A Normal Form for Preventing Redundant Tuples in Relational Databases

Hugh Darwen
University of Warwick, UK

C. J. Date
Independent Consultant

Ronald Fagin*
IBM Research – Almaden

ABSTRACT

We introduce a new normal form, called *essential tuple normal form (ETNF)*, for relations in a relational database where the constraints are given by functional dependencies and join dependencies. ETNF lies strictly between fourth normal form and fifth normal form (5NF, also known as projection-join normal form). We show that ETNF, although strictly weaker than 5NF, is exactly as effective as 5NF in eliminating redundancy of tuples. Our definition of ETNF is semantic, in that it is defined in terms of tuple redundancy. We give a syntactic characterization of ETNF, which says that a relation schema is in ETNF if and only if it is in Boyce-Codd normal form and some component of every explicitly declared join dependency of the schema is a superkey.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*normal forms, schema and subschema*

General Terms

Algorithms, Design, Theory

Keywords

database design, redundancy, essential tuple normal form, fourth normal form, fifth normal form, BCNF, relational database

1. INTRODUCTION

Database design can be characterized as a matter of deciding what the database schema should be – that is, of deciding what attributes (intuitively, "column names") should be in each individual relation schema, and what the constraints should be on the data. It has always been a goal of database design to make the database schema as redundancy-free as possible, and ever since the publication of Codd's first papers on the subject, the discipline of further normalization (normalization for short) has been used to help achieve that goal. Thus, normalization can be described, informally, as a process that leads to a design in which the database schema is redundancy-free. In this paper, we define what it means for a tuple in a relation to be redundancy-free, or "essential", and we define a normal form where every tuple is essential.

In the commercial world, it has long been thought that a database schema must be in fifth normal form (5NF), also known as projection/join normal form (PJ/NF) [9], in order for its tuples to be redundancy-free. We show, however, that a new normal form, one that is strictly weaker than 5NF, is just as effective as 5NF in eliminating redundancy of tuples. In fact, our new normal form – which we call *essential tuple normal form (ETNF)* – is necessary and sufficient for the purpose.[1] In that sense, therefore, we suggest that ETNF is superior to 5NF.

We now illustrate this issue of tuple redundancy through two examples. The first example suffers from tuple redundancy, and the second does not. The first example reminds us of the problem that 5NF is intended to solve, and the second example shows that 5NF is not necessarily the best solution to the problem after all.

EXAMPLE 1.1. Assume that the relation schema $\mathbf{R}$ has exactly three attributes: S ("suppliers"), P ("parts"), and J ("projects"). A tuple (s, p, j) means, intuitively, that supplier s supplies part p to project j. Assume also that the only constraint of $\mathbf{R}$ is the join dependency (JD) $\bowtie\{SP, PJ, JS\}$. Intuitively, this JD says:

- If supplier s supplies part p, and part p is supplied to project j, and project j is supplied by supplier s, then supplier s supplies part p to project j.[2]

The relation schema $\mathbf{R}$ is not in 5NF. Furthermore, it suffers from tuple redundancy, as we now discuss. Let r be a relation that is an instance of $\mathbf{R}$ (so that r has attributes S, P, and J, and r satisfies the JD). Assume that r has tuples (s, p, j'), (s', p, j), and (s, p', j), and possibly other tuples, and that $s \neq s'$, $p \neq p'$, and $j \neq j'$. Because of the JD $\bowtie\{SP, PJ, JS\}$, it follows that r, as an instance of

*Contact author; email fagin@us.ibm.com

[1] In selecting this name for our new normal form, we were influenced by the notion of essentiality suggested by Codd in [5]. Briefly, to say that some data construct is *essential* is to say that its loss would cause a loss of information. Every tuple in every instance of an ETNF relation schema is essential in this sense.

[2] When we say "supplier s supplies part p", we mean "supplier s supplies part p *to some project*" (and similarly for "part p is supplied to project j" and "project j is supplied by supplier s").

