow C_i, 1 \leq i \leq m\).

**Example 1.** Let \(U = ABCDEF\) and \(J = \{AB, BC, AC, CD, DE, DF\}\). \(J\) has only one maximal connected component, namely \(ABCD\). If we remove \(CF\) from \(J\) to obtain \(J' = \{AB, B, A, D, DE\}\), which is equivalent to \(\{AB, DE\}\), then \(J'\) has two maximal connected components \(AB\) and \(DE\). Note that \(ABDE\) is not a connected component of \(J\) since \(B\) and \(D\) are not connected in \(J\) (or in its corresponding hypergraph).

We also note that \(J\) implies \(CF \rightarrow AB\) and \(CF \rightarrow DE\), and thus that MVD(\(J\)) also implies \(CF \rightarrow AB\) and \(CF \rightarrow DE\).

### 3. Main theorem

In this section, we prove Theorem 3 whose corollary is the basis for Algorithms 1 and 2.

**Lemma 2.** Let \(r\) be a two-row relation over \(U\) with rows \(r_1\) and \(r_2\). Let \(Z = \{A \in U \mid r_1(A) = r_2(A)\}\). Relation \(r\) satisfies a JD \(J = \{R_1, \ldots, R_p\}\) if and only if \(J' = \{R_1 - Z, \ldots, R_p - Z\}\) has only one maximal connected component.

**Proof.** (if) Suppose \(J'\) has only one maximal connected component \(C\). Let \(G = \{R_i \mid R_i\ in J\ and R_i - Z\ is a nonempty subset of C\}\) and let \(\tilde{G} = \{R_1, \ldots, R_p\} - G\). Now suppose that \(r\) does not satisfy \(J\) so that \(w_{i=1}^p \pi_{R_i} r\) generates a new row \(t\), \(t \neq r_1\) and \(t \neq r_2\). It must be that \(w_i(R_i) = t(R_i)\) where \(w_i\) is either \(r_1\) or \(r_2\) but not both. Since every pair of attributes in \(C\) is connected, to make these \(p\) rows joinable, we only have two choices. We have to let \(w_i = r_1\) for each \(R_i\) in \(G\) or we have to let \(w_i = r_2\) for each \(R_i\) in \(G\). (For the \(R_i\)'s in \(\tilde{G}\), \(r_1(R_i) = r_2(R_i)\).) If we make the first choice, \(t = r_1\); if we make the second choice, \(t = r_2\). We therefore have a contradiction and \(r\ satisfies \(J\).

(only if) Suppose \(J'\) has at least two maximal connected components \(C'\) and \(C''\), then \(J\) implies \(Z \rightarrow C'\) and \(Z \rightarrow C''\). Since \(C'\) and \(C''\) are both nonempty, \(r\ does not satisfy \(Z \rightarrow C'\) or \(Z \rightarrow C''\). Thus, \(r\ does not satisfy \(J\) – a contradiction. \(\square\)

**Theorem 3.** Let \(U\) be a finite set of attributes and \(\Sigma\) be a set of JDSs, MVDs and FDs over \(U\). Let \(\Sigma = J \cup F\) where \(J\) is a set of JDSs \(\{J_1, \ldots, J_n\}\) (including MVDs but not FDs), and \(F\) is a set of FDs.

1. \(\Sigma\) implies \(X \rightarrow Y\) if and only if \(\operatorname{MVD}(J_1) \cup \cdots \cup \operatorname{MVD}(J_n) \cup F\) implies \(X \rightarrow Y\).
2. \(\Sigma\) implies \(X \rightarrow A\) if and only if \(\operatorname{MVD}(J_1) \cup \cdots \cup \operatorname{MVD}(J_n) \cup F\) implies \(X \rightarrow A\).

A result similar to (1) appears in [3]. We therefore prove (2). We also mention that (1) can be proved using Lemma 2 similar to the way we prove (2). Moreover, this proof is considerably shorter and more straightforward than the proof of the corresponding result in [3].

