EFFICIENT DATABASE UPDATES WITH INDEPENDENT SCHEMES*

RICCARDO TORLONE† AND PAOLO ATZENI†

Abstract. The weak instance model is a framework to consider the relations in a database as a whole, regardless of the way attributes are grouped in the individual relations. Queries and updates can be performed involving any set of attributes. The management of updates is based on a lattice structure on the set of legal states, and inconsistencies and ambiguities can arise.

In the general case, the test for consistency and determinism may involve the whole database. In this paper it is shown how, for the highly significant class of independent schemes, updates can be handled efficiently, considering only the relevant portion of the database.

Key words. Relational databases, weak instance model, lattice on database states, update operations, chase procedure, optimization

AMS subject classifications. 68P15, 68Q25, 05A05

1. Introduction. In a relational database, the universe of discourse is represented by means of a set of relations. The *weak instance approach* [6, 18, 22, 23, 24, 31] provides a framework to consider a database as a whole, regardless of the way attributes appear in the various relation schemes. The information content of a database state is considered to be embodied in the *representative instance*, a sort of relation with variables, obtained by extending all relations to the global set of attributes (called the *universe*) and then by *chasing* [21] their union. Query answering is performed by first computing the *total projection* [26] of the representative instance on the set of attributes involved in the query, and then executing whatever further operations are needed. Any subset of the universe can in principle be queried.

EXAMPLE 1.1. *Consider a database scheme with relations* $R_1(EDP)$, $R_2(DMP)$ *and the functional dependencies* $E \rightarrow D$, $D \rightarrow M$ *as constraints. Figures 1.1, 1.2, and 1.3, show a consistent database state on this scheme, the corresponding representative instance, and a total projection, respectively.*

Recently, following a new interest on the theory of database updates [1], we proposed a formal approach to updates in the weak instance model [9]; coherently with the management of queries, which allows the retrieval of tuples over any subset of the universe, insertions and deletions over any subset of the universe are allowed. Problems of consistency and determinism arise, and have been completely characterized.

EXAMPLE 1.2. *If we want to insert in the state in Figure 1.1 a tuple defined on the attributes* EM, *with values* Jim *for* E *and* White *for* M, *we can consistently add the tuple* $<$ MS, White, C $>$ *to the second relation: because of the dependency* $D \rightarrow M$, *the chase would combine the tuple* $<$ Jim, MS, C $>$, *already in* r_1, *with this tuple, generating a tuple in the representative instance with values* Jim *for* E *and* White *for* M. *Conversely, if we want to insert into the same state a tuple defined on* EPM, *with values* Dan *for* E, D *for* P, *and* Moore *for* M, *we obtain a potential result only if we add one tuple to* r_1 *and one tuple to* r_2, *with the given values for* EPM, *and with the same value for* D *whichever it be (provided that the constraints*

*A preliminary version of this paper appeared in the *Proceedings of the ACM SIGMOD International Conf. on Management of Data*, Atlantic City, May 1990. This work was partially supported by *MURST* and by *Consiglio Nazionale delle Ricerche*.

†Dipartimento di Informatica e Automazione, Università di Roma Tre, Via della Vasca Navale, 84 — 00146 Roma, Italy ({torlone,atzeni}@inf.uniroma3.it)

r_1:

Employee	Dept	Project
John	CS	A
John	CS	B
Bob	EE	B
Jim	MS	C

r_2:

Dept	Manager	Project
CS	Smith	A
IE	White	B
EE	Jones	B
CS	Smith	B

FIG. 1.1. *A database state.*

Employee	Dept	Project	Manager
John	CS	A	Smith
John	CS	B	Smith
Bob	EE	B	Jones
Jim	MS	C	v_1
v_2	CS	A	Smith
v_3	IE	B	White
v_4	EE	B	Jones

FIG. 1.2. *A representative instance.*

Employee	Manager
John	Smith
Bob	Jones

FIG. 1.3. *A total projection of the representative instance.*

are not violated). In this case, some further piece of information has to be added, and there are several possible choices — this is a case of nondeterminism. Finally, an inconsistency arises if we try to insert a tuple on EM with John *for E and* White *for M, since the functional dependencies imply that the only manager of* John *is* Smith.

In the general case, the characterizations for consistency and determinism of insertions require the construction of the representative instance of the state, by means of the application of the chase procedure to a set of data that involves the whole database [9]. However, updating a state often involves only a small part of the database. The case is therefore similar to that of query answering, where the crucial step is the computation of the representative instance, which has to be completely constructed in the general case. To solve this problem, various classes of schemes were introduced, beginning with *independent schemes* [17, 19, 25, 26], where queries can be answered by means of simple relational expressions, optimizable and independent of the actual database state [5, 19, 28, 27]. In this paper we show that a similar approach can be followed for updating as well: we study updates to relational databases through weak instances and show that they can be implemented efficiently.

The paper is organized as follows. In § 2 we briefly review the needed background. In § 3 we review definitions and characterizations of updates in the weak instance model. In § 4 we consider independent schemes, and characterize consistency and determinism with this class of schemes. On the basis of these results, in § 5 we present practical and efficient algorithms for update operations, and in § 6 we show that some step of these algorithms can be simplified under certain further assumptions. In § 7, we discuss about modifications, another important class of database update

operations, and finally, in § 8, we conclude by summarizing our contribution.

2. Background Definitions and Notation.

2.1. Relations, Databases and Tableaux. The *universe* U is a finite set of symbols $\{A_1A_2...A_m\}$, called *attributes*. As usual, we use the same notation A to indicate both the single attribute A and the singleton set $\{A\}$. Also, we indicate the union of attributes (or sets thereof) by means of the juxtaposition of their names. A *relation scheme* is an object $R(X)$, where R is the *name* of the relation scheme and X is a subset of U. A *database scheme* is a collection of relation schemes $\mathbf{R} = \{R_1(X_1), ..., R_n(X_n)\}$, with distinct relation names (which therefore can be used to identify the relation schemes) and such that the union of the X_i's is the universe U.

The *domain* D is the disjoint union of two countably infinite sets, the set of *constants* and the set of *variables* (for the sake of simplicity we assume that all attributes have the same domain). A *tuple* on a set of attributes X is a function t from X to D. If t is a tuple on X, and $Y \subseteq X$, then $t[Y]$ denotes the restriction of the mapping t to Y, and is therefore a tuple on Y. A tuple t on a set of attribute X is *total* if it does not involve variables.

An *tableau* T is a set of tuples on the universe U. We say that a variable or a constant is *unique* in T if it appears only once in it. A *relation* on a relation scheme $R(X)$ is a finite set of total tuples on X. A (*database*) *state* of a database scheme $\mathbf{R}$ is a function $\mathbf{r}$ that maps each relation scheme $R_i(X_i) \in \mathbf{R}$ to a relation on $R_i(X_i)$. With a slight abuse of notation, given $\mathbf{R} = \{R_1, ..., R_n\}$, we write $\mathbf{r} = \{r_1, ..., r_n\}$.

Given a database state $\mathbf{r}$, the *state tableau* for $\mathbf{r}$ is a tableau (denoted by $T_{\mathbf{r}}$) formed by taking the union of all the relations in $\mathbf{r}$ extended to U by means of unique variables.

The *total projection* ($\pi^{\downarrow}$) is an operator on tableaux that produces relations, generating, given a tableau T and a subset X of U, the set of total tuples on X that are restrictions of tuples in T: $\pi^{\downarrow}_X(T) = \{t[X] | t \in T \text{ and } t[X] \text{ is total}\}$. We will use two (orthogonal) generalizations of the total projection: (1) the *restricted total projection* of a tableau T on $X \subseteq U$ with respect to a set of constants C, denoted by $\pi^{\downarrow}_X[C](T)$, is the set of total tuples on X that do not contain constants in C: $\pi^{\downarrow}_X[C](T) = \{t[X] | t \in T, t[X] \text{ is total and, for each } A_i \in X, t[A_i] \notin C\}$; and (2) the *total projection of a tableau* T *on a database scheme* $\mathbf{R}$, denoted by $\pi^{\downarrow}_{\mathbf{R}}(T)$, is the state obtained by totally projecting T on the various relation schemes.

In this paper, we shall consider *relational expression* whose only operators are select (σ), project (π), (natural) join ($\bowtie$) and union ($\cup$), and whose operands are relation schemes of a fixed database scheme. Given a database state $\mathbf{r}$ of a scheme $\mathbf{R}$ and a relational expression E with operands in $\mathbf{R}$, we denote with $E(\mathbf{r})$ the relation obtained by substituting $\mathbf{r}$ into the corresponding relation variables in E, and evaluating E according to the usual definitions of relational operators [7, 20, 29]. The *target* of E is the set of attributes in U on which $E(\mathbf{r})$ is defined.

2.2. Constraints: local satisfaction and global consistency. Associated with a database scheme there is usually a set of *constraints*, that is, properties that are satisfied by the legal states. There are two notions of satisfaction: *local* satisfaction, defined on individual relations, and *global* satisfaction, or *consistency*, defined on the database state. We introduce the two concepts in turn. Various classes of constraints have been defined in the literature [20, 29]; here, we limit our attention to functional dependencies.

Let $YZ \subseteq X \subseteq U$; a relation r on a scheme $R(X)$ *(locally) satisfies* the *functional dependency (FD)* $Y \rightarrow Z$ if, for every pair of tuples $t_1, t_2 \in r$ such that $t_1[Y] = t_2[Y]$, it is the case that $t_1[Z] = t_2[Z]$. Without loss of generality, in the following we will often assume that all FDs have the form $Y \rightarrow A$, where A is a single attribute.

Given a set of FDs, it usually happens that there are additional FDs implied by this set. The *closure* of a set of FDs F, denoted by F^+, is the set of dependencies that are logically implied by F, and the *closure* of a set of attributes X with respect to a set of FDs F, denoted by X_F^+ (or simply X^+ when F is understood from context), is the set of attributes $\{A \mid X \rightarrow A \in F^+\}$. A set of FDs F is said to be *nonredundant* if there is no $Y \rightarrow A \in F$ such that $(F - \{Y \rightarrow A\})^+ = F^+$. Let $R(X)$ be a relation scheme and F be a set of FDs defined on $R(X)$; a set of attributes $Y \subseteq X$ is a *superkey* of R if $Y \rightarrow X \in F^+$.

Let $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ be a database scheme; without loss of generality we associate with $\mathbf{R}$ a set of nonredundant FDs $F = \cup_{i=1}^{n} F_i$, where, for every $1 \leq i \leq n$, the FDs in F_i are defined on $R_i(X_i)$. Note that we assume the FDs to be embedded in relation schemes.

A state $\mathbf{r} = \{r_1, \ldots, \mathbf{r}_n\}$ *globally satisfies* [18] a set of FDs F, if there is a relation w on the universe U (called a *weak instance* for $\mathbf{r}$ with respect to F) that (locally) satisfies F and contains the relations of $\mathbf{r}$ in its projections over the respective relation schemes: $\pi_{X_i}(w) \supseteq r_i$, for $1 \leq i \leq n$. A state that globally satisfies the set of dependencies associated with the database scheme is also said *(globally) consistent*.

2.3. The Chase Procedure. The chase [21] is a procedure that receives as input a tableau T and generates a tableau $CHASE_F(T)$ that, if possible, satisfies the given dependencies F. If only functional dependencies are considered, the process modifies values in the tableau, by equating variables and "promoting" variables to constants as follows. We use $\tau =< t_1, t_2, Y \rightarrow A >$ to denote a *chase step*, with reference to a tableau T, where t_1 and t_2 are tuples in T and $Y \rightarrow A \in F$. We say that a chase step τ is *valid* if $t_1[Y] = t_2[Y]$, and that τ is *applied* to T, denoted by $\tau(T)$, if it is valid and the A–value of t_1 and t_2 are modified as follows: if one of them is a constant and the other is a variable then the variable is changed (is *promoted*) to the constant, otherwise the values are equated. If a chase step tries to identify two constants, then we say that the chase encounters a *contradiction*, and the process stops, generating a special tableau that we call the *inconsistent tableau*, and indicate with T_∞. The tableau $CHASE_F(T)$ is obtained from T by applying all valid chase steps exhaustively to T. Next lemma states a property of the chase that will be used in the following.

LEMMA 2.1. [5, Lemma 4.1] *Let t be a tuple defined on U and $\mathbf{r} = \pi_{\mathbf{R}}^{\downarrow}(\{t\})$. Let $R_{i_0}(X_{i_0}) \in \mathbf{R}$ and t_{i_0} be the tuple in $CHASE_F(T_{\mathbf{r}})$ corresponding to $\pi_{X_{i_0}}(\{t\})$. Then $t_{i_0}[X_{i_0}^+] = t[X_{i_0}^+]$.*

2.4. Query answering in the Weak Instance Model. The definition of global satisfaction is clearly not practical since in general there may be many weak instances (often infinitely many). However, the existence of a weak instance can be studied by means of the notion of representative instance, a tableau on the universe U of the attributes.