$\mathbf{R}$, must contain also the tuple (s,p,j). So in the relation r, the information in the tuple (s,p,j) is represented twice: first, explicitly, by the tuple (s,p,j), and second, implicitly, by the tuples (s,p,j'), (s',p,j), and (s,p',j) and the JD. Intuitively, the tuple (s,p,j) is what we shall call "fully redundant" (the "fully" refers to the fact that the entire tuple is redundant).

Standard principles of normalization suggest that we decompose the relation schema $\mathbf{R}$ into three relation schemas, with attributes, respectively, of $\{S,P\}$, $\{P,J\}$, and $\{J,S\}$, and with no FDs or JDs specified. These three new relation schemas, intuitively, represent projections onto these sets of attributes. Then the tuple redundancy problem disappears. $\square$

Example 1.2. Let $\mathbf{R}'$ be a relation schema that is the same as the relation schema $\mathbf{R}$ of Example 1.1, except that in addition to the JD $\bowtie\{SP,PJ,JS\}$, it also has the functional dependency (FD) $SP \rightarrow J$. Intuitively, this FD says:

- Any given supplier s supplies a given part p to at most one project j.

It is well known that a collection of FDs and JDs can logically imply other FDs and JDs. We can think of the FDs and JDs that are specified for the relation schema as *explicit* dependencies, and the FDs and JDs that are not given explicitly, but are logically implied by the explicit dependencies, as *implicit* dependencies. For example, if $A \rightarrow B$ and $B \rightarrow C$ are explicit FDs, and $A \rightarrow C$ is not an explicit FD, then $A \rightarrow C$ is an implicit FD. An example of an implicit FD that arises through the interaction of an FD and a JD will be given in Example 3.1. However, in the case of the relation schema $\mathbf{R}'$ in the example we are now considering, we shall show later (in the proof of Theorem 6.1) a straightforward verification, using the chase process[3] that no new nontrivial FDs or JDs are implied by our two dependencies. (A dependency is *trivial* if it "always holds" for relations with the appropriate attributes.)

What normal form is the relation schema $\mathbf{R}'$ in? Since there are no nontrivial multivalued dependencies (MVDs[4]), it follows easily from the characterization of 4NF in [9] that $\mathbf{R}'$ is in 4NF. However, $\mathbf{R}'$ is not in 5NF, because the JD is not logically implied by the keys (in particular by the only nontrivial FD $SP \rightarrow J$).

Let r be a relation that is an instance of $\mathbf{R}'$ (so that r has attributes S, P, and J, and satisfies the two dependencies). Assume, as we did in Example 1.1, that r has tuples (s,p,j'), (s',p,j), and (s,p',j), and possibly other tuples. Unlike Example 1.1, we no longer assume that that $j \neq j'$. Because of the JD, it follows that r, as an instance of $\mathbf{R}'$, must contain the tuple (s,p,j). But because r contains the tuples (s,p,j') and (s,p,j), and because of the FD $SP \rightarrow J$, it follows that $j = j'$. So we no longer have the tuple redundancy that we exhibited in Example 1.1, because the information in the tuple (s,p,j) is represented only once. Thus, unlike the situation in Example 1.1, this information is not "also represented implicitly" by the tuples (s,p,j'), (s',p,j), and (s,p',j) and the JD, since these three tuples have (s,p,j) as one of their members (because $(s,p,j') = (s,p,j)$).

Here is another way to view this example. We typically expect a nontrivial JD to "force new tuples to be present". But this does not happen in the relation schema $\mathbf{R}'$. Thus, assume that we start with a relation r' that contains the tuples (s,p,j'), (s',p,j), and (s,p',j), and assume that we wish to extend r' to a relation r that satisfies the constraints ("extend" means that $r' \subseteq r$). Then we do not need to add the tuple (s,p,j) to r', since, as we showed, necessarily $j = j'$, and so the tuple (s,p,j) is already present in r'. $\square$

We argue, therefore, that there is nothing wrong with the relation schema $\mathbf{R}'$ of Example 1.2 as far as tuple redundancy issues are concerned, even though it is not in 5NF. We now define two notions of tuple redundancy.