**Proof.** (2) The if-part is trivial since \(J_i\) implies \(\operatorname{MVD}(J_i), 1 \leq i \leq n\). Suppose \(\operatorname{MVD}(J_1) \cup \cdots \cup \operatorname{MVD}(J_n) \cup F\) does not imply \(X \rightarrow A\). By Theorem 8 in [5], there is a two-row relation \(r\) over \(U\) such that \(r\ satisfies \(\operatorname{MVD}(J_1) \cup \cdots \cup \operatorname{MVD}(J_n) \cup F\) but not \(X \rightarrow A\). To prove the only if-part, it suffices to show that \(r\ also satisfies \(J\). Assume \(r\ does not satisfy \(J_i = \{R_1, \ldots, R_p\}\) for some \(i\). Let \(Z = \{A \in U \mid r_1(A) = r_2(A)\}\) where \(r_1\) and \(r_2\) are the two rows in \(r\). By Lemma 2, \(J'_i = \{R_1 - Z, \ldots, R_p - Z\}\) has at least two maximal connected components \(C\) and \(C'\). Since \(C\) and \(C'\) are both nonempty, \(r\ does not satisfy
Algorithm 1.

Input: A finite set \( \Sigma \) of JDs, MVDs and FDs, and an FD \( X \rightarrow A \) over a set of attributes \( U \).

Output: \( \Sigma \) implies \( X \rightarrow A \) or \( \Sigma \) does not imply \( X \rightarrow A \).

Initialization: Establish a two-row relation \( r \) over \( U \) with rows \( r_1 \) and \( r_2 \); \( r_1 \) has \( a \)'s in all the columns; \( r_2 \) has \( a \)'s in all the \( X \)-columns and \( b \)'s in all the \( (U-X) \)-columns. Also, rewrite each FD \( X \rightarrow Y \) in \( \Sigma \) to have single-attribute right-hand sides, \( X \rightarrow B_i \) (1 \( \leq i \) \( \leq n \)) such that \( Y = B_1 \ldots B_n \).

Useful macro: \( Z = \{ B \in U \mid r_1(B) = r_2(B) \} \) and \( W = U - Z \). (Note that \( Z \) and \( W \) change during the execution of Algorithm 1.)

Termination: If \( r_2(\Lambda) \) is or becomes an \( a \), stop immediately and conclude: \( \Sigma \) implies \( X \rightarrow A \); otherwise, stop after Step 3 below and conclude: \( \Sigma \) does not imply \( X \rightarrow A \). (Note that for \( X \rightarrow A \); either \( r_2(\Lambda) = a \) initially or the \( b \) in \( r_2(\Lambda) \) must be changed to an \( a \) by Step 1. For some other attribute \( D \), a \( b \) in \( r_2(D) \) may be changed from a \( b \) to an \( a \) in Step 2, but this does not imply \( X \rightarrow D \).)

1. While there is an FD \( V \rightarrow B \) in \( \Sigma \) such that \( V \subseteq Z \) and \( B \in W \), change \( r_1(B) \) to an \( a \).

2. While there is a JD \( *[R_1, \ldots, R_p] \) (including MVDs but not FDs) in \( \Sigma \) such that \( *[R_1 - Z, \ldots, R_p - Z] \) has at least two maximal connected components, choose the maximal connected component \( C \) that contains the attribute \( A \) and modify \( r_2 \) so that all \( b \)'s in \( r_2(U - C) \) become \( a \)'s.

3. If \( r \) does not satisfy some FD in \( \Sigma \), go back to Step 1.

Step 1 modifies \( r_2 \) so that \( r \) satisfies the FDs in \( \Sigma \). Similarly, Step 2 modifies \( r_2 \) so that \( r \) satisfies the JDs in \( \Sigma \) that are not FDs.