The *representative instance* for a state $\mathbf{r}$, indicated with $\mathrm{RI}_{\mathbf{r}}$, is the tableau obtained by chasing the state tableau $T_{\mathbf{r}}$ of $\mathbf{r}$ with respect to the dependencies associated with the database state.

The main property of the representative instance is that a database state is con-

sistent if and only if the corresponding representative instance is not the inconsistent tableau [18]. Also, for every consistent state $\mathbf{r}$ and for every X, the X-total projection of the representative instance of $\mathbf{r}$ is equal to the set of tuples that appear in the projection on X of every weak instance of $\mathbf{r}$ [23].

The *weak instance approach* to query answering allows queries to be formulated on databases as if they were composed of just one relation over the universe U. For each query, being $X \subseteq U$ the set of attributes involved, the evaluation requires a first step that computes the relation over X implied by the current state: for the above consideration, it follows that the X-total projection of the representative instance is the natural content of this relation.

We say that a tuple t over a set of attributes X *x-belongs* (in symbols $t\widehat{\in}\mathbf{r}$) to a consistent state $\mathbf{r}$ of a database scheme $\mathbf{R}$ with a universe $U \supseteq X$ if t belongs to the X-total projection of the representative instance of $\mathbf{r}$.

Finally, the *completion* $\mathbf{r}^*$ of a consistent state $\mathbf{r}$ is the state obtained by projecting the representative instance of $\mathbf{r}$ on the scheme $\mathbf{R}$, that is, $\mathbf{r}^* = \pi^{\downarrow}_{\mathbf{R}}(\mathrm{RI}_{\mathbf{r}})$ [24]. A consistent state $\mathbf{r}$ is *complete* if it coincides with its completion, that is, if $\mathbf{r} = \pi^{\downarrow}_{\mathbf{R}}(\mathrm{RI}_{\mathbf{r}})$.

2.5. Tableau and Relational Expression Containment. A *valuation function* v is a function from D to D that is the identity on constants. A valuation function v can be extended to tuples and tableaux as follows: (i) given a tuple t on a set of attributes X, $v(t)$ is a tuple on X such that $v(t)[A] = v(t[A])$, for every $A \in X$; (ii) given a tableau $T = \{t_1, ..., t_n\}$, $v(T) = \{v(t_1), ..., v(t_n)\}$.

Given two tableaux T_1, T_2, we say that T_1 is *contained* in T_2 (in symbols $T_1 \leq T_2$) if T_2 is the inconsistent tableau T_∞, or there is a valuation function ψ (called in this case *containment mapping*) defined on all the symbols appearing in T_1 such that $\psi(T_1) \subseteq T_2$.[1] If both $T_1 \leq T_2$ and $T_2 \leq T_1$, the two tableau are *equivalent*. Note that, by definition, the inconsistent tableau properly contains every other tableau. We now recall some useful properties of tableaux and chase.

LEMMA 2.2. [9, Lemma 1] *For every tableaux T, T_1 and for every set F of FDs the following statements hold:*

1. $T \leq \mathit{CHASE}_F(T)$;
2. *if* $T \leq T_1$, *then* $\mathit{CHASE}_F(T) \leq \mathit{CHASE}_F(T_1)$;
3. *if* $T \leq T_1$ *and* $T_1 = \mathit{CHASE}_F(T_1)$, *then* $\mathit{CHASE}_F(T) \leq T_1$.

2.6. Independent Schemes. A scheme is *independent* [17, 25] if, for all its states, local satisfaction implies global satisfaction. Independent schemes are clearly important from the practical point of view, because the global consistency of their states can be verified in a local manner, looking at the individual relations, without having to build and chase the state tableau. Graham and Yannakakis [17] derived an efficient test for independence (later improved by other authors [19, 27]).

Independent schemes are also important in the weak instance approach to query answering, because they guarantee the efficient computation of total projection of the representative instance [5, 19]. Atzeni and Chan [5], Ito et al. [19], and Sagiv [28] showed that for every independent scheme and for every subset X of its universe, there is a relational algebra expression E_X that computes the total projection of the representative instance for every state of the scheme. In the approach of Atzeni and Chan, E_X is a union of *simple chase join expression* (scje's) [5, 4, 13], a restricted form of project-join expression, that we recall next.

[1] Note that tableau containment is defined in the opposite direction when refers to containment of queries.

A preliminary concept is needed: a *derivation sequence (ds)* of some relation scheme $R_{i_0}(X_{i_0})$ is a finite sequence of FDs $\sigma =< Y_1 \rightarrow Z_1, ..., Y_m \rightarrow Z_m >$ from F such that, for all $1 \leq j \leq m$: $Y_j \subseteq X_{i_0}Z_1...Z_{j-1}$ and $Z_j \cap X_{i_0}Z_1...Z_{j-1} = \emptyset$. We say that σ *covers* a set of attributes X if $X_{i_0}Z_1...Z_j \supseteq X$. Essentially, a ds of $R_{i_0}(X_{i_0})$ is a sequence of FDs used in computing (part of) the closure of X_{i_0}.

Given a ds $\sigma =< Y_1 \rightarrow Z_1, ..., Y_m \rightarrow Z_m >$ covering a set of attributes X, the *simple chase join expressions (scje) for* σ *over* X is the expression:

$$\pi_X(R_{i_0} \bowtie \pi_{Y_1Z_1}(R_{i_1}) \bowtie ... \bowtie \pi_{Y_mZ_m}(R_{i_m}))$$

where $Y_j \rightarrow A_j$, for $1 \leq j \leq m$, is an FD in F_{i_j}, and is therefore embedded in R_{i_j}. The subexpressions $R_{i_0}, \pi_{Y_1Z_1}(R_{i_1}), \ldots, \pi_{Y_mZ_m}(R_{i_m})$ are called the *components* of the scje.

Before closing this section, we mention an important property of independent schemes that will be often used in the sequel: each derived value in the chase of $T_{\mathbf{r}}$ is "uniquely" derived for an independent scheme.

LEMMA 2.3. [16, Lemma 4] *Let* $\mathbf{R}$ *be an independent scheme. Then for any consistent state* $\mathbf{r}$*, for any* $R_j \in \mathbf{R}$ *and for any FD* $Y \rightarrow A \in F_j$*, it is the case that* $\pi^{\downarrow}_{YA}(CHASE_F(T_{\mathbf{r}})) = \pi_{YA}(r_j)$.

3. Updating in the Weak Instance Model. In this section we briefly review our approach to updates in the weak instance model [9]. Similarly to the approach to query answering, it allows updates to be formulated on every subset of the universe. As a preliminary tool, we introduce a partial order on states, which extends a known notion of equivalence of states [24], then we discuss insertions, and finally deletions.

3.1. A lattice on states. A state $\mathbf{r}_1$ is *weaker* than a state $\mathbf{r}_2$ ($\mathbf{r}_1 \preceq \mathbf{r}_2$) if every weak instance of $\mathbf{r}_2$ is also a weak instance of $\mathbf{r}_1$. Two states $\mathbf{r}_1$, $\mathbf{r}_2$ are *equivalent* ($\mathbf{r}_1 \sim \mathbf{r}_2$) if both $\mathbf{r}_1 \preceq \mathbf{r}_2$ and $\mathbf{r}_2 \preceq \mathbf{r}_1$. The relation $\preceq$ is a partial order on the set of the complete states, since it is reflexive, antisymmetric, and transitive. Also, it is strongly related to (tableau) containment of representative instances, and (set) containment of total projections, and relations, as stated in the next theorem.

THEOREM 3.1. [9, Theorem 1] *Let* $\mathbf{r}_1 = \{r_{1,1}, \ldots, r_{1,n}\}$ *and* $\mathbf{r}_2 = \{r_{2,1}, \ldots, r_{2,n}\}$ *be two states. Properties 1, 2, and 3 below are equivalent; if the states are complete, then 4 is also equivalent to the others.*

1. *The state* $\mathbf{r}_1$ *is weaker than the state* $\mathbf{r}_2$: $\mathbf{r}_1 \preceq \mathbf{r}_2$.
2. *The representative instance of* $\mathbf{r}_1$ *is contained in the representative instance of* $\mathbf{r}_2$: $\mathrm{RI}_{\mathbf{r}_1} \leq \mathrm{RI}_{\mathbf{r}_2}$.
3. *For every* $X \subseteq U$*, the* X*-total projection of* $\mathrm{RI}_{\mathbf{r}_1}$ *is a subset of the* X*-total projection of* $\mathrm{RI}_{\mathbf{r}_2}$: $\pi^{\downarrow}_X(\mathrm{RI}_{\mathbf{r}_1}) \subseteq \pi^{\downarrow}_X(\mathrm{RI}_{\mathbf{r}_2})$.
4. *The state* $\mathbf{r}_1$ *is a relationwise subset of the state* $\mathbf{r}_2$: *for every* $R_i \in \mathbf{R}$*, it is the case that* $r_{1,i} \subseteq r_{2,i}$.

By the equivalence of part 1 and part 3 of theorem above it follows that two states are equivalent if and only if, for every X, their X-total projections are equal, that is if they have identical query answering behavior. Therefore, it makes sense to consider equivalence classes of states. As representatives of the various classes, we will use the set of complete states since it is known that each consistent state is equivalent to one and only one complete state [24, §3].

In [9] we showed that the partial order $\preceq$ extended to the *complete inconsistent state* (a special state defined as the projection of the inconsistent tableau on the database scheme) induces a complete lattice [12] on the set of complete states, that

is, every set of complete states has both a greatest lower bound (glb) and a least upper bound (lub).

3.2. Insertions. Let $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ be a database scheme, with $U = X_1X_2 \ldots X_n$. Given a state $\mathbf{r}$ of $\mathbf{R}$ and a tuple t over a set of attributes $X \subseteq U$, we consider the *insertion* of t into $\mathbf{r}$ defined through the following notion of result.

A state $\mathbf{r}_p$ is a *potential result* for the insertion of t into $\mathbf{r}$ if $\mathbf{r} \preceq \mathbf{r}_p$ and $t\widehat{\in}\mathbf{r}_p$. Various cases for an insertion of a tuple in a consistent state exist: the insertion of a tuple t over X in a state is *possible* if there is a consistent state $\mathbf{r}'$ such that $t\widehat{\in}\mathbf{r}'$, a possible insertion is *consistent* if it has a consistent potential result, and a possible and consistent insertion is *deterministic* if the glb of the potential results is a potential result. Note that the notion of determinism is defined only for possible and consistent insertions. When an insertion is deterministic, we consider the glb of the potential results as *the* result of the insertion. In plain words, an insertion is possible if the dependencies allow us to generate t in the representative instance of a state $\mathbf{r}'$ possibly unrelated to the given state, it is consistent when the new tuple does not contradict the information content of the original state, and it is deterministic when the insertion can be univocally performed by adding only the information that is strictly needed.

In [9] we showed the following general characterizations for possibility, consistency, and determinism.

THEOREM 3.2. [9, Theorem 2] *The insertion of t in a state is possible if and only if there is a relation scheme $R_i(X_i) \in \mathbf{R}$ such that F implies the FD $X_i \rightarrow X$.*

Let $\mathrm{RI}_{\mathbf{r}}$ be the representative instance of $\mathbf{r}$. The characterization of both consistency and determinism is based on the construction of a tableau obtained by adding to $\mathrm{RI}_{\mathbf{r}}$ a tuple t_e obtained by extending t to the universe U by means of unique variables. Let $T_{t,\mathbf{r}}$ be such a tableau.

THEOREM 3.3. [9, Theorem 3] *Let the insertion of t in $\mathbf{r}$ be possible. It is consistent if and only if $\mathit{CHASE}_F(T_{t,\mathbf{r}}) \neq T_\infty$.*

DEFINITION 3.4 (State $\mathbf{r}_{+t}$). *The state $\mathbf{r}_{+t}$, or simply $\mathbf{r}_+$ when t is understood from context, is obtained from $\mathbf{r}$ and t by (totally) projecting $\mathit{CHASE}_F(T_{t,\mathbf{r}})$ on the database scheme: $\mathbf{r}_{+t} = \pi^{\downarrow}_{\mathbf{R}}(\mathit{CHASE}_F(T_{t,\mathbf{r}}))$.*

LEMMA 3.5. [9, Lemma 8] *Let the insertion of t in $\mathbf{r}$ be possible and consistent. Then $\mathbf{r}_+$ is the glb of the potential results.*

THEOREM 3.6. [9, Theorem 4] *Let the insertion of t in $\mathbf{r}$ be possible and consistent. It is deterministic if and only if $\mathit{CHASE}_F(T_{t,\mathbf{r}}) \equiv \mathrm{RI}_{\mathbf{r}_+}$.*

COROLLARY 3.7. [9, Corollary 1] *Let the insertion of t in $\mathbf{r}$ be possible and consistent; it is deterministic if and only if $t\widehat{\in}\mathbf{r}_+$.*

Corollary 3.7 gives an effective characterization of determinism: given $\mathbf{r}$ and t, we can build $T_{t,\mathbf{r}}$, chase it with respect to the given constraints, then generate $\mathbf{r}_+$ and compute its representative instance $\mathrm{RI}_{\mathbf{r}_+}$, and finally check whether the total projection $\pi^{\downarrow}_X(\mathrm{RI}_{\mathbf{r}_+})$ contains t.