Tuple redundancy Our first notion of tuple redundancy is with respect to FDs. It is based on the classical Boyce-Codd normal form (BCNF) [4]. The definition we now give of BCNF is equivalent to the definition given in [4].

Definition 1.3. *A relation schema* $\mathbf{R}$ *is in* Boyce-Codd normal form (BCNF) *if every explicit or implicit FD of* $\mathbf{R}$ *is logically implied by the keys of* $\mathbf{R}$. $\square$

It is well known (see, for example, [9]) that $\mathbf{R}$ is in BCNF if and only if for every explicit or implicit nontrivial FD $X \rightarrow Y$ of $\mathbf{R}$, necessarily X is a superkey (a *superkey* is a subset of the attributes that is also a superset of a key).

The fact that both explicit and implicit FDs of $\mathbf{R}$ need to be considered in Definition 1.3 will be demonstrated in Example 3.1.

A *tuple (over a finite set $\mathcal{A}$ of attributes)* is a function with domain $\mathcal{A}$ and range some set of values. Thus, if A is an attribute, then $t(A)$ is the value for attribute A of tuple t. If $X \subseteq \mathcal{A}$, and t is a tuple, then $t[X]$ is the restriction of t to the set X. Thus, $t[X]$ is also a tuple, which is sometimes referred to as the *projection* of the tuple t on X. We now define the notion of a tuple being *partly redundant*. This notion is based on the intuition that an FD $X \rightarrow A$ is thought of as a function that associates with every X-value some unique A-value. Intuitively, if t is a tuple, then $t[X]$ uniquely determines the value $t(A)$.

Definition 1.4. *Let* $\mathbf{R}$ *be a relation schema, let* r *be a relation that is an instance of* $\mathbf{R}$*, and let* t *be a tuple of* r*. The tuple* t *is* partly redundant in r (with respect to $\mathbf{R}$) *if (a) there is a tuple* t' *of* r *with* $t \neq t'$*, (b) there is a nontrivial FD* $X \rightarrow A$ *of* $\mathbf{R}$ *(explicit or implicit), and (c)* $t[X] = t'[X]$. $\square$

The reason we say that t is partly redundant is as follows. Since $t[X] = t'[X]$, and since the relation r satisfies the FD $X \rightarrow A$, it follows that $t(A) = t'(A)$. Thus, intuitively, the information as to what A-value is associated with some X-value is given by both t and t'. We have the following proposition.

Proposition 1.5. *Let* $\mathbf{R}$ *be an arbitrary relation schema. Then* $\mathbf{R}$ *is in BCNF if and only if no instance has a partly redundant tuple.*

Proof. Assume first that $\mathbf{R}$ is in BCNF, and that some instance r of $\mathbf{R}$ has a partly redundant tuple; we shall derive a contradiction. Let t be a partly redundant tuple of r. Then by definition, there is a tuple t' of r with $t \neq t'$ and there

[3]To decide when a given dependency is a logical consequence of a set of dependencies, we shall often make use of the *chase* process [1, 13]. Since the chase is a standard tool, we do not describe here the details of the chase.

[4]MVDs, along with other notions mentioned in the introduction, will be formally defined in Section 2.

is a nontrivial explicit or implicit FD $X \rightarrow A$ of $\mathbf{R}$ such that $t[X] = t'[X]$. Since $\mathbf{R}$ is in BCNF, it follows from the comment after Definition 1.3 that necessarily X is a superkey. Hence, since $t[X] = t'[X]$, it follows that $t = t'$. This is our desired contradiction.