Before we prove that Algorithm 1 is correct, we first prove a lemma whose corollaries are used later.

Lemma 5. At any stage of Algorithm 1, \( X \rightarrow W \).

(Notaem that \( Z = \{ B \in U \mid r_1(B) = r_2(B) \} \) and \( W = U - Z \), and that these two sets are constantly updated during the execution of Algorithm 1.)

Proof. We proceed by induction. When we start Algorithm 1, \( Z = X \) and \( W = U - X \). Thus, \( X \rightarrow W \) initially. For the induction step, we assume \( X \rightarrow W \). If \( r \) does not satisfy an FD \( V \rightarrow B \) in \( \Sigma \), then \( V \subseteq Z \) and \( B \notin W \). Since \( Z \cap W = \emptyset \) and \( V \cap W = \emptyset \). Since \( X \rightarrow W \) by the induction hypothesis, \( V \rightarrow B \), \( V \cap W = \emptyset \), and \( B \in W \), by C2, \( X \rightarrow B \). In Step 1, \( r_2(B) \) is changed from a \( b \) to an \( a \) and then \( W \) is updated to be \( W - B \). Since \( X \rightarrow B \), \( X \rightarrow W \) continuously. If \( r \) does not satisfy a JD \( J \rightarrow *[R_1, \ldots, R_p] \) in \( \Sigma \), by Lemma 2, \( J' \rightarrow *[R_1 - Z, \ldots, R_p - Z] \) has at least two maximal connected components. Let \( C_1, \ldots, C_m, m \geq 2 \), be the maximal connected components of \( J' \). Therefore, \( C_1 \cup \cdots \cup C_m = W \). By Theorem 3 in [2], \( J \) implies \( Z \rightarrow C_i, 1 \leq i \leq m \). Since \( X \rightarrow W \) by the induction hypothesis, \( Z \rightarrow C_i, Z \cap W = \emptyset \) and \( C_i \subseteq W \), by the subset rule, \( X \rightarrow C_i, 1 \leq i \leq m \). In Step 2, \( W \) becomes one of the \( C_i \)'s and thus \( X \rightarrow W \) continuously. \( \square \)

Corollary 6. If \( r_2(\Lambda) \) is or becomes an \( a \), \( X \rightarrow A \).

Proof. If \( r_2(\Lambda) \) is an \( a \), then \( A \in X \) and thus \( X \rightarrow A \) trivially. Suppose \( A \notin X \); that is, \( r_2(\Lambda) \) is initialized as a \( b \) and thus \( A \in W \). Since \( A \in C \) in Step 2 of Algorithm 1, \( r_2(\Lambda) \) can only become an \( a \) in Step 1. In Step 1, \( r_2(\Lambda) \) becomes an \( a \) only if there is an FD \( V \rightarrow A \) in \( \Sigma \) such that \( V \subseteq Z \)
and \( A \in W \). Since \( V \subseteq Z \) and \( Z \cap W = \emptyset \), \( W \cap V = \emptyset \). By Lemma 5, \( X \rightarrow W \) continuously. Since \( X \rightarrow W, V \rightarrow A, W \cap V = \emptyset, \) and \( A \in W \), by C2, \( X \rightarrow A \). \( \square \)

**Corollary 7.** If there is a JD \(*[R_1,\ldots,R_p]*\) in \( \Sigma \) such that \( C_1,\ldots,C_m, m \geq 2 \), are the maximal connected components of \(*[R_1 - Z,\ldots,R_p - Z]*\), then \( X \rightarrow C_i, 1 \leq i \leq m \).

**Proof.** By Lemma 5, \( X \rightarrow W \) continuously. The rest of the proof is similar to that of Corollary 6. The main difference is that instead of using C2 in this proof, we use the subset rule for MVDs. \( \square \)

**Theorem 8.** Let \( U \) be a finite set of attributes and \( \Sigma \) be a set of JDS, MVDs and FDs over \( U \). \( \Sigma \) implies \( X \rightarrow A \) if and only if Algorithm 1 outputs "\( \Sigma \) implies \( X \rightarrow A \)."