EXAMPLE 3.1. *Consider the first insertion in Example 1.2. The insertion is possible since for $R_1(X_1)$ we have $X_1 \rightarrow X \in F^+$. Then, following the definitions, we could build the tableau $T_{t,\mathbf{r}}$ and then chase it: the tableaux we obtain are reported in Figure 3.1. It is possible to see that, if we project $\mathit{CHASE}_F(T_{t,\mathbf{r}})$ on the database scheme, we obtain the state we suggested as a result.*

Note that the insertion of a tuple on a set of attributes $X \subseteq U$ allows a form of "side-effect", since it may cause the addition of some extra-information for the attributes not in X. For instance, in the above example, we have shown that the insertion of the tuple with values *Jim* for E and *White* for M produces the insertion

$T_{t,\mathbf{r}}$

Employee	Dept	Project	Manager
John	CS	A	Smith
John	CS	B	Smith
Bob	EE	B	Jones
Jim	MS	C	v_1
v_2	CS	A	Smith
v_3	IE	B	White
v_4	EE	B	Jones
Jim	v_5	v_6	White

$CHASE_F(T_{t,\mathbf{r}})$

Employee	Dept	Project	Manager
John	CS	A	Smith
John	CS	B	Smith
Bob	EE	B	Jones
Jim	MS	C	White
v_2	CS	A	Smith
v_3	IE	B	White
v_4	EE	B	Jones
Jim	MS	v_5	White

r_{+_1}:

Employee	Dept	Project
John	CS	A
John	CS	B
Bob	EE	B
Jim	MS	C

r_{+_2}:

Dept	Manager	Project
CS	Smith	A
IE	White	B
EE	Jones	B
CS	Smith	B
MS	White	C

FIG. 3.1. $T_{t,\mathbf{r}}$, $CHASE_F(T_{t,\mathbf{r}})$ *and* $\mathbf{r}_+$ *for Example 3.1.*

of the tuple $<$ MS, White, C $>$ in r_2: this tuple states that *White* is the manager of the *MS* department and is involved into the project C, and these are information not directly provided by the user. This fact is however just a consequence of the weak instance model framework in which the FDs allows us to derive, in the representative instance, further information from tuples of a database. Thus, the insertion of a new tuple (over a relation or any set of attributes) may induce new values for old tuples in the representative instance. However, with our approach, this side-effect is always kept minimal since we have defined the result of an insertion as the glb of all the potential results: in this way the original state is always changed as little as possible. We will come back on this issue in § 6, where we will show that, under certain conditions, the side-effect can also be kept "under control".

3.3. Deletions. The definitions are somehow symmetric with respect to those concerning insertions.

A state $\mathbf{r}_p$ is a *potential result* for the deletion of a tuple t from a state $\mathbf{r}$ if $\mathbf{r}_p \preceq \mathbf{r}$ and $t \widehat{\notin} \mathbf{r}_p$. The empty state is a consistent potential result for every deletion, and so there is no need to define the notions of possible and consistent results for deletions. A deletion is *deterministic* if the lub of the potential results is a potential result. When a deletion is deterministic, we consider the lub of the potential results as *the* result of

the deletion.

Let $\mathbf{r}$ be a consistent state and t be a tuple on X that x-belongs to $\mathbf{r}$. We derived the following characterizations for deletions.

LEMMA 3.8. [9, Lemma 9] *The deletion of t from $\mathbf{r}$ is deterministic only if there is a relation scheme $R_i(X_i)$ such that $X \subseteq X_i$.*

DEFINITION 3.9 (State $\mathbf{r}_{-t}$). *The state $\mathbf{r}_{-t}$, or simply $\mathbf{r}_-$ when t is understood from context, is obtained from $\mathbf{r}$ and t by removing, from each relation r_i such that $X \subseteq X_i$, each tuple t' such that $t'[X] = t[X]$.*

THEOREM 3.10. [9, Theorem 5] *The deletion of t from $\mathbf{r}$ is deterministic if and only if* (i) *there is a relation scheme $R_i(X_i)$ such that $X \subseteq X_i$ and* (ii) $t \widehat{\notin} \mathbf{r}_-$.

4. Insertions for Independent Schemes. In the same way as query answering can be efficiently performed for independent schemes, we want to show that, for this meaningful class of schemes, updates defined over any subset of the universe can be managed efficiently.

With respect to deletions, Theorem 3.10 already gives an efficient way for checking for determinism and for performing the update. With respect to insertions, the problem is more complex in general. Regarding to possibility, Theorem 3.2 gives a complete characterization at the scheme level, which can be verified very efficiently by using the closure algorithm proposed by Bernstein [11], but with regard to consistency and determinism, the tests require the chase of $T_{t,\mathbf{r}}$ (Theorems 3.3 and 3.6), a tableau involving the whole database state. Since computing the chase of a tableau takes polynomial time in the number of the rows of the tableau [2], it follows that, the tests for consistency and determinism of an insertion require time and size polynomial with respect to the size of the database state.

In this section we show that it is possible to derive alternative methods for checking consistency and determinism of insertions to independent schemes, that are easier to implement and optimize.

4.1. Consistency. Throughout this section we will consider a consistent state $\mathbf{r}$ on a scheme $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ and the insertion of a tuple t over $X \subseteq U$ in $\mathbf{r}$, assuming it is possible.

Let $\overline{t}$ be the "extension" of t with respect to $\mathbf{r}$ generated by Algorithm 4.1 (shown in Figure 4.1). It turns out that $\overline{t}$ has interesting properties, which make it fundamental in the efficient check of both consistency and determinism.

Let us introduce a property to be used shortly.

CONDITION 4.1. *There is no $V \rightarrow A \in F_j$, for $j \in \{1, \ldots, n\}$, such that:*

(i) *$VA \subseteq \overline{X}$, and*

(ii) *there exists $t' \in r_j$ such that $t'[V] = \overline{t}[V]$ and $t'[A] \neq \overline{t}[A]$*

LEMMA 4.1. *If Condition 4.1 holds then $\overline{t}$ is uniquely defined.*

Proof. Condition 4.1 implies that when at any step there are two or more alternatives, those not chosen remain valid after the transformation, and therefore can be later applied. □

DEFINITION 4.2 (State $\tilde{\mathbf{r}}$). *Let $\overline{t}$ be the extension of t with respect to $\mathbf{r}$, and $\tilde{t}$ be a tuple over the universe U obtained by extending $\overline{t}$ to U by means of "new constants", that is, a unique constants not already appearing in $\mathbf{r}$. Then, the state $\tilde{\mathbf{r}}$ is obtained from $\mathbf{r}$ and $\tilde{t}$ by adding, to each relation $r_j \in \mathbf{r}$ on $R_j(X_j)$, the tuple $\tilde{t}[X_j]$.*

LEMMA 4.3. *If $\mathbf{R}$ is independent and Condition 4.1 holds, then $\tilde{\mathbf{r}}$ is a consistent potential result for the insertion of t in $\mathbf{r}$.*

ALGORITHM 4.1.
Input : *a tuple* t *over* $X \subseteq U$ *and a database state* $\mathbf{r}$;
Output: *the "extension"* $\overline{t}$ *of* t *with respect to* $\mathbf{r}$ *and the set of attributes* $\overline{X}$;
begin
$t_U[X] := t$; /* t_U *is a tuple of distinct variables over* U */
$W := X$;
while *there exists some* $V \to A \in F_j$, $1 \leq j \leq n$, *such that* $V \subseteq W, A \notin W$
and *there exists* $t' \in r_j$ *such that* $t'[V] = t_U[V]$
do begin
$t_U[A] := t'[A]$;
$W := W \cup A$
end;
return $t_U[W]$ *and* W
end.

FIG. 4.1. *Algorithm for the computation of the extension of a tuple.*

Proof. We have to show that (1) $\tilde{\mathbf{r}}$ is consistent, (2) $\tilde{\mathbf{r}}$ is a potential result.
(1) $\tilde{\mathbf{r}}$ is consistent: since $\mathbf{R}$ is independent, it is sufficient to show that $\tilde{\mathbf{r}}$ is locally consistent. By way of contradiction assume that it is not. Let $\tilde{r}_j$ be a relation that violates the respective FDs F_j. Since $\mathbf{r}$ is consistent, r_j satisfies F_j, and so the violation has to involve the new tuple $\tilde{t}[X_j]$ together with a tuple $t' \in r_j$; that is, there is an FD $V \to A \in F_j$ such that $t'[V] = \tilde{t}[V]$ and $t'[A] \neq \tilde{t}[A]$. Since the values of $\tilde{t}$ over attributes in $U - \overline{X}$ are all new constants, $\tilde{t}[V] = t'[V]$ implies $V \subseteq \overline{X}$ and so $\overline{t}[V] = t'[V]$. Then, the computation of $\overline{t}$ would also add A to W and so to $\overline{X}$ because of $t' \in r_j$, and therefore $\tilde{t}[A] = \overline{t}[A]$. Then, we would have $t'[V] = \overline{t}[V]$ and $t'[A] \neq \overline{t}[A]$, for some $V \to A \in F_j$ with $VA \subseteq \overline{X}$, against Condition 4.1.
(2) $\tilde{\mathbf{r}}$ is a potential result: by construction $\mathbf{r} \preceq \tilde{\mathbf{r}}$. Also, since the insertion of t in $\mathbf{r}$ is possible, by Theorem 3.2, there is a relation scheme $R_i(X_i) \in \mathbf{R}$ such that F implies the FD $X_i \to X$, and so, $X_i^+ \supseteq X$. Let t' be the tuple in $CHASE_F(T_{\tilde{\mathbf{r}}})$ originating from $\pi_{X_i}(\tilde{t})$. Then, by Lemma 2.1, $t'[X_i^+] = \tilde{t}[X_i^+]$, and since $X_i^+ \supseteq X$, by definition of $\tilde{t}$ we have $t'[X] = t$. It follows that $t \in \pi_X(\mathrm{RI}_{\tilde{\mathbf{r}}})$, and so $\tilde{\mathbf{r}}$ is a potential result. □

THEOREM 4.4. *The insertion of* t *in a state* $\mathbf{r}$ *of an independent scheme is consistent if and only if Condition 4.1 holds.*

Proof. (Only if) The construction of $\overline{t}$ can be seen as an initial sequence in the chase of $T_{t,\mathbf{r}}$ with respect to F. Then, violation of Condition 4.1 implies that $CHASE_F(T_{t,\mathbf{r}}) = T_\infty$ and so, by Theorem 3.3, the claims follows.
(If) By Lemma 4.3. □

This theorem gives us an effective and efficient method to check for consistency of insertions in independent schemes: instead of performing the chase of $T_{t,\mathbf{r}}$ (as requested by Theorem 3.3), it is sufficient to apply Algorithm 4.1 and to check for violations of FDs involving $\overline{t}$.

EXAMPLE 4.1. *The scheme in Example 1.1 is clearly independent. Now consider the first insertions in Example 1.2. We have* $\overline{t} =<$ Jim, MS, White $>$, *so the insertion is consistent, since such a tuple does not violate the FDs that involves. Conversely, if we want to insert the tuple* $<$ John, White $>$, *we would have an inconsistent insertion since* $\overline{t} =<$ John, CS, White $>$, *and Condition 4.1 does not hold for the FD* $D \to M$ *and, for instance, the tuple* $<$ CS, Smith, A $>$ *of* r_2.

4.2. Determinism. Throughout this section we will consider a consistent database state $\mathbf{r}$ of a scheme $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ and the insertion of a tuple t over $X \subseteq U$ in $\mathbf{r}$, assuming it is possible and consistent.

As in § 4.1, let $\overline{t}$ be the extension of t with respect to $\mathbf{r}$ generated by Algorithm 4.1, $\tilde{t}$ be the tuple obtained by extending $\overline{t}$ to the universe U by means of a set of new constants C_{new}, and $\tilde{\mathbf{r}}$ be the state obtained by adding to the original state $\mathbf{r}$ the projections of $\tilde{t}$ over the various relation schemes.

DEFINITION 4.5 (State $\hat{\mathbf{r}}$). *Let $\tilde{\mathbf{r}}^*$ be the completion of $\tilde{\mathbf{r}}$, that is, $\tilde{\mathbf{r}}^* = \pi^{\downarrow}_{\mathbf{R}}(\mathrm{RI}_{\tilde{\mathbf{r}}})$. Then, $\hat{\mathbf{r}}$ is the state obtained from $\tilde{\mathbf{r}}^*$ by removing all the tuples having some new constant: $\hat{\mathbf{r}} = \pi^{\downarrow}_{\mathbf{R}}[C_{new}](\mathrm{RI}_{\tilde{\mathbf{r}}})$*

We have the following result for the state $\hat{\mathbf{r}}$ (we recall that $\mathbf{r}_+ = \pi^{\downarrow}_{\mathbf{R}} CHASE_F(T_{t,\mathbf{r}})$, where $T_{t,\mathbf{r}} = \mathrm{RI}_{\mathbf{r}} \cup \{t_e\}$ and t_e is the tuple obtained by extending t to the universe U by means of unique variables).