Assume now that $\mathbf{R}$ is not in BCNF; we shall show that $\mathbf{R}$ has an instance with a partly redundant tuple. Since $\mathbf{R}$ is not in BCNF, it follows from the comment after Definition 1.3 that there is a nontrivial explicit or implicit FD $X \rightarrow Y$ of $\mathbf{R}$ where X is not a superkey. Since X is not a superkey, there is an attribute A such that $X \rightarrow A$ is not an FD (explicit or implicit) of $\mathbf{R}$.

Let s and s' be tuples, whose attributes are the attributes of $\mathbf{R}$, with $s[X] = s'[X]$, and where $s(B) \neq s'(B)$ for every attribute B not in X. Let us now apply the chase process to the relation containing just these two tuples s and s', where we treat the entries of the tuples as variables that can be equated by the chase process. Let r be the relation that is the result of the chase, let t be the tuple of r that s is eventually converted into at the end of the chase, and let t' be the tuple of r that s' is eventually converted into at the end of the chase. Since $X \rightarrow A$ is not an FD (explicit or implicit) of $\mathbf{R}$, it follows by the standard theory of the chase [1, 13] that $t(A) \neq t'(A)$. Hence, $t \neq t'$. Further, by the standard theory of the chase [1, 13], we know that r satisfies the dependencies of $\mathbf{R}$. Since $s[X] = s'[X]$, and since s (respectively, s') is eventually converted into t (respectively, t') as a result of the chase, it follows that $t[X] = t'[X]$. Since $t[X] = t'[X]$ and $t \neq t'$, where t and t' are tuples of the relation r with r satisfying the dependencies of $\mathbf{R}$, it follows that t is a partly redundant tuple of the instance r of $\mathbf{R}$. So an instance of $\mathbf{R}$ has a partly redundant tuple, which was to be shown. $\square$

Let $\mathbf{R}$ be a relation schema, let S be a set of tuples, and let t be a tuple. We say that S *logically implies* t *(with respect to* $\mathbf{R}$*)* if every instance of $\mathbf{R}$ that contains all of the tuples in S also necessarily contains the tuple t. We now define a second notion of tuple redundancy.

DEFINITION 1.6. *Let* $\mathbf{R}$ *be a relation schema, let* r *be a relation that is an instance of* $\mathbf{R}$*, and let* t *be a tuple of* r*. The tuple* t *is* fully redundant in r (with respect to $\mathbf{R}$) *if there is a set* S *of tuples in* r *where* $t \notin S$ *such that* S *logically implies* t *with respect to* $\mathbf{R}$. $\square$

Intuitively, t is fully redundant if its presence is already logically implied anyway by the presence of other tuples. Similarly to the discussion in Example 1.1, in the relation r the information in the tuple t is represented twice: first, explicitly, by the tuple t, and second, implicitly, by the tuples in S and the dependencies. It is easy to see that if the relation schema is specified only by FDs, then there can be no fully redundant tuple.

Note that "partly redundant" and "fully redundant" are orthogonal notions: it is possible for a tuple to be fully redundant without being partly redundant, and vice-versa.

We now give Fagin's [10] definition of a *deletion anomaly.*[5] We then show that a relation schema that is specified only by FDs and JDs has a deletion anomaly if and only if it has an instance with a fully redundant tuple.

[5]Technically, we have to replace the word "dependencies" by "constraints" to obtain Fagin's definition.

DEFINITION 1.7. *A relation schema* $\mathbf{R}$ *is said to have a* deletion anomaly *if there are relations* r_1 *and* r_2*, each with the attributes of* $\mathbf{R}$ *as their attributes, such that*

1. r_2 *consists of the tuples of* r_1 *along with exactly one other tuple* t*,*
2. r_1 *does not satisfy all of the dependencies of* $\mathbf{R}$*, and*
3. r_2 *satisfies all of the dependencies of* $\mathbf{R}$*.*

$\square$

Thus, a deletion anomaly arises when the seemingly harmless operation of deleting just the tuple t from the instance r_2 of the schema $\mathbf{R}$ leads to r_1, which is not an instance of $\mathbf{R}$, since it does not satisfy all of the dependencies of $\mathbf{R}$.