**Proof.** (if) Suppose Algorithm 1 outputs "\( \Sigma \) implies \( X \rightarrow A \)." Then, \( r_2(A) \) is initialized as or becomes an \( a \) in Algorithm 1. Thus, by Corollary 6, \( \Sigma \) implies \( X \rightarrow A \).

(only if) By way of contradiction, suppose Algorithm 1 outputs "\( \Sigma \) does not imply \( X \rightarrow A \)." Then, \( r \) satisfies \( \Sigma \), but \( r_2(A) \) remains a \( b \). But now, \( r \) is a counterexample to \( \Sigma \) implies \( X \rightarrow A \). Thus, by Corollary 4, \( \Sigma \) does not imply \( X \rightarrow A \) - a contradiction. \( \square \)

5. **Testing MVD implication**

In this section, we present Algorithm 2 that tests if \( \Sigma \) implies an MVD. Algorithm 2 is based on Algorithm 1 (and thus it is also based on Corollary 4). Given an MVD \( X \rightarrow Y \), like Algorithm 1, Algorithm 2 manipulates a two-row relation in certain ways and \( \Sigma \) implies \( X \rightarrow Y \) if and only if Algorithm 2 fails to create a two-row relation that satisfies \( \Sigma \) but not \( X \rightarrow Y \).

Basically, Algorithm 2 keeps track of two sets, namely \( Y' \) and \( Y'' \) where \( X \rightarrow Y' \) and \( X \rightarrow Y'' \) continuously (we shall prove these facts in Theorem 12). If \( Y' \) grows from \( \emptyset \) to \( Y \) or \( Y'' \) grows from \( \emptyset \) to \( U - XY \), then \( \Sigma \) implies \( X \rightarrow Y \). Otherwise, we shall have a two-row counterexample relation.

**Algorithm 2.**

Input: A finite set \( \Sigma \) of JDS, MVDs and FDs, and an MVD \( X \rightarrow Y \) over \( U \). Without loss of generality we may assume \( X \cap Y = \emptyset \).

Output: \( \Sigma \) implies \( X \rightarrow Y \) or \( \Sigma \) does not imply \( X \rightarrow Y \).

Initialization: \( Y' = Y'' = \emptyset \). Also, rewrite each FD \( V \rightarrow B \) in \( \Sigma \) to have single-attribute right-hand sides, \( X \rightarrow B_i (1 \leq i \leq n) \) such that \( Y = B_1 \ldots B_n \).

Useful macro: \( Z \) and \( W \) are defined as in Algorithm 1.

Termination: If \( Y' \) is or becomes \( Y \) or \( Y'' \) is or becomes \( U - XY \), stop immediately and conclude: \( \Sigma \) implies \( X \rightarrow Y \); otherwise, stop in Step 4 as directed and conclude: \( \Sigma \) does not imply \( X \rightarrow Y \).

1. If \( X + Y \) is trivial, let \( Y' \) be \( Y \) and stop; otherwise, establish a two-row relation \( r \) over \( U \) with rows \( r_1 \) and \( r_2 \): \( r_1 \) has \( a \)'s in all the columns; and \( r_2 \) has \( a \)'s in all the \( X \)-columns and \( b \)'s in all the \( (U - X) \)-columns.

2. While there is an FD \( V \rightarrow B \) in \( \Sigma \) such that \( V \subseteq Z \) and \( B \in W \), change \( r_2(B) \) to an \( a \). If \( B \in Y \), add \( B \) to \( Y' \) and stop if \( Y' \) becomes \( U \) - \( XY \). If \( B \in (U - XY) \), add \( B \) to \( Y'' \) and stop if \( Y'' \) becomes \( U - XY \).