THEOREM 4.6. *The state $\hat{\mathbf{r}}$ coincides with the state $\mathbf{r}_+$.*

Proof. By Lemma 4.3, $\tilde{\mathbf{r}}$ is a potential result and therefore, since by Lemma 3.5 the state $\mathbf{r}_+$ is the glb of the potential result, we have $\mathbf{r}_+ \preceq \tilde{\mathbf{r}}$ and so $\mathbf{r}_+ \preceq \tilde{\mathbf{r}}^*$ as every state is equivalent to its completion. Since $\mathbf{r}_+$ is complete (being constructed as the projection of a chased tableau), by equivalence of parts 1 and 4 of Theorem 3.1, we have that $\mathbf{r}_+$ is a relationwise subset of $\tilde{\mathbf{r}}^*$, and so of $\hat{\mathbf{r}}$ which is obtained from $\tilde{\mathbf{r}}^*$ by eliminating tuples that do not appear in $\mathbf{r}_+$. Thus, to complete the proof we need to show that for every $R_i \in \mathbf{R}$, it is also the case that $\hat{r}_i \subseteq r_{+_i}$, where $\hat{r}_i \in \hat{\mathbf{r}}$ and $r_{+_i} \in \mathbf{r}_+$. We will prove this part by showing that it is possible to build a tableau T, such that: $T \leq CHASE_F(T_{t,\mathbf{r}})$ and $\hat{\mathbf{r}} = \pi^{\downarrow}_{\mathbf{R}}(T)$. Since $\mathbf{r}_+ = \pi^{\downarrow}_{\mathbf{R}}(CHASE_F(T_{t,\mathbf{r}}))$, the fact that $\hat{\mathbf{r}}$ is a relationwise subset of $\mathbf{r}_+$ would then follow directly by the definitions of containment mapping and total projection.

Let $T_1 = \mathrm{RI}_{\mathbf{r}} \cup \{\overline{t}_e\}$ where $\overline{t}_e$ is obtained by extending $\overline{t}$ to the universe U by means of unique variables. Since $\overline{t}$ is built using chase steps that are valid in $T_{t,\mathbf{r}}$ and since the chase is independent of the order of the individual chase steps [21], it follows that T_1 can be seen as an intermediate result in chasing $T_{t,\mathbf{r}}$ and that $CHASE_F(T_1) = CHASE_F(T_{t,\mathbf{r}})$. Similarly, let $\mathbf{r}_{\tilde{t}}$ be the state obtained by projecting $\tilde{t}$ over the various relation schemes and let $T_2 = \mathrm{RI}_{\mathbf{r}} \cup CHASE_F(T_{\mathbf{r}_{\tilde{t}}})$: since, by construction, $T_{\tilde{\mathbf{r}}} = T_{\mathbf{r}} \cup T_{\mathbf{r}_{\tilde{t}}}$, we have that T_2 can be considered as an intermediate result in chasing $T_{\tilde{\mathbf{r}}}$, and that $CHASE_F(T_2) = CHASE_F(T_{\tilde{\mathbf{r}}}) = \mathrm{RI}_{\tilde{\mathbf{r}}}$. The tableau T_2 contains all the tuples of $\mathrm{RI}_{\mathbf{r}}$ and, by Lemma 2.1, n tuples t_i such that $t_i[X_i^+] = \tilde{t}[X_i^+]$, for $1 \leq i \leq n$. Then, let ϕ be a function from D to D that (i) maps the new constants in $\tilde{t}$ to unique variables in T_2 and (ii) is the identity on all the other elements in D, and consider the tableau $T_3 = \phi(T_2)$. This tableau is composed by the tuples of $\mathrm{RI}_{\mathbf{r}}$ and by n tuples t'_i such that: $t'_i[A] = \overline{t}[A]$ if $A \in \overline{X}$, and $t'_i[A]$ is a variable otherwise. It easily follows that $T_3 \leq T_1$ because of a containment mapping that is the identity on $\mathrm{RI}_{\mathbf{r}}$ and maps the n tuples t'_i to $\overline{t}_e$. Now, let us consider the chase of T_2 and let ψ be the function that maps each symbol appearing in T_2 to the symbols to which it is change by the chase, that is, $\psi(T_2) = CHASE_F(T_2)$, and let $\psi' = \phi \circ \psi$. The function ψ' coincides with ψ except for the variables v that have been changed to new constants by chasing T_2, that is, $\psi'(v) = \phi(\psi(v)) = v_j$ if $\psi(s)$ is a new constants c_j such that $\phi(c_j) = v_j$, and $\psi'(s) = \psi(s)$ for all the other symbols s in T_2. Since all the chase steps that can be applied to T_2 are also valid in the tableau T_3 if we replace the new constants with unique variables, it follows that $\psi'(T_2)$ coincides with the chase of T_3, that is, $\psi'(T_2) = CHASE_F(T_3)$. Let $T_4 = \phi(\mathrm{RI}_{\tilde{\mathbf{r}}})$: we have $T_4 = \phi(\psi(T_2)) = \psi'(T_2) = CHASE_F(T_3)$, and therefore, since $T_3 \leq T_1$, by part 2 of Lemma 2.2, we have

$CHASE_F(T_3) \leq CHASE_F(T_1)$, and so $T_4 \leq CHASE_F(T_1) = CHASE_F(T_{t,\mathbf{r}})$.

Now, since ϕ maps new constants to new variables and is the identity on the other elements of D, we have that $\pi^{\downarrow}_{\mathbf{R}}(T_4)$ exactly coincides with $\hat{\mathbf{r}}$. As argued above, it follows that $\hat{\mathbf{r}}$ is a relationwise subset of $\mathbf{r}_+ = \pi^{\downarrow}_{\mathbf{R}}(CHASE_F(T_{t,\mathbf{r}}))$. □

COROLLARY 4.7. *The insertion of a tuple t in a state $\mathbf{r}$ is deterministic if and only if $t \widehat{\in} \hat{\mathbf{r}}$.*

Proof. It follows by Corollary 3.7 and Theorem 4.6. □

Corollary 4.7 gives us an alternative method to check for determinism that does not requires the chase of the special tableau $T_{t,\mathbf{r}}$ over the whole database.

EXAMPLE 4.2. *Consider again the insertion of $t =<$ Jim, White $>$ over EM in the state in Figure 1.1. We have in this case $\overline{t} =<$ Jim, MS, White $>$ over EDM and $\tilde{t} =<$ Jim, MS, c_1, White $>$ over U. Hence, $\tilde{\mathbf{r}}$ is obtained by adding $<$ Jim, MS, $c_1 >$ to r_1 and $<$ MS, White, $c_1 >$ to r_2. In computing the completion of $\tilde{\mathbf{r}}$, the tuple $<$ MS, White, C $>$ is also added to the second relation and so, by deleting the tuples with the new constant c_1, we obtain again the state we suggest as the result.*

In the following section we will show how the new method can be efficiently implemented if the scheme is independent. For this purpose, we now mention some properties of $T_{t,\mathbf{r}}$ and $\hat{\mathbf{r}}$ for the class of independent schemes. We recall that given a tuple t over a set of attributes $X \subseteq U$, the tuple t_e denotes its extension to U by means of unique variables.

LEMMA 4.8. *If $\mathbf{R}$ is independent then for any $R_j \in \mathbf{R}$ and for any $Y \rightarrow A \in F_j$ it is the case that $\pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}})) = \pi_{YA}(r_j) \cup \pi^{\downarrow}_{YA}(\{\overline{t}_e\})$.*

Proof. It is sufficient to prove that $\pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}})) \subseteq \pi_{YA}(r_j) \cup \pi^{\downarrow}_{YA}(\{\overline{t}_e\})$ as the other containment, by construction, is trivial. Let us consider the state $\tilde{\mathbf{r}}$: since, by Lemma 4.3, it is a consistent potential result, we have that $\mathbf{r} \preceq \tilde{\mathbf{r}}$ and $t \widehat{\in} \tilde{\mathbf{r}}$. Hence, $\mathrm{RI}_{\mathbf{r}} \leq \mathrm{RI}_{\tilde{\mathbf{r}}}$ and $t \in \pi^{\downarrow}_{X}(\mathrm{RI}_{\tilde{\mathbf{r}}})$ and therefore, since all the variables in t_e are unique, $T_{t,\mathbf{r}} = \mathrm{RI}_{\mathbf{r}} \cup \{t_e\} \leq \mathrm{RI}_{\tilde{\mathbf{r}}}$, and so, by part 3 of Lemma 2.2, it follows that $CHASE_F(T_{t,\mathbf{r}}) \leq CHASE_F(T_{\tilde{\mathbf{r}}})$. Now let $t' \in \pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}}))$. By equivalence of parts 2 and 3 of Lemma 3.1 we have $\pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}})) \subseteq \pi^{\downarrow}_{YA}(CHASE_F(T_{\tilde{\mathbf{r}}}))$, hence, $t' \in \pi^{\downarrow}_{YA}(CHASE_F(T_{\tilde{\mathbf{r}}}))$. Since $\mathbf{R}$ is independent and so, by Lemma 2.3, $\pi^{\downarrow}_{YA}(CHASE_F(T_{\tilde{\mathbf{r}}})) = \pi_{YA}(\tilde{r}_j)$, it follows that $t' \in \pi_{YA}(\tilde{r}_j)$. Now, by definition of $\tilde{\mathbf{r}}$, we have that $t' \in \pi_{YA}(r_j)$ or $t' = \tilde{t}[YA]$. In the first case the proof is complete; in the second we have two subcases: (i) $YA \subseteq \overline{X}$ and so $\tilde{t}[YA] = \overline{t}[YA]$, which would again prove the claim, and (ii) $YA \not\subseteq \overline{X}$ and therefore $\tilde{t}[YA]$ contains constants not appearing in $CHASE_F(T_{t,\mathbf{r}})$ and so $\tilde{t}[YA] \notin \pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}}))$; this, however, contradicts $\tilde{t}[YA] = t' \in \pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}}))$. □

LEMMA 4.9. *If $\mathbf{R}$ is independent and the insertion of t in $\mathbf{r}$ is deterministic then for any $R_i(X_i)$ and for any $Y \rightarrow A \in F_i$, it is the case that $\pi_{YA}(\hat{r}_i) = \pi_{YA}(r_i) \cup \pi^{\downarrow}_{YA}(\{\overline{t}_e\})$.*

Proof. If the insertion is deterministic then, by Theorem 3.6, we have that $\mathrm{RI}_{\mathbf{r}_+} \equiv CHASE_F(T_{t,\mathbf{r}})$, and so, by Theorem 4.6 and by equivalence of parts 1 and 2 of Theorem 3.1, $\mathrm{RI}_{\hat{\mathbf{r}}} \equiv CHASE_F(T_{t,\mathbf{r}})$. Also, since $\mathbf{R}$ is independent, by Lemma 2.3 we have that for any $R_i(X_i) \in \mathbf{R}$ and for any FD $X \rightarrow A \in F_i$, it is the case that $\pi_{YA}(\hat{r}_i) = \pi^{\downarrow}_{YA}(\mathrm{RI}_{\hat{\mathbf{r}}})$. By equivalence of parts 2 and 3 of Theorem 3.1, it follows that $\pi_{YA}(\hat{r}_i) = \pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}}))$, and so, by Lemma 4.8, we have $\pi_{YA}(\hat{r}_i) = \pi_{YA}(r_i) \cup \pi^{\downarrow}_{YA}(\{\overline{t}_e\})$. □

ALGORITHM 5.1.
Input: *a state* $\mathbf{s}$, *a tuple* t_Y *over* $Y \subseteq U$, *a set of attributes* $V \supseteq Y$, *a set of constants* C;
Output: *a relation* s_{out};
begin
  let E_V *be the AC-expression for* V;
  repeat
    select a scje $E_i = \pi_V(R_{i_0} \bowtie \pi_{Y_1 Z_1}(R_{i_1}) \bowtie \ldots \bowtie \pi_{Y_m Z_m}(R_{i_m}))$ *from* E_V;
    $s^{(0)} := \{t_Y\} \bowtie s_{i_0}$; /* $s_{i_0} \in \mathbf{s}$ *on* R_{i_0} */
    $W := Y \cup X_{i_0}$;
    $k := 0$;
    repeat
      select a component $\pi_{Y_j Z_j}(R_{i_j})$ *from* E_i *such that* $W \supseteq Y_j$, *choosing first those such that* $Y \cap Y_j Z_j \neq \emptyset$ (*if any*);
      $s^{(k+1)} := s^{(k)} \bowtie \pi_{Y_j Z_j}(s_{i_j})$ /* $s_{i_j} \in \mathbf{s}$ *on* R_{i_j} */;
      $W := W \cup Y_j Z_j$;
      $k := k + 1$;
    until ($s^{(k)} = \emptyset$) **or** (*all the components of* E_i *have been selected*);
    $s_{out} := \pi_V(s^{(k)}) - \{$*tuples with constants in* $C\}$;
  until ($s_{out} \neq \emptyset$) **or** (*all scje's of* E_V *have been selected*);
  return s_{out}
end.