LEMMA 1.8. *An equivalent definition to Definition 1.7 can be obtained by replacing part (1) of Definition 1.7 by "*$r_1 \subsetneq r_2$*".*

PROOF. Since r_1 is a proper subset of r_2, there are relations $s_1, \ldots, s_k$ where $s_1 = r_1$ and $s_k = r_2$, and where s_{i+1} is obtained from s_i by adding exactly one tuple, for $1 \leq i \leq k-1$. Since s_1 does not satisfy the dependencies of $\mathbf{R}$ while s_k does, there is j with $1 \leq j \leq k-1$ such that s_j does not satisfy all of the dependencies of $\mathbf{R}$ while s_{j+1} satisfies all of the dependencies of $\mathbf{R}$. Let $r'_1 = s_j$, let $r'_2 = s_{j+1}$, and let t be the tuple in s_{j+1} but not not s_j. Then the conditions of Definition 1.7 hold when r'_1 plays the role of r_1, and r'_2 plays the role of r_2. $\square$

PROPOSITION 1.9. *Let* $\mathbf{R}$ *be a relation schema that is specified only by FDs and JDs. Then* $\mathbf{R}$ *has a deletion anomaly if and only if* $\mathbf{R}$ *has an instance with a fully redundant tuple.*

PROOF. Assume first that $\mathbf{R}$ has a deletion anomaly; we shall show that $\mathbf{R}$ has an instance with a fully redundant tuple. Let r_1, r_2, and t be as in Definition 1.7. Let r be r_2, and let S be r_1. Since $r_1 \subseteq r_2$ and r_2 satisfies the FDs of $\mathbf{R}$, also r_1 satisfies the FDs of $\mathbf{R}$. Therefore, it is not hard to see that r_2 is the result of applying a JD of $\mathbf{R}$ to r_1. Hence, S logically implies t with respect to $\mathbf{R}$. Since $t \notin S$, it follows by definition that t is fully redundant in r with respect to $\mathbf{R}$.

Now assume that $\mathbf{R}$ has an instance with a fully redundant tuple; we shall show that $\mathbf{R}$ has a deletion anomaly. Let t, r, and S be as in Definition 1.6. Since S logically implies t, and $t \notin S$, we know that S does not satisfy the dependencies of $\mathbf{R}$. Let r_1 be S, and let r_2 be r. Then by Lemma 1.8, it follows that $\mathbf{R}$ has a deletion anomaly, as desired. $\square$

Fagin [10] also defines the notion of an insertion anomaly, which we now discuss. He defines *key dependencies*, which are FDs where the right-hand side is the set of all attributes, and *domain dependencies*, which are sentences of the form $\mathrm{IN}(A, S)$, where A is an attribute and S is a set of values. The domain dependency $\mathrm{IN}(A, S)$ holds for a relation r if the value of every tuple on the attribute A is a member of the set S. (Other than in this introductory section, we shall not consider domain dependencies in this paper.)

DEFINITION 1.10. *A relation schema* $\mathbf{R}$ *is said to have an* insertion anomaly *if there are relations* r_1 *and* r_2*, each with the attributes of* $\mathbf{R}$ *as their attributes, such that*

1. r_2 *consists of the tuples of* r_1 *along with exactly one other tuple* t*,*

2. *r_1 satisfies all of the dependencies of* **R**,

3. *r_2 does not satisfy all of the dependencies of* **R**, *and*

4. *r_2 satisfies the key dependencies and domain dependencies of* **R**.

□

Note that parts (2) and (3) are the reverse of parts (2) and (3) of Definition 1.7.