3. While there is a JD \(*[R_1,\ldots,R_p]*\) (including MVDs but not FDs) in \( \Sigma \) such that \( C_1,\ldots,C_m, m \geq 2 \), are the maximal connected components of \(*[R_1 - Z,\ldots,R_p - Z]*\), divide \( \{C_1,\ldots,C_m\} \) into three sets, namely, \( S_1 = \{C_i \mid C_i \subseteq Y\} \), \( S_2 = \{C_i \mid C_i \subseteq (U - XY)\} \), and \( S_3 = \{C_1,\ldots,C_m\} - (S_1 \cup S_2) \). Update \( Y' = Y' \cup C_i \) where \( C_i \in S_1 \). If \( Y' = Y \), stop. Update \( Y'' = Y'' \cup C_i \) where \( C_i \in S_2 \). If \( Y'' = U - XY \), stop. If \( S_3 = \emptyset \), then arbitrarily choose a \( C_i \) in \( S_3 \) and modify \( r \) so that all \( b \)'s in \( r_2(U - C_i) \) become \( a \)'s. If \( S_3 \neq \emptyset \), all \( b \)'s in \( r_2 \) become \( a \)'s.

4. If \( r \) does not satisfy some FD in \( \Sigma \), go back to Step 2; else if \( W \cap Y \neq \emptyset \) and \( W \cap (U - XY) \neq \emptyset \), then stop and conclude \( \Sigma \) does not imply \( X \rightarrow Y \); else reestablish \( r_2 \) such that all the \((XY'Y'')\)-columns are \( a \)'s and all the \((U - (XY'Y''))\)-columns are \( b \)'s and then go back to Step 2.

Except for the last part of Step 4 where we reestablish \( r_2 \), Algorithm 2 is reasonably straightforward. We give an example to illustrate the last part of Step 4 and to show that this part is necessary.
Example 9. Let \( U = ABCDEF \) and \( \Sigma = \{ J_1 = \{ AB, BCD, CDEF \}, J_2 = \{ AC, CD, BD, BF, EF, AE \}, C \rightarrow D \} \). Algorithm 2 constructs a two-row counterexample relation showing that \( \Sigma \) does not imply \( C \rightarrow AE \). Let us denote a tuple on the relation scheme \( ABCDEF \) by \( \langle v_1 v_2 \ldots v_6 \rangle \) in which \( v_i, 1 \leq i \leq 6 \), is either an \( a \) or a \( b \), and where \( v_1 \) is a value under the \( A \)-column; \( \ldots \); and \( v_6 \) is a value under the \( F \)-column. Initially, \( Y' = Y'' = \emptyset \), \( r_1 = \langle aaaaaa \rangle \), and \( r_2 = \langle bbabbb \rangle \). In Step 2, after we apply the FD \( C \rightarrow D \), \( r_2 \) becomes \( \langle bbabbb \rangle \) and \( Y'' \) becomes \( \langle aaaaaa \rangle \). In Step 3, after we remove \( CD \) from \( J_1 \), \( AB \) and \( EF \) are the two resulting maximal connected components. Therefore, both \( S_1 \) and \( S_2 \) are \( \emptyset \), and \( S_3 = \{ AB, EF \} \). We now have two choices. Suppose we choose \( AB \). Then, \( r_2 \) becomes \( \langle bbabbb \rangle \). However, if we remove \( CDEF \) from \( J_2 \), \( A \) and \( B \) are the two resulting maximal connected components. In Step 3, \( Y'' \) becomes \( A \), \( Y'' \) becomes \( BD \), and \( r_2 \) becomes \( \langle aaaaaa \rangle \). But now in Step 4, \( r_2 \) is reestablished as \( \langle aaaaab \rangle \). In Step 2, the relation satisfies the FD \( C \rightarrow D \). In Step 3, after we remove \( ABCD \) from either \( J_1 \) or \( J_2 \), we have only one resulting maximal connected component. Therefore, in Step 4, we conclude that \( \Sigma \) does not imply \( C \rightarrow AE \).