FIG. 5.1. *Basic algorithm for update operations.*

5. Algorithms for Update Operations. On the basis of the results of the previous sections, we present in this section efficient algorithms that can be used to update relational databases on independent schemes. Such algorithms exploit some known result on query answering for this class of schemes. In particular, we will use the results of Atzeni and Chan [5] who proved that, for independent schemes, the total projection of the representative instance can be obtained by means of a union of scje's and proposed an algorithm to compute and optimize this expression, which runs in polynomial time with respect to the size of the database scheme. In the following, we will denote with $E_X = \cup E_i$ the union of scje's for a set of attributes $X \subseteq U$ [5, Algorithm 5.8], and we will refer to E_X as the *AC-expression* for X.

5.1. Basic algorithm. In this subsection we provide an algorithm (reported in Figure 5.1) that will be used for several purposes in the sequel, and whose aim is to test efficiently whether certain total tuple appears in the representative instance of a state defined on an independent scheme. More specifically, given a consistent state $\mathbf{s}$ of an independent scheme $\mathbf{R}$, a tuple t_Y over $Y \subseteq U$, a set of attributes $V \subseteq U$ such that $Y \subseteq V$ and a set of constants C, Algorithm 5.1 verifies whether there exists total tuples on V in $\mathrm{RI}_{\mathbf{s}}$, without constants in C, that coincide with t_Y on the attributes Y, that is, whether $\sigma_{Y=t_Y}(\pi_V^{\downarrow}[C](\mathrm{RI}_{\mathbf{s}}))$ is not empty. If $Y = V$ and $C = \emptyset$, the algorithm simply tests whether $t_Y \widehat{\in} \mathbf{s}$. The role of the set of attributes V and of the set of constants C will be clarified later.

The search is efficiently performed by joining t_Y with tuples of $\mathbf{s}$ on the basis of the scje's in the AC-expression E_V for V, giving precedence to the components having some attribute in Y. This corresponds in performing selections on the values in t_Y as

early as possible while computing $\sigma_{Y=t_Y}(E_V(\mathbf{r}))$.

In the inner loop of the algorithm the expression $\sigma_{Y=t_Y}(E_i(\mathbf{r}))$ for a scje $E_i \in E_V$ is evaluated. The loop halts as soon as the result turns out to be empty or when the scje has been completely computed. Then, the derived tuples having no constants in C are stored in the relation s_{out}. The algorithm stops as soon as s_{out} is not empty or when all the scje's in E_V have been examined. Thus, at termination, we have that s_{out} contains tuples of $\sigma_{Y=t_Y}(E_V(\mathbf{s}))$ without constants in C, and that $s_{out} \neq \emptyset$ if and only if $\sigma_{Y=t_Y}(E_V(\mathbf{s}))$ contains at least one tuple without constants in C. Since in [5] it is proved that a tuple $t' \in \pi^{\downarrow}_V(\mathrm{RI}_\mathbf{s})$ if and only if $t' \in E_V(\mathbf{s})$, the following result easily follows.

LEMMA 5.1. *Assume that Algorithm 5.1 receives as input a tuple t_Y over $Y \subseteq U$, a consistent state $\mathbf{s}$ of a scheme $\mathbf{R}$, a set of attributes $V \subseteq U$ such that $Y \subseteq V$ and a set of constants C. Then, at termination: (1) $s_{out} \subseteq \pi^{\downarrow}_V[C](\sigma_{Y=t_Y}(\mathrm{RI}_\mathbf{s}))$, and (2) $s_{out} \neq \emptyset$ if and only if $\pi^{\downarrow}_V[C](\sigma_{Y=t_Y}(\mathrm{RI}_\mathbf{s})) \neq \emptyset$.*

COROLLARY 5.2. *Let $Y = V$ and $C = \emptyset$, then the output relation s_{out} of Algorithm 5.1 is not empty if and only if $t_Y \widehat{\in} \mathbf{s}$.*

Note that, for efficiency purposes, Algorithm 5.1 does not compute a complete total projection of the representative instance. Note also that if $\pi^{\downarrow}_V(\sigma_{Y=t_Y}(\mathrm{RI}_\mathbf{s}))$ is not empty, then s_{out} is not deterministic, since it depends on the order in which the scje's have been selected. However, this is not important since, as we will see, we just need a relation satisfying the above properties.

5.2. Performing insertions. Let $\mathbf{r}$ be a consistent state of a scheme $\mathbf{R}$, and consider the insertion of a tuple t over $X \subseteq U$ in $\mathbf{r}$, assuming that it is possible and consistent. In § 4 it has been shown that the property of determinism for insertions can be verified on the state $\hat{\mathbf{r}}$. The construction of this state requires the computation of the completion of the state $\tilde{\mathbf{r}}$. We will show now that we do not need to compute the full completion of $\tilde{\mathbf{r}}$, since, as suggested by Lemma 4.9, it is sufficient to find only those tuples without new constants of $\hat{\mathbf{r}}^*$, that coincide with $\overline{t}$ on the attributes involved in some FD.

More specifically, let us consider the following database state. Again, $\overline{t}$ denotes the extension of t with respect to $\mathbf{r}$, $\tilde{t}$ the tuple obtained by extending $\overline{t}$ to the universe U by means of new constants, $\tilde{\mathbf{r}}$ the state obtained by adding to the original state $\mathbf{r}$ the projections of $\tilde{t}$ over the various relation schemes and $\tilde{\mathbf{r}}^*$ its completion.

DEFINITION 5.3 (State $\check{\mathbf{r}}$). *A state $\check{\mathbf{r}}$ is obtained from $\mathbf{r}$ and $\tilde{\mathbf{r}}$: (1) by adding, to each relation $r_i \in \mathbf{r}$ on $R_i(X_i)$ such that $\overline{X} \supseteq X_i$, the tuple $\overline{t}[X_i]$, and (2) by adding, to each relation $r_j \in \mathbf{r}$ over $R_j(X_j)$ such that, $\overline{X} \not\supseteq X_j$ and $\overline{X} \supseteq YA$ for some $Y \rightarrow A \in F_j$, at least one tuple without new constants $t_j \in \tilde{r}^*_j$, such that $t_j[YA] = \overline{t}[YA]$ and $t_j \notin r_j$.*

An important point here is that, if the deterministic condition is satisfied, in case (2) above there must exists at least one tuple without new constants t_j in $\tilde{r}^*_j$ such that $t_j[YA] = \overline{t}[YA]$ and $t_j \notin r_j$. This follows from the fact that, by Lemma 4.9, for each $Y \rightarrow A \in F_j$, $\pi_{YA}(\hat{r}_j) = \pi_{YA}(r_i) \cup \pi^{\downarrow}_{YA}(\{\overline{t}_e\})$, and the fact that $\hat{r}_j$ is obtained from $\tilde{r}^*_j$ by just deleting tuples with new constants. Note also that, in general, several tuples may satisfy this property.

We have the following results for $\check{\mathbf{r}}$.

LEMMA 5.4. *If $\mathbf{R}$ is independent and the insertion of t in $\mathbf{r}$ is deterministic then $\check{\mathbf{r}} \sim \hat{\mathbf{r}}$.*

Proof. We have to show that for deterministic insertions: (1) $\hat{\mathbf{r}} \preceq \check{\mathbf{r}}$ and (2) $\check{\mathbf{r}} \preceq \hat{\mathbf{r}}$.

(1) $\hat{\mathbf{r}} \preceq \check{\mathbf{r}}$. We prove this part by showing that, for every $R_i(X_i) \in \mathbf{R}$ and for every $t_i \in \hat{r}_i$, either (i) $t_i \in \check{r}_i$, or (ii) $t_i \in \pi^{\downarrow}_{X_i}(\mathrm{RI}_{\tilde{\mathbf{r}}})$. It would follow that every $t_i \in \hat{r}_i$ belongs to the relation $\check{r}^*_i$ in the completion $\check{\mathbf{r}}^*$ of $\check{\mathbf{r}}$. Since, by Theorem 4.6, $\hat{\mathbf{r}}$ coincides with a complete state and is therefore itself complete, the fact that $\hat{\mathbf{r}} \preceq \check{\mathbf{r}}$ then follows by the equivalence of parts 1 and 4 of Theorem 3.1.

So, let $t_i \in \hat{r}_i$ for some $R_i(X_i) \in \mathbf{R}$. By construction of $\hat{\mathbf{r}}$, we have three possible cases: (1) $t_i \in r_i$ where $r_i \in \mathbf{r}$, that is, t_i belongs to a relation of the original state, (2) $t_i \notin r_i$ and $\overline{X} \supseteq X_i$, and so $t_i = \overline{t}[X_i]$, and (3) $t_i \notin r_i$ and $\overline{X} \not\supseteq X_i$, and so t_i has been generated in chasing $T_{\tilde{\mathbf{r}}}$. In the first two cases we also have $t_i \in \check{r}_i$ by construction. With respect to the third case, we will show that t_i can be generated by chasing $T_{\tilde{\mathbf{r}}}$.

Thus, let t_i be a tuple of $\hat{r}_i$ on $R_i(X_i) \in \mathbf{R}$, such that $t_i \notin r_i$ and $\overline{X} \not\supseteq X_i$. Since, by Theorem 4.6, $\hat{\mathbf{r}} = \mathbf{r}_+$, we have that, by construction of $\mathbf{r}_+$, the tuple t_i is also generated by chasing $T_{t,\mathbf{r}}$. Moreover, since the insertion is deterministic, by Theorem 3.6, $CHASE_F(T_{t,\mathbf{r}})$ is equivalent to the representative instance of a database state. It follows that t_i originates in $CHASE_F(T_{t,\mathbf{r}})$ from a tuple t_0 of a relation $r_0 \in \mathbf{r}$ (indeed, t_0 can not originate from $\overline{t}$ since this would imply that $t_i = \overline{t}[X_i]$ and so $\overline{X} \supseteq X_i$). Let $\chi = \tau_1, ..., \tau_m$ be the sequence of chase steps that allows us to generate t_i from t_0 in the chase of $T_{t,\mathbf{r}}$. Since $\mathbf{R}$ is independent, by Lemma 4.8, each τ_k in χ promotes a variable to a constant in $\mathbf{r}$ or in t, because of a tuple t_j and an FD $Y \rightarrow A$, such that $t_j[YA] \in \pi_{YA}(r_j) \cup \pi^{\downarrow}_{YA}(\{\overline{t}_e\})$.

Now, for each $Y \rightarrow A \in F_j$ such that $\overline{X} \supseteq YA$, if $\overline{X} \supseteq X_j$, the tuple $\overline{t}[X_i]$ is in $\check{r}_j$ by construction. Moreover, if $\overline{X} \supseteq YA$ and $\overline{X} \not\supseteq X_j$, we have argued above that, for deterministic insertions, there must exists at least one tuple without new constants t'_j in $\tilde{r}^*_j$ such that $t'_j[YA] = \overline{t}[YA]$ and so, by construction, a tuple satisfying this property is also in $\check{r}_j$. It follows that the above sequence of chase step χ is also valid in $T_{\tilde{\mathbf{r}}}$, that is, it allows us to generate t_i from t_0 (which, by construction, belongs to $\check{\mathbf{r}}$), by chasing $T_{\tilde{\mathbf{r}}}$. As argued above, this concludes part (1) of the proof.
(2) $\check{\mathbf{r}} \preceq \hat{\mathbf{r}}$: it easily follows by their definitions. □

THEOREM 5.5. *For independent schemes the insertion of t in $\mathbf{r}$ is deterministic if and only if $t \widehat{\in} \check{\mathbf{r}}$.*

Proof. (Only if) If the insertion is deterministic then, by Corollary 4.7, $t \widehat{\in} \hat{\mathbf{r}}$, and, by Lemma 5.4, $\check{\mathbf{r}} \sim \hat{\mathbf{r}}$. It follows that $t \widehat{\in} \check{\mathbf{r}}$.
(If) Since, by construction, $\check{\mathbf{r}} \preceq \hat{\mathbf{r}}$, if $t \widehat{\in} \check{\mathbf{r}}$ then it is also the case that $t \widehat{\in} \hat{\mathbf{r}}$ and therefore, by Corollary 4.7, the insertion is deterministic. □

A state $\check{\mathbf{r}}$ can be efficiently derived from $\mathbf{r}$ and $\tilde{\mathbf{r}}$ by using Algorithm 5.2, reported in Figure 5.2. In this algorithm, C_{new} denotes the set of new constants used in the construction of $\tilde{t}$, and, for each $R_i(X_i) \in \mathbf{R}$, $Y^+_{F_i}$ denotes the *local closure* of a set of attributes $Y \subseteq X_i$, with respect to F_i. The following lemma confirms the correctness of the algorithm.