An insertion anomaly arises when the operation of adding the tuple t to the instance r_1 of the schema **R** leads to r_2, which is not an instance of **R**, even though the tuple t does not agree with any tuple of r_1 on a key, and the tuple t satisfies the domain dependencies.

Fagin [10] defines an "ultimate" normal form, which he calls *domain-key normal form (DK/NF)*, which holds if every constraint is logically implied by the key dependencies and domain dependencies. He shows that a relation schema is in DK/NF if and only if it has no insertion anomalies and no deletion anomalies. The next proposition says that if we restrict our attention to relation schemas where the only constraints are FDs and JDs (in particular, where domain dependencies are not allowed) then DK/NF is equivalent to 5NF, and both are equivalent to there being no insertion anomaly.

PROPOSITION 1.11. *Let* **R** *be a relation schema that is specified only by FDs and JDs. The following are equivalent.*

1. **R** *is in DK/NF.*

2. **R** *is in 5NF.*

3. **R** *has no insertion anomalies.*

PROOF. (1) and (2) are equivalent, because both say that all of the constraints (which in this case are only FDs and JDs) are implied by the keys. It is shown in [10] that (1) implies (3). It is also shown in [10] that a relation schema with arbitrary constraints (not necessarily just FDs or JDs) is in DK/NF if and only if (1) it has no insertion anomalies, and (2) the empty relation satisfies all of the constraints. In our case of interest, where the only constraints are FDs and JDs, it is automatically true that the empty relation satisfies all of the constraints. It therefore follows easily that (3) implies (1). □

Let us say that a tuple is *essential* if it is neither partly nor fully redundant. We now define our new normal form.

DEFINITION 1.12. *A relation schema* **R** *is in* essential tuple normal form (ETNF) *if every tuple in every instance of* **R** *is essential.* □

As we shall show in the proof of Theorem 6.1, the relation schema $\mathbf{R}'$ of Example 1.2 is in ETNF but not 5NF.

Define the *components* of a JD $\bowtie\{C_1, \ldots, C_k\}$ to be the sets $C_1, \ldots, C_k$. It can be shown by an argument similar to that in Example 1.2 that if a relation schema **R** is in BCNF and some component of every explicit JD of **R** is a superkey (as with the JD in Example 1.2, where the component SP is a key), then **R** is in ETNF (we shall give a proof in Section 4.2). It is interesting (and quite nontrivial to prove) that the converse also holds. Thus, we shall prove the following theorem, which is our main technical result.

THEOREM 1.13. *Let* **R** *be a relation schema specified only by FDs and JDs. Then* **R** *is in ETNF if and only if it is in BCNF and some component of every explicit JD of* **R** *is a superkey.*

Of course, we could replace "explicit JD" by "explicit or implicit JD" in Theorem 1.13, since every implicit JD can be made explicit if desired. We state Theorem 1.13 as we do to make the "if" direction stronger. Thus, the "if" direction of Theorem 1.13 tells us that to decide if a BCNF relation schema is in ETNF, we need to check only the explicit JDs to see if some component is a superkey; it then can be shown to follow automatically that for the implicit JDs also, some component is a superkey. We note that we could make the "if" direction even stronger by replacing "every explicit JD" by "every explicit, irreducible JD".[6] But we do not do this, since the statement of the theorem seems a little cleaner without the word "irreducible" in it, and since it is clear that some component of every explicit, irreducible JD of **R** is a superkey if and only if some component of every explicit JD of **R** is a superkey.

Theorem 1.13 gives a syntactic characterization of ETNF. It is quite interesting that this syntactic characterization of ETNF is closely related to syntactical characterizations of two other normal forms between 4NF and 5NF. We discuss one of these now, and the other in Section 1.1.

The first of these other normal forms arose because of a common misperception in the literature about 5NF, which we now discuss, after giving some more definitions.