Before we prove that Algorithm 2 is correct, we first prove two lemmas. The first guarantees that the algorithm terminates; and the second is useful in proving the algorithm correct.

**Lemma 10.** Algorithm 2 terminates.

**Proof.** Let \( X \rightarrow Y \) be the MVD being tested. If \( X \rightarrow Y \) is trivial, the algorithm stops in Step 1. Since the number of FDs is finite, the while loop in Step 2 always terminates. In Step 3, if \( Y' \) becomes \( Y \) or \( Y'' \) becomes \( U - XY \), we stop. If not, we continue and observe that either \( S_1 = \emptyset \) or \( S_3 \neq \emptyset \). If \( S_3 = \emptyset \), all \( b \)'s in \( r_2 \) become \( a \)'s, and the while loop in Step 3 halts. If \( S_3 \neq \emptyset \), then there are \( n \geq 2 \) maximal connected components \( C_1, \ldots, C_n \) such that \( C_i \neq \emptyset \), \( C_i \cap C_j = \emptyset \) if \( i \neq j \), \( 1 \leq i, j \leq n \), and \( C_1 \cup \cdots \cup C_n = W = U - Z \). Let \( C_k \) be the chosen maximal connected component. Since we modify \( r_2 \) so that all \( b \)'s in \( r_2(U - C_k) \) become \( a \)'s and since none of the components of \( S_3 \) is empty and since there are at least two components of \( S_3 \), at least one \( b \) becomes an \( a \). Hence, \( Z \) grows in every iteration (unless the loop halts because \( Y' \) becomes \( Y \), or \( Y'' \) becomes \( U - XY \), or unless \( S_3 = \emptyset \)). Since \( U \) is finite, \( Z \) cannot grow indefinitely without becoming \( U \), in which case the while loop in Step 3 halts.

For Step 4, we first observe that neither \( Y' \) nor \( Y'' \) ever diminishes in size. Second, we observe that if \( Y'Y'' \) always grows larger each time we reestablish \( r_2 \), then eventually either \( Y' \) becomes \( Y \) or \( Y'' \) becomes \( U - XY \) and we stop (unless we stop sooner). We therefore need to show that Step 4 either halts with a negative response or that \( Y'Y'' \) always grows larger in each iteration. In Step 4, if \( r \) does not satisfy some FD in \( \Sigma \), we return to Step 2 where we either add to \( Y' \) or \( Y'' \), which causes \( Y'Y'' \) to grow. Otherwise, if \( r \) satisfies all the FDs in \( \Sigma \), we stop if \( W \cap Y \neq \emptyset \) and \( W \cap (U - XY) \neq \emptyset \). Thus, we assume that \( W \cap Y = \emptyset \) or \( W \cap (U - XY) = \emptyset \) and reestablish \( r_2 \) as \( a \)'s in the \( XY'Y'' \)-columns and \( b \)'s elsewhere. If \( r \), as newly established, does not satisfy some FD in \( \Sigma \), \( Y'Y'' \) grows, so we assume that \( r \) satisfies all the FDs in \( \Sigma \) and enter Step 3. But here, since we are assuming that \( Y'Y'' \) does not grow, \( S_1 = S_2 = \emptyset \). Furthermore, \( S_1 \) cannot be empty or become empty for if so then since \( S_1 \) and \( S_2 \) are also empty, the condition of the while loop must have been false. The only way to continue is if \( S_3 \) is not empty. But then since every \( C_i \) of \( S_3 \) has an attribute in both \( Y \) and \( U - XY \), \( W \) retains at least one \( b \) in \( Y \) and at least one \( b \) in \( U - XY \). Since we cannot continue to add \( a \)'s to \( r_2 \) indefinitely, we must eventually terminate with at least one \( b \) remaining in both \( Y \) and \( U - XY \). But, now either \( r \) does not satisfy an FD in \( \Sigma \) so that \( XY'Y'' \) grows or \( W \cap Y \neq \emptyset \) and \( W \cap (U - XY) \neq \emptyset \) and Step 4 halts. \( \Box \)