LEMMA 5.6. *Assume that Algorithm 5.2 receives as input a state $\mathbf{r}$ of an independent scheme $\mathbf{R}$, a tuple t over $X \subseteq U$ and its extensions $\overline{t}$ and $\tilde{t}$. Then, the output of the algorithm is a state $\check{\mathbf{r}}$.*

Proof. Since the tuples added to $\mathbf{r}_{out}$ in step (a) of Algorithm 5.2 belong to $\check{\mathbf{r}}$ by definition, it is sufficient to show that in step (b), only tuples satisfying condition (2) of the definition of $\check{\mathbf{r}}$, are added to $\mathbf{r}_{out}$, and nothing else. First, note that if there is a tuple $t_i \in \tilde{r}_i$ such that $t_i[YA] = \overline{t}[YA]$, for some $Y \rightarrow A$ in F_i, it easily follows that $\overline{t}$ is defined on $Y^+_{F_i}$ and $t_i[Y^+_{F_i}] = \overline{t}[Y^+_{F_i}]$. Then, by Lemma 5.1, in step (b) of Algorithm 5.2, we add to $\mathbf{r}_{out}$ at least one tuple (if any) without new constants in $\pi^{\downarrow}_{X_i}[C_{new}](\sigma_{Y^+_{F_i} = \overline{t}[Y^+_{F_i}]}(\mathrm{RI}_{\tilde{\mathbf{r}}}))$. It follows that, for each relation $r_j \in \mathbf{r}$ over $R_j(X_j) \in \mathbf{R}$

ALGORITHM 5.2.
Input : $\mathbf{r}$, t *over* $X \subseteq U$, $\overline{t}$ *over* $\overline{X}$, $\tilde{t}$ *over* U;
Output: *a state* $\check{\mathbf{r}}$;
begin
  $\mathbf{r}_{out} := \mathbf{r}$;
  for each $R_i(X_i) \in \mathbf{R}$
  do if $\overline{X} \supseteq X_i$
(a) **then** $r_{out_i} := r_{out_i} \cup \{\overline{t}[X_i]\}$
  else for each $Y \rightarrow A \in F_i$ *such that* $\overline{X} \supseteq YA$
    do if *does not exist* $t' \in r_{out_i}$ *such that* $t'[YA] = \overline{t}[YA]$
      then begin
        execute Algorithm 5.1 over $\tilde{\mathbf{r}}$, $\overline{t}[Y_{F_i}^+]$, X_i *and* C_{new};
(b) $r_{out_i} := r_{out_i} \cup s_{out}$
      end;
  return $\mathbf{r}_{out}$
end.

FIG. 5.2. *Algorithm for the generation of a state* $\check{\mathbf{r}}$.

such that, $\overline{X} \not\supseteq X_j$ and $\overline{X} \supseteq YA$ for some $Y \rightarrow A \in F_j$, we add, in step (b) of Algorithm 5.2, at least one tuple $t_j \in \tilde{r}_j^*$ without new constants, such that: $t_j[YA] = \overline{t}[YA]$ and $t_j \notin r$, and nothing else. □

EXAMPLE 5.1. *Let* $\mathbf{r}$ *be the state in Figure 1.1 and* $t =<$ Jim, White $>$ *over* EM. *We have seen, in Example 4.1, that* $\overline{t} =<$ Jim, MS, White $>$ *over* $\overline{X} = EDM$. *Let* $\tilde{t} =<$ Jim, MS, c_1, White $>$ *and consider the execution of Algorithm 5.2 over these inputs. We have* $\overline{X} \not\supseteq X_1 = EDP$ *and* $\overline{X} \supseteq ED$ *where* $E \rightarrow D \in F_1$, *but* $\overline{t}[ED] \in \pi_{ED}(r_1)$ *and therefore* $r_{out_1} = r_1$. *We then have* $\overline{X} \not\supseteq X_2 = DMP$, $\overline{X} \supseteq DM$ *for the functional dependency* $D \rightarrow M \in F_2$, *and* $\overline{t}[DM] \notin \pi_{DM}(r_2)$. *Thus, in this case, Algorithm 5.1 needs to be executed with* $\tilde{\mathbf{r}}$, $<$ MS, White $>$, DMP *and* $\{c_1\}$ *as inputs. We have:* $E_{DMP} = R_2 \cup \pi_{DMP}(R_1 \bowtie \pi_{DM}(R_2))$. *For the first scje in* E_{DMP} *we obtain* $s_{out} = \emptyset$, *since* $\overline{t}[DM] \bowtie \tilde{r}_2 = \{<$ MS, c_1, White $>\}$. *For the second scje we have:* $\pi_{DMP}(\overline{t}[DM] \bowtie \tilde{r}_1 \bowtie \pi_{DM}(\tilde{r}_2)) = \{<$ MS, C, White $>, <$ MS, c_1, White $>\}$. *Hence, we obtain:* $r_{out_2} = r_2 \cup \{<$ MS, C, White $>\}$. *Thus, in this case,* $\check{\mathbf{r}}$ *coincides with* $\hat{\mathbf{r}}$ *(see Example 4.2) and* $t \widehat{\in} \check{\mathbf{r}}$ — *this confirms the determinism of the insertion of* t *in* $\mathbf{r}$.

We are now ready to give an algorithm (reported in Figure 5.3) summarizing all phases of insert operations to independent schemes.

Step (1) of Algorithm 5.3 checks for possibility (Theorem 3.2). This test requires the computation of closures of sets of attributes, an operation that can be performed in time $O(\|F\|)$, where $\|F\|$ is the size of the description of F, by using Bernstein algorithm [10]. Since the closure has to be performed $|\mathbf{R}|$ times in the worst case, where $|\mathbf{R}|$ is the number of relation schemes in $\mathbf{R}$, it follows that testing for possibility is bounded by $O(|\mathbf{R}| \times \|F\|)$.

In step (2) the extension $\overline{t}$ of t with respect to the state $\mathbf{r}$ is computed using Algorithm 4.1. This algorithm requires the computation of the closure of a set of attributes and, at each step of the computation, the selection of a tuple given a value on the left hand side of an FD. The selection time depends on the cardinality of the relation, but it can be strongly reduced by defining indexes on the left hand side of all the FDs in F. Let $k_{F,\mathbf{r}}$ be the maximum time needed to search for tuples in a

ALGORITHM 5.3.
Input : $\mathbf{r}$ *and* t *over* $X \subseteq U$;
Output: "not possible" | "not consistent" | "not deterministic" |
the result of the insertion of t *in* $\mathbf{r}$;
begin
(1) **if** *not exists* $R_i(X_i) \in \mathbf{R}$ *such that* $X_i^+ \supseteq X$
then return "not possible" **and stop**;
(2) *compute* $\overline{t}$ *and* $\overline{X}$ *using Algorithm 4.1;*
(3) **if** *Condition 4.1 does not hold*
then return "not consistent" **and stop**;
(4) *compute* $\check{\mathbf{r}}$ *using Algorithm 5.2;*
(5) *verify that* $t \widehat{\in} \check{\mathbf{r}}$ *using Algorithm 5.1;*
(6) **if** $s_{out} = \emptyset$
then return "not deterministic"
else return $\check{\mathbf{r}}$
end.

FIG. 5.3. *Algorithm for the execution of insert operations.*

database state $\mathbf{r}$, given a value on the left hand side of an FD in F. The cost of step (2) is then proportional to $\|F\| \times k_{F,\mathbf{r}}$. Under the same hypothesis, Condition 4.1 in step (3) (which, by Theorem 4.4, corresponds in checking for possibility) can be tested in time $O(|F| \times k_{F,\mathbf{r}})$, where $|F|$ is the cardinality of the set of FDs F. This is because in an independent scheme each FD is embedded in at most one relation scheme [17].

In step (4) the state $\check{\mathbf{r}}$ is computed with Algorithm 5.2. This algorithm requires, in the worst case, the execution of Algorithm 5.1 a number of times that, by the above property of independent schemes is bounded by $|F|$. For any given X, there are at most as many scje's in E_X as relation schemes in $\mathbf{R}$ [5, 3]. It follows that the execution of Algorithm 5.1 corresponds to the execution of a number of relational algebra expressions bounded by $|F'| \times |\mathbf{R}|$, where F' denotes the FDs $Y' \rightarrow A'$ in F, such that there is no other FD $Y \rightarrow A$ in some $F_i \subseteq F$, for which $Y_{F_i}^+ \supseteq Y'A'$. The cost of each expression is bounded by $|r_{max}| \times k_{F,\mathbf{r}} \times |F|$, where $|r_{max}|$ is the size of the largest relation in $\mathbf{r}$. In fact, the computation starts by a relation not larger than $|r_{max}|$, and then performs a number of joins, bounded by $|F|$, that simply require, for each tuple in the intermediate relation, the search for a tuple in a relation, given a value for its key (which is the lhs of and FD): this is because, at each step, the attributes of the intermediate relation include the lhs of the FD over which each component is projected. Note also that the size of the computed relation is always bounded by $|r_{max}|$. In sum, step (4) is bounded by: $|r_{max}| \times |F|^2 \times |\mathbf{R}| \times k_{F,\mathbf{r}}$. On the average however, it turns out that the cost of this operation is quite limited. This is because: (i) updates often involve only a very small portion of FDs in F, (ii) relational expressions are optimized as described in [5] and selections are performed as early as possible, and (iii) the number of scje's for a set of attributes is small when the scheme enjoys the desirable property of "independent updatability" [16].

Finally, step (5) corresponds in testing for determinism (Theorem 5.5 and Corollary 5.2) and requires the execution of Algorithm 5.1 once more. Note that the state $\tilde{\mathbf{r}}$ as well as the state $\check{\mathbf{r}}$ do not need to be effectively constructed. So, Algorithms 5.1 and 5.2 could be slightly modified in order to work on the tuples in $\mathbf{r}$, $\tilde{\mathbf{r}}$ and $\check{\mathbf{r}}$, without

ALGORITHM 5.4.
Input : $\mathbf{r}$ *and* t *over* $X \subseteq U$;
Output: "not deterministic" | *the result of the deletion of* t *from* $\mathbf{r}$;
begin
(1) **if** *not exists* $R_i(X_i) \in \mathbf{R}$ *such that* $X_i \supseteq X$
    then return "not deterministic" **and stop**;
(2) **for each** $R_i(X_i) \in \mathbf{R}$ **do if** $X_i \supseteq X$
    then $r_{-_i} := r_i - \sigma_{X=t}(r_i)$;
(3) *verify that* $t \widehat{\in} \mathbf{r}_-$ *using Algorithm 5.1*;
(4) **if** verified
    then return "not deterministic" **and stop**
    else return $\mathbf{r}_-$
end.

FIG. 5.4. *Algorithm for the execution of delete operations.*

actually adding tuples to $\mathbf{r}$.

By the discussion above, it turns out that Algorithm 5.3 provides a practical and efficient way to perform the insertion of a tuple t over any set of attributes $X \subseteq U$ in a state of an independent scheme.

EXAMPLE 5.2. *In Examples 4.1 and 5.1 it is reported the execution of all the steps of Algorithm 5.3 for the tuple* $t =< \text{Jim}, \text{White} >$ *over EM and the state in Figure 1.1.*

5.3. Performing deletions. By the results of the previous sections is also possible to give an efficient method for performing delete operations on a database state of an independent scheme. This algorithm is reported in Figure 5.4.

In step (1), the necessary condition for determinism of Lemma 3.8 is tested: it requires time proportional to $|\mathbf{R}|$. Then, in step (2), the state $\mathbf{r}_-$ is computed by executing a number of selections that is again bounded by $|\mathbf{R}|$. Then, by Theorem 3.10 and Corollary 5.2, step (3) corresponds in testing for determinism. This requires one execution of Algorithm 5.1 which is optimized as discussed in the previous subsection. Also in this case, the algorithm can be slightly modified in order to work on the tuples $\mathbf{r}$ and $\mathbf{r}_-$ without actually deleting tuples from $\mathbf{r}$.

Thus, again, given a state $\mathbf{r}$ of an independent scheme and a tuple t over any set of attributes $X \subseteq U$, Algorithm 5.4 provides a practical and efficient way to perform the deletion of t from $\mathbf{r}$.

EXAMPLE 5.3. *Assume we want to delete the tuple* $t =< \text{Smith}, \text{B} >$ *over MP from the state in Figure 1.1. The condition in step (1) of Algorithm 5.4 is verified for the scheme* $R_2(DMP)$*. Then, the state* $\mathbf{r}_-$ *is obtained in step (2) by deleting the tuple* $< \text{CS}, \text{Smith}, \text{B} >$ *from* r_2*. However, it is easy to see that this tuple can be reconstructed from the tuple* $< \text{John}, \text{CS}, \text{B} >$ *in* r_1 *and the tuple* $< \text{CS}, \text{Smith}, \text{A} >$ *in* r_2*. It follows that in step (3) we obtain* $t \widehat{\in} \mathbf{r}_-$ *and therefore the insertion is not deterministic. Conversely, the deletion of* $t =< \text{White}, \text{B} >$ *over the same attributes is deterministic since in this case we would have* $t \widehat{\notin} \mathbf{r}_-$.