DEFINITION 1.14. *Let* **R** *be a relation schema. Then a JD* $\bowtie\{C_1, \ldots, C_k\}$ *of* **R** *is* irreducible (with respect to **R**) *if there is no proper subset* $\{C_{i_1}, \ldots, C_{i_s}\}$ *of* $\{C_1, \ldots, C_k\}$ *such that* $\bowtie\{C_{i_1}, \ldots, C_{i_s}\}$ *is a JD (explicit or implicit) of* **R**. □

Note that if the JDs $\bowtie\{C_1, \ldots, C_k\}$ and $\bowtie\{C_{i_1}, \ldots, C_{i_s}\}$ are as in Definition 1.14, then $\bowtie\{C_{i_1}, \ldots, C_{i_s}\}$ logically implies $\bowtie\{C_1, \ldots, C_k\}$. So the irreducible JDs logically imply all of the JDs of **R**. Note also that if one component C_i of a JD J is a proper subset of another component of J, then J is not irreducible, since the JD J' that is the result of removing the component C_i from J is an explicit or implicit JD of **R** (because J and J' are logically equivalent).

DEFINITION 1.15. [14] *A relation schema* **R** *is in* superkey normal form (SKNF) *if every component of every irreducible JD of* **R** *(explicit or implicit) is a superkey.* □

This definition of superkey normal form is due to Normann [14].[7] As Normann notes, this definition appears in Maier's textbook [12] as a preliminary version of 5NF (before Maier gives the "real" definition, due to Fagin [10]). A primary reason that SKNF is of interest is that, as shown by Normann [14], many textbooks (including the textbook [6] by our second author) incorrectly give the definition of SKNF as a definition of 5NF. An equivalent definition to SKNF was given by Vincent [18], who calls it 5NFR. Normann and Vincent both show that 5NF $\Rightarrow$ SKNF $\Rightarrow$ 4NF, and that neither reverse implication holds. Our interest in SKNF is for two reasons.

[6]The definition of an *irreducible JD* is given in Definition 1.14.

[7]Normann abbreviates superkey normal form as SNF rather than SKNF, but to us the abbreviation SNF looks too physically similar to the abbreviation 5NF.

- It is an intermediate normal form (a normal form between 4NF and 5NF), and we wish to know the relationship between the various intermediate normal forms, including ours.

- The syntactic condition that defines SKNF in Definition 1.15 turns out to be very similar to syntactic characterizations of the other intermediate normal forms.

These issues are discussed in Section 1.2.

Date and Fagin [7] give simple conditions, that we now describe, that guarantee that a relation schema is in a higher normal form, namely 4NF or 5NF. It is shown in [7] that if a relation schema $\mathbf{R}$ is in third normal form (3NF) and every key has only one attribute, then $\mathbf{R}$ is in 5NF. It is also shown in [7] that if a relation schema $\mathbf{R}$ is in BCNF and some key has only one attribute, then $\mathbf{R}$ is in 4NF. In Section 5, we show that the conclusion of this latter result can be strengthened to say that $\mathbf{R}$ is in ETNF. Thus, we show that if a relation schema $\mathbf{R}$ is in BCNF and some key has only one attribute, then $\mathbf{R}$ is in ETNF. This is nice, because it gives a simple, natural condition involving FDs only (that is, not involving JDs) that guarantees ETNF.

1.1 Related work

Vincent [17] defines the notion of a data value (an entry of a tuple) being redundant. We now give a definition equivalent to his. Since it is somewhat ambiguous to refer to a value in a tuple (e.g., that same value could appear several times within the tuple), we reference a value $t(A)$ of a tuple t as the pair (t, A).

Definition 1.16. [17] *Let* $\mathbf{R}$ *be a relation schema, let* r *be a relation that is an instance of* $\mathbf{R}$*, let* t *be a tuple of* r*, and let* A *be an attribute of* $\mathbf{R}$*, The pair* (t, A) *is* redundant in r (with respect to $\mathbf{R}$) *if whenever* t' *is a tuple with* $t'(A) \neq t(A)$ *and* $t'(B) = t