Before we state and prove Lemma 11, we note that Algorithm 1 and Algorithm 2 are basically the same. Step 2 of Algorithm 2 corresponds to Step 1 of Algorithm 1. Step 3 of Algorithm 2 corresponds roughly to Step 2 of Algorithm 1. One difference between these two steps is that in Step 2 of Algorithm 1, we know which maximal connected component to choose, but in Step 3 of Algorithm 2, we arbitrarily choose one maximal connected component in \( S_3 \). The other difference is that in Algorithm 2, we keep track of two sets \( Y' \) and \( Y'' \). We observe that neither difference is needed in the proofs for Corollaries 6 and 7. Hence, we use these two corollaries in our proof of Lemma 11.
Lemma 11. At any stage of Algorithm 2, \( X \rightarrow Y' \) and \( X \rightarrow Y'' \).

**Proof.** If \( X \rightarrow Y \) is trivial, \( Y' \) becomes \( Y \) in Step 1 and we stop. Clearly, \( X \rightarrow Y' \) and \( X \rightarrow Y'' \). Suppose \( X \rightarrow Y \) is not trivial. We first show that \( X \rightarrow Y' \) and \( X \rightarrow Y'' \) continually before \( r_2 \) is reestablished for the first time. Initially, \( Y' = Y'' = \emptyset \). Hence, \( r_2 \) has \( a \)'s only in the \( X \)-columns and \( b \)'s elsewhere. By Corollary 6, if \( r_2(A) \) becomes an \( a \), \( X \rightarrow A \) and we either add \( A \) to \( Y' \) or \( Y'' \). Therefore, \( X \rightarrow Y' \) and \( X \rightarrow Y'' \) after we add attributes to either set in Step 2. By Corollary 7, if there is a \( \mathrm{JD} \ [R_1, \ldots, R_p] \) in \( \Sigma \) such that \( C_1, \ldots, C_m, m \geq 2 \), are the maximal connected components of \( [R_1 - Z, \ldots, R_p - Z] \), then \( X \rightarrow C_i, 1 \leq i \leq m \). For each \( C_i \) that is a subset of \( Y \), we add it to \( Y' \), and for each \( C_i \) that is a subset of \( U - XY \), we add it to \( Y'' \). Therefore, \( X \rightarrow Y' \) and \( X \rightarrow Y'' \) after we add attributes to either set in Step 3. Now, if we reestablish \( r_2 \) in Step 4 so that \( r_2 \) only has \( a \)'s in the \( XY'Y'' \)-columns (let \( XY'Y'' = V \)), then using the same reasoning, we can show that \( V \rightarrow Y' \) and \( V \rightarrow Y'' \) continually until we reestablish \( r_2 \) for the second time in Step 4. Since we have shown that \( X \rightarrow Y' \) and \( X \rightarrow Y'' \) when we reestablish \( r_2 \) for the first time, \( X \rightarrow XY'Y'' \) (or \( X \rightarrow V \)). Hence, just before \( r_2 \) is reestablished for the second time, since \( X \rightarrow V \) and \( V \rightarrow Y' \), we have \( X \rightarrow (Y' - V) \). But the attributes in \( Y' - V \) are exactly the attributes we add to \( Y' \) after \( r_2 \) is reestablished for the first time and before it is reestablished for the second time. Hence, \( X \rightarrow Y' \) before it is reestablished the second time. By using the same reasoning inductively, we can show that \( X \rightarrow Y' \) continually. Similarly, \( X \rightarrow Y'' \). \( \square \)

**Theorem 12.** Let \( U \) be a finite set of attributes and \( \Sigma \) be a set of \( \mathrm{JDs} \), \( \mathrm{MVDs} \) and \( \mathrm{FDs} \) over \( U \). \( \Sigma \) implies \( X \rightarrow Y \) if and only if Algorithm 2 outputs “\( \Sigma \) implies \( X \rightarrow Y \).”