6. Possible Simplifications of the Algorithms. In this section we show that under certain further assumptions, update operations can be managed easier.

6.1. Insertions. We recall that a database scheme $\mathbf{R}$ is *separable* if it is independent and every consistent state on $\mathbf{R}$ is complete [16] (Chan and Mendelzon also

provided an efficient test for separability).

Now, let $\mathbf{r}$ be a state of a database scheme $\mathbf{R}$, t be a tuple over $X \subseteq U$ and $\overline{t}$ be the extension of t with respect to $\mathbf{r}$. Let us consider the following state.

DEFINITION 6.1 (State $\overline{\mathbf{r}}$). *The state $\overline{\mathbf{r}}$ is obtained from $\mathbf{r}$ and $\overline{t}$ by adding the tuple $\overline{t}[X_j]$ to each relation $r_j \in \mathbf{r}$ on $R_j(X_j) \in \mathbf{R}$ such that $X_j \subseteq \overline{X}$.*

THEOREM 6.2. *If $\mathbf{R}$ is separable then $\overline{\mathbf{r}} = \mathbf{r}_+$.*

Proof. If $\mathbf{R}$ is separable, then the state $\tilde{\mathbf{r}}$ coincides with its completion $\tilde{\mathbf{r}}^*$, and therefore, by construction, $\hat{\mathbf{r}}$ can be obtained by eliminating from $\tilde{\mathbf{r}}$ the tuples with new constants. The state we obtain clearly coincides with $\overline{\mathbf{r}}$, and therefore, by Theorem 4.6, the claim follows. □

COROLLARY 6.3. *The insertion of a tuple t in a state $\mathbf{r}$ on a separable scheme is deterministic if and only if $t \widehat{\in} \overline{\mathbf{r}}$.*

Proof. It follows by Corollary 3.7 and Theorem 6.2. □

By this result, the test for determinism and the computation of the minimum result for insertions to separable schemes can be done more efficiently since in this case it is not required to compute the state $\check{\mathbf{r}}$ as it suffices to refer to the state $\overline{\mathbf{r}}$ which can be easily generated.

EXAMPLE 6.1. *The scheme of the state $\mathbf{r}$ in Figure 1.1 is independent but not separable since, for instance, the state obtained by deleting the tuple $<$ CS, Smith, B $>$ from r_2 is not complete (see Example 5.3). Therefore, we have in general $\overline{\mathbf{r}} \neq \mathbf{r}_+$. In fact, for to the tuple $t =<$ Jim, White $>$ over EM, we have that $\overline{t} =<$ Jim, MS, White $>$ and so $\overline{\mathbf{r}} = \mathbf{r}$, whereas we have shown that the insertion of t in $\mathbf{r}$ is deterministic. It is easy to show that, if the second relation would contain only the attributes D and P, the scheme was separable: in this case the insertion above could be performed by adding to r_2 the tuple $<$ MS, White $>$, which is indeed embedded in $\overline{t}$.*

The state $\overline{\mathbf{r}}$ has an interesting property: it contains only the tuples that can be derived directly from t, by extending this tuple with values from tuples of $\mathbf{r}$ using the FDs in F. Then, by simply computing the extension of t with Algorithm 4.1, we immediately know not only the tuples to insert to the original database, but also the "side-effect" generated by the insertion. Therefore, when $\overline{\mathbf{r}}$ is the result of the insertion, we can keep the side-effect under control. Unfortunately, as shown in the example above, even for independent schemes, $\overline{\mathbf{r}}$ is not always the correct result, and insertion operations require, in the general case, a more involved computation, as described in § 5.

Interestingly however, it is possible to give for independent schemes "local" conditions at scheme level (which therefore can be efficiently tested) that allow us to refer to the state $\overline{\mathbf{r}}$ for insert operations even for schemes that are nonseparable. Let us consider the following property which refer to the insertion of a tuple t over $X \subseteq U$ in a state $\mathbf{r}$ of a scheme $\mathbf{R}$.

CONDITION 6.1. *For every relation scheme $R_i(X_i) \in \mathbf{R}$, at least one of the following holds:*

(i) $X_i \subseteq \overline{X}$,

(ii) $YA \not\subseteq \overline{X}$ *for any* $Y \rightarrow A \in F_i$,

(iii) F_i *contains an FD whose left hand side is a superkey of* X_i.

THEOREM 6.4. *Let $\mathbf{R}$ be independent and assume that Condition 6.1 holds. Then, the insertion of t in $\mathbf{r}$ is deterministic if and only if $t \widehat{\in} \overline{\mathbf{r}}$.*

Proof. (Only if) We prove this part by showing that if, for an independent scheme, Condition 6.1 holds and the insertion is deterministic, then for every $R_i(X_i) \in \mathbf{R}$

such that $X_i \not\subseteq \overline{X}$ it is the case that $\hat{r}_i = r_i$. The fact that $t \widehat{\in} \overline{\mathbf{r}}$ would then follow by Theorem 5.5 and the fact that in this case, by construction, $\check{\mathbf{r}} = \overline{\mathbf{r}}$. So, by way of contradiction, assume that $\mathbf{R}$ is independent, Condition 6.1 holds and there is a scheme $R_i(X_i) \in \mathbf{R}$ such that $X_i \not\subseteq \overline{X}$ and $\hat{r}_i \neq r_i$. By definition of $\hat{\mathbf{r}}$, this implies that: (i) $YA \subseteq \overline{X}$, for some $Y \rightarrow A \in F_i$, (ii) there is no tuple in r_i which coincides with $\overline{t}$ on YA, and (iii) there is a tuple $t' \in \tilde{r}_i^*$ without new constants such that $t'[YA] = \overline{t}[YA]$. But because of Condition 6.1, if $X_i \not\subseteq \overline{X}$ and $YA \subseteq \overline{X}$, for some $Y \rightarrow A \in F_i$, then there must exist an FD $Z \rightarrow B \in F_i$ such that Z is a superkey of X_i. Now, since the scheme is independent, by Lemma 2.3, $\pi^{\downarrow}_{ZB}(CHASE_F(T_{\check{\mathbf{r}}})) = \pi_{ZB}(\tilde{r}_i)$, and since Z is a superkey, this implies $\pi^{\downarrow}_{X_i}(CHASE_F(T_{\check{\mathbf{r}}})) = \tilde{r}_i$ and therefore $t' = \tilde{t}[X_i]$. This, in turn, implies that $t' = \overline{t}[X_i]$, since t' does not contain new constants, and therefore $X_i \subseteq \overline{X}$ — a contradiction.
(If) If $t \widehat{\in} \overline{\mathbf{r}}$ then it is also the case that $t \widehat{\in} \check{\mathbf{r}}$ since, by construction, $\overline{\mathbf{r}} \preceq \check{\mathbf{r}}$. Then, the claim follows by Theorem 5.5. □

EXAMPLE 6.2. *Consider again the insertion of the tuple* $t =<$ Jim, White $>$ *over EM in the state* $\mathbf{r}$ *in Figure 1.1: in this case Condition 6.1 is not verified since* $\overline{X} = EDM$ *and, for* $R_1(X_1)$*, we have:* $X_1 \not\subseteq \overline{X}$*,* $E \rightarrow D \in F_1$ *and is embedded in* $\overline{X}$*, and the lhs of* $E \rightarrow D$ *does not contain a superkey of* X_1*. On the other hand, if we consider the insertion of* $t =<$ Jim, D $>$ *over EP in the same state, we obtain* $\overline{t} =<$ Jim, MS, D $>$*, and since in this case Condition 6.1 holds (*$X_1 \subseteq \overline{X}$ *and* $YA \not\subseteq \overline{X}$ *for any* $Y \rightarrow A \in F_2$*), the result of this insertion can be obtained by simply adding* $\overline{t}$ *to the first relation.*

Testing for Condition 6.1 on an independent scheme requires in the worst case time proportional to $|F| \times \|F\|$, since, as we have said before, each FD is embedded in at most one relation scheme. This test can be performed after step (3) of Algorithm 5.3, and if it succeeds, then steps (4)–(6) of Algorithm 5.3 can be substituted by just one step verifying, by using Algorithm 5.1, that $t \widehat{\in} \overline{\mathbf{r}}$.

6.2. Deletions. With respect to deletions, we can test for determinism more efficiently if it is the case that every piece of information that is defined on some subset of a relation scheme is explicitly represented in the database. This is the property of the *embedded-complete* schemes [15]: a scheme $\mathbf{R}$ is *embedded-complete* if for any consistent state $\mathbf{r}$ of $\mathbf{R}$ and for any $X \subseteq U$, such that there exists $R_i(X_i) \in \mathbf{R}$ with $X_i \supseteq X$, it is the case that $\pi^{\downarrow}_X(\mathrm{RI}_{\mathbf{r}}) = \bigcup_{X_j \supseteq X} \pi_X(r_j)$.

Let us consider the deletion of any tuple over a set of attributes $X \subseteq U$ from a state $\mathbf{r}$ of the scheme $\mathbf{R}$.

THEOREM 6.5. *The deletion of a tuple* t *from a state* $\mathbf{r}$ *on an embedded-complete scheme is deterministic if and only if there is a relation scheme* $R_i(X_i)$ *such that* $X_i \supseteq X$*.*

Proof. (Only if) By Lemma 3.8.
(If) This part follows by Theorem 3.10 and the fact that, if the database scheme is embedded-complete, then no tuple over a set of attributes which is contained in a relation scheme can be generated by the chase from other tuples, and so it is never the case that $t \widehat{\notin} \mathbf{r}_-$. □

Also in this case, it is possible to state local conditions at scheme level for independent schemes that allow us to test for determinism of a deletion of a tuple t over a set of attributes $X \subseteq U$ as follows.

CONDITION 6.2. *For every relation scheme* $R_i(X_i)$ *in* $\mathbf{R}$*, at least one of the following holds:*

(i) $X \not\subseteq X_i$,

(ii) *for every relation scheme* $R_j(X_j)$ *in* $\mathbf{R}$, $i \neq j$, $X \not\subseteq X_j^+$,

(iii) F_i *contains an FD whose left hand side is a superkey of* X.

THEOREM 6.6. *Let* $\mathbf{R}$ *be independent and assume that Condition 6.2 holds. Then, the deletion of* t *from* $\mathbf{r}$ *is deterministic if and only if there is a relation scheme* $R_i(X_i)$ *such that* $X \subseteq X_i$.

Proof. (Only if) By Lemma 3.8.

(If) Assume by way of contradiction that, for an independent scheme $\mathbf{R}$, Condition 6.2 holds and there is a relation scheme $R_i(X_i) \in \mathbf{R}$ such that $X \subseteq X_i$ but the deletion is not deterministic. By Theorem 3.10 it follows that $t \widehat{\in} \mathbf{r}_-$, and therefore there is a tuple t' in $\mathrm{RI}_{\mathbf{r}_-}$ such that $t'[X] = t$. Then, let t_j be the tuple from which t' originates and assume that $t_j \in r_{-_j}$ over $R_j(X_j)$. Clearly, $X \not\subseteq X_j$ and so $i \neq j$. Moreover, since we have assumed that $t \widehat{\in} \mathbf{r}$, by Theorem 3.2, $X_j \rightarrow X \in F^+$, and so, $X \subseteq X_j^+$. This implies that there is an FD $Y \rightarrow A \in F_i$ such that Y is a superkey of X, otherwise Condition 6.2 would be false. Since $\mathbf{R}$ is independent, by Lemma 2.3, we have $\pi^{\downarrow}_{YA}(\mathrm{RI}_{\mathbf{r}_-}) = \pi_{YA}(r_{-_i})$, and since Y is a superkey of X, we have that $\pi^{\downarrow}_{X}(\mathrm{RI}_{\mathbf{r}_-}) = \pi_X(r_{-_i})$. But, by construction, $t \notin \pi_X(r_{-_i})$ and so it follows that $t \widehat{\notin} \mathbf{r}_-$ — a contradiction. □

EXAMPLE 6.3. *Consider the deletion* $t =< \mathrm{CS}, \mathrm{Smith}, \mathrm{B} >$ *over* $X = DMP$ *from the state* $\mathbf{r}$ *in Figure 1.1. Condition 6.2 does not hold since* $X \subseteq X_2 = DMP$, $X \subseteq X_1^+ = U$, *and the only FD* $D \rightarrow M$ *in* F_2 *has a left hand side that is not a superkey of* X. *Therefore, for deletions defined on EDP we need to check whether* $t \notin \mathbf{r}_-$ *(it turns out that the deletion of* t *is not deterministic as shown in Example 5.3). Note that the scheme of* $\mathbf{r}$ *is not embedded-complete. Conversely, Condition 6.2 is verified for the tuple* $t' =< \mathrm{CS}, \mathrm{Smith} >$ *over* DM *since we have* $DM \subseteq X_2$ *and* $DM \subseteq X_1^+$, *but in this case* D *is a superkey of* DM. *Hence, the deletion of* t' *from* $\mathbf{r}$ *is deterministic and the result of the deletion can be obtained by deleting the tuples* $< \mathrm{CS}, \mathrm{Smith}, \mathrm{A} >$ *and* $< \mathrm{CS}, \mathrm{Smith}, \mathrm{B} >$ *from* r_2.