**Proof.** (if) Suppose Algorithm 2 outputs “\( \Sigma \) implies \( X \rightarrow Y \).” Then, \( Y' \) is initialized as or becomes \( Y \) or \( Y'' \) becomes \( U - XY \). Since by Lemma 11, \( X \rightarrow Y' \) and \( X \rightarrow Y'' \), \( \Sigma \) implies \( X \rightarrow Y \).

(only if) By way of contradiction, suppose Algorithm 2 outputs “\( \Sigma \) does not imply \( X \rightarrow Y \).” Thus, \( r \) satisfies every dependency in \( \Sigma \) and \( W \cap Y \neq \emptyset \) and \( W \cap (U - XY) \neq \emptyset \). But now, \( \pi_{XY} \Sigma \neq \pi_{X(Y - XY)} \Sigma \neq r \). Thus, \( \Sigma \) does not imply \( X \rightarrow Y \) – a contradiction. \( \square \)

6. Time complexity of the algorithms

In this section, we compare the time complexity of our algorithms, particularly Algorithm 2, and the algorithm in [3]. Given an MVD, Algorithm 2 tries to find a counterexample to show that \( \Sigma \) does not imply the given MVD. The algorithm in [3], however, first finds the dependency basis. Algorithm 2 is designed to solve the implication problem of a particular MVD. On the contrary, if there are many MVDs to be tested, it may be wise to first find the dependency basis since the dependency basis can then be used to solve many implication problems.

We now argue that on average Algorithm 2 takes about one third of the running time of the algorithm in [3]. Since an FD can only be applied once, the running time of Step 2 of Algorithm 2 is linear. Step 1 executes only once. The number of times Step 4 executes depends on Step 3. Hence, Step 3 of Algorithm 2 dominates all the other steps. As pointed out in [3], the time complexity of splitting up a maximal connected component in Step 3 is bounded by \( O(||\Sigma||) \), where \( ||\Sigma|| \) is the space required to write down \( \Sigma \). In the worst case, there are at most \( |U| \) maximal connected components to be split up, where \( |U| \) is the number of attributes in \( U \). Hence, Algorithm 2 is bounded by \( O(|U| \cdot ||\Sigma||) \), which is the same as that of the algorithm in [3].

On average, however, Algorithm 2 performs better than this. Since Algorithm 2 tries to find a counterexample, in Step 3, only the maximal connected components in \( S_2 \) are useful. By Lemma 2, we thus randomly choose one maximal connected component \( C \) from \( S_2 \) to modify the two-tuple relation \( r \) such that the two tuples disagree only under the \( C \)-columns and agree elsewhere. The newly modified \( r \) now satisfies the JD in hand but not the MVD that is being tested. Since we only need to consider the maximal connected components in \( S_1 \), we thus save time by ignoring the maximal connected components in \( S_3 \) and \( S_4 \). Because a maximal connected component, on the average, is equally likely to be in \( S_1 \), \( S_2 \) or \( S_3 \), the average running time of Algorithm 2 is about one third of that of the worst case. Note that the same analysis can be applied to Algorithm 1.
7. Concluding remarks

We have presented two algorithms, one for testing if a finite set $\Sigma$ of JDs and FDs implies an FD and another for testing if $\Sigma$ implies an MVD, and we have proved that these algorithms are correct. These two algorithms are based on the fact that if $\Sigma$ does not imply an FD or MVD, there is always a two-row counterexample relation. Our algorithms systematically look for such counterexample relations and produces one if the implication we are testing does not hold. On the average, this approach leads to a decrease of about one-third over the running time of the dependency-basis algorithm in [3].

References