Condition 6.2 can be tested for independent schemes in time bounded by $\|F\| \times (|\mathbf{R}|^2 + |F|)$ and can be performed after step (2) of Algorithm 5.4. If such a condition is verified then the test for determinism in step (3) is no more necessary as in this case the state $\mathbf{r}_-$ is surely the maximum result.

7. Modification operations. The present paper study insertions and deletions of tuples as basic database update operations. However, modifications form indeed another important class of update operations, often used in practical situations. The goal of this section is to briefly discuss about modifications of tuples: we show that they naturally fit in our framework and that, in general, results on insertions and deletions can be used to characterize them.

A simple modification operation consists in changing the values of a single tuple. Therefore, we can represent a modification of a state $\mathbf{r}$ by means of a pair (t_{old}, t_{new}), where t_{old} and t_{new} are tuples defined over the same set of attributes $X \subseteq U$: the intended meaning of this operation is clearly to substitute t_{old} by t_{new} in $\mathbf{r}$.

According to the definition of insertions and deletions, this operation should be realized by altering the information content of the original state as little as possible. Thus, in the framework we have defined, a modification (t_{old}, t_{new}), defined over any set of attributes $X \subseteq U$, of a database state $\mathbf{r}$ for a scheme $\mathbf{R}$, can be defined through the following notion of result.

A state $\mathbf{r}_p$ is a *potential result* for the modification (t_{old}, t_{new}) to $\mathbf{r}$ if: (1) $t_{old} \widehat{\notin} \mathbf{r}_p$, (2) $t_{new} \widehat{\in} \mathbf{r}_p$, and (3) for every state $\mathbf{r}' \preceq \mathbf{r}$ of $\mathbf{R}$ such that (a) $t_{old} \widehat{\notin} \mathbf{r}'$ and (b) the

insertion of t_{new} in $\mathbf{r}'$ is consistent, it is the case that $\mathbf{r}' \preceq \mathbf{r}_p$.

If we assume that $t_{old} \widehat{\in} \mathbf{r}$, a modification is always possible, but similarly to insertions, inconsistency and nondeterminism may arise. We then say that a modification is *consistent* if it has a consistent potential result, and is *deterministic*, if the glb of the potential results is itself a potential result.

By the definition above, it turns out that a modification can be implemented, in most cases, through a deletion followed by an insertion. Specifically, we can easily show the following results

LEMMA 7.1. *Let* $\mathbf{r}$ *be a database state of a scheme* $\mathbf{R}$*, and* (t_{old}, t_{new}) *be a modification defined over a set of attributes* $X \subseteq U$*. Then, the following properties hold:*

(i) *If the deletion of* t_{old} *from* $\mathbf{r}$ *is deterministic and the insertion of* t_{new} *to* $\mathbf{r}_{-t_{old}}$ *is consistent, then the modification* (t_{old}, t_{new}) *of* $\mathbf{r}$ *is consistent.*

(ii) *If the deletion of* t_{old} *from* $\mathbf{r}$ *is deterministic and the insertion of* t_{new} *to* $\mathbf{r}_{-t_{old}}$ *is deterministic, then the modification* (t_{old}, t_{new}) *of* $\mathbf{r}$ *is deterministic.*

(iii) *If the deletion of* t_{old} *from* $\mathbf{r}$ *is deterministic then* $(\mathbf{r}_{-t_{old}})_{+t_{new}}$ *is the glb of the potential results.*

We note that the converse of the above results does not hold in general. This is shown in the following example.

EXAMPLE 7.1. *Consider the database scheme* $\mathbf{R} = \{R_1(AB), R_2(BC)\}$*, with the FDs* $A \rightarrow B$ *and* $B \rightarrow C$ *defined for it, and the state of* $\mathbf{R}$*:*

$$\mathbf{r} = \{r_1 = \{< 1, 2 >\}, r_2 = \{< 2, 3 >\}\}.$$

Consider now the modification $(t_{old}, t_{new}) = (< 1, 2, 3 >, < 1, 2, 5 >)$*, defined over* ABC*. The deletion of* t_{old} *from* $\mathbf{r}$ *is not deterministic, since it can be realized either by deleting the tuple* $< 1, 2 >$ *from* r_1 *or the tuple* $< 2, 3 >$ *from* r_2*. However, we have that the modification is indeed deterministic. In fact, let* $\mathbf{r}'$ *and* $\mathbf{r}''$ *be the potential results for the deletion of* t_{old} *from* $\mathbf{r}$*:*

$$\mathbf{r}' = \{r'_1 = \emptyset, r'_2 = \{< 2, 3 >\}\} \text{ and } \mathbf{r}'' = \{r''_1 = \{< 1, 2 >\}, r''_2 = \emptyset\}.$$

Then, it is easy to see that the insertion of t_{new} *in* $\mathbf{r}'$ *is inconsistent, whereas the insertion of* t_{new} *in* $\mathbf{r}''$ *is consistent and deterministic. It follows that the glb of the potential results is indeed a potential result and can be obtained by substituting* $< 2, 3 >$ *by* $< 2, 5 >$ *in* r_2*.*

The above example shows that in general, in order to implement a modification (t_{old}, t_{new}) of a state $\mathbf{r}$, we have first to find all the *maximal potential results* for the deletion of t_{old} from $\mathbf{r}$. We recall that a maximal potential result $\mathbf{r}_M$ for a deletion is a potential result for which there is no other potential result $\mathbf{r}_p$ such that $\mathbf{r}_M \preceq \mathbf{r}_p$ [9]. Then, we have to select, among them, the states for which the insertion of t_{new} is consistent and deterministic. Finally, the results for the insertion of t_{new} to the selected states have to be compared: if there is one that is weaker than each other, the modification is deterministic.

From the discussion above, it turns out that the algorithms derived for implementing insertions and deletions can be also used to implement modification operations.

8. Conclusions. In this paper we have shown that similarly to what has been done with respect to query answering, efficient algorithms to characterize and perform update operations defined over any subset of the universe can be given for the highly significant class of independent schemes. In fact, the various characterizations, which

require time and space polynomial with respect to the size of the database state [9], can be verified for this class of schemes efficiently, as in this case we can derive a restricted number of optimized relational expression that allow us to refer only to the relevant portion of the database.

In particular, with respect to insert operations, we have first shown that the property of consistency can be efficiently tested by considering only to the "extension" of the tuple to be inserted (which is obtained by adding to t further values derived from the original state and the constraints) and the involved FDs. With respect to the property of determinism, we have first provided an alternative method that does not require the construction of a special tableau over the whole database. This method refers to a state obtained from the original database by adding tuples that, for independent schemes, can be efficiently derived. If the insertions is deterministic then this special state corresponds to the result of the insertion. We have then provided for both insertions and deletions practical algorithms implementing the various characterizations. We have finally showed that under some further conditions, update operations can be managed more easily.

Clearly, when non-deterministic updates arise, the system should not simply reject them but rather try to resolve these situations in some way. Indeed, this can be done in several ways since, in general, the problem is that some information for satisfying the request is missing and there are several possible choices for providing this extra information. For instance, potential ambiguities can be solved by means of a dialogue with the users, similarly to the approach described in [8]. Therefore, the algorithms we have presented can be extended in several ways in order to try to resolve non-deterministic update operations.

Recently, new classes of database schemes, generalizing the class of independent schemes, have been introduced [14, 30]. Similarly to the independent ones, these schemes enjoy the property that the consistency of a database state after a simple update to a base relation can be efficiently verified. Thus, it could be interesting to extend the results of this paper to these more general classes of schemes.

Acknowledgment. We would like to thank the anonymous referees for their very helpful comments and suggestions.

REFERENCES

[1] S. Abiteboul, *Updates, a new frontier*, in Second International Conference on Data Base Theory (ICDT'88), Bruges, *Lecture Notes in Computer Science 326*, Springer-Verlag, 1988, pp. 1–18.

[2] A. Aho, C. Beeri, and J. Ullman, *The theory of joins in relational databases*, ACM Trans. on Database Syst., 4 (1979), pp. 297–314.

[3] P. Atzeni and E. Chan, *Efficient query answering in the representative instance approach*, in Fourth ACM SIGACT SIGMOD Symp. on Principles of Database Systems, 1985, pp. 181–188.

[4] ———, *Efficient optimization of simple chase join expressions*, ACM Trans. on Database Syst., 14 (1989), pp. 212–230.

[5] ———, *Efficient and optimal query answering on independent schemes*, Theoretical Computer Science, 77 (1990), pp. 291–308.

[6] P. Atzeni and M. De Bernardis, *A new interpretation for null values in the weak instance model*, Journal of Comp. and System Sc., 41 (1990), pp. 25–43.

[7] P. Atzeni and V. DeAntonellis, *Relational Database Theory: A Comprehensive Introduction*, Benjamin and Cummings Publ. Co., Menlo Park, California, 1993.

[8] P. Atzeni and R. Torlone, *Solving ambiguities in updating deductive databases*, in Mathematical Fundamentals of Data Base Systems (MFDBS'91), Rostock, Germany, *Lecture Notes in Computer Science 495*, Springer-Verlag, 1991, pp. 104–118.

[9] ———, *Updating relational databases through weak instance interfaces*, ACM Trans. on Database Syst., 17 (1992), pp. 718–746.
[10] C. Beeri and P. Bernstein, *Computational problems related to the design of normal form relational schemas*, ACM Trans. on Database Syst., 4 (1979), pp. 30–59.
[11] P. Bernstein, *Synthesizing third normal form relations from functional dependencies*, ACM Trans. on Database Syst., 1 (1976), pp. 277–298.
[12] G. Birkhoff, *Lattice Theory*, Colloquium Publications, Volume XXV, American Mathematical Society, third ed., 1967.
[13] E. Chan, *Optimal computation of total projections with unions of simple chase join expressions*, in ACM SIGMOD International Conf. on Management of Data, 1984, pp. 149–163.
[14] E. Chan and H. Hernandez, *Independence-reducible database schemes*, in Seventh ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems, 1988, pp. 163–173. Also, *J. ACM*, to appear.
[15] E. Chan and A. Mendelzon, *Answering queries on embedded-complete database schemes*, Journal of the ACM, 34 (1987), pp. 349–375.
[16] ———, *Independent and separable database schemes*, SIAM J. Comput., 16 (1987), pp. 841–851.
[17] M. Graham and M. Yannakakis, *Independent database schemas*, Journal of Comp. and System Sc., 28 (1984), pp. 121–141.
[18] P. Honeyman, *Testing satisfaction of functional dependencies*, Journal of the ACM, 29 (1982), pp. 668–677.
[19] M. Ito, M. Iwasaki, and T. Kasami, *Some results on the representative instance in relational databases*, SIAM J. Comput., 14 (1985), pp. 334–354.
[20] D. Maier, *The Theory of Relational Databases*, Computer Science Press, Potomac, Maryland, 1983.
[21] D. Maier, A. Mendelzon, and Y. Sagiv, *Testing implications of data dependencies*, ACM Trans. on Database Syst., 4 (1979), pp. 455–468.
[22] D. Maier, D. Rozenshtein, and D. Warren, *Window functions*, in Advances in Computing Research, Vol.3, P. Kanellakis and F. Preparata, eds., JAI Press, 1986, pp. 213–246.
[23] D. Maier, J. Ullman, and M. Vardi, *On the foundations of the universal relation model*, ACM Trans. on Database Syst., 9 (1984), pp. 283–308.
[24] A. Mendelzon, *Database states and their tableaux*, ACM Trans. on Database Syst., 9 (1984), pp. 264–282.
[25] Y. Sagiv, *Can we use the universal instance assumption without using nulls?*, in ACM SIGMOD International Conf. on Management of Data, 1981, pp. 108–120.
[26] ———, *A characterization of globally consistent databases and their correct access paths*, ACM Trans. on Database Syst., 8 (1983), pp. 266–286.
[27] ———, *On computing restricted projections of the representative instance*, in Fourth ACM SIGACT SIGMOD Symp. on Principles of Database Systems, 1985, pp. 173–180.
[28] ———, *Evaluation of queries in independent database schemes*, Journal of the ACM, 38 (1991), pp. 120–161.
[29] J. Ullman, *Principles of Database Systems*, Computer Science Press, Potomac, Maryland, second ed., 1982.
[30] K. Wang and M. Graham, *Constant-time maintainability: a generalization of independence*, ACM Trans. on Database Syst., 17 (1992), pp. 201–246.
[31] M. Yannakakis, *Querying weak instances*, in Advances in Computing Research, Vol.3, P. Kanellakis and F. Preparata, eds., JAI Press, 1986, pp. 185–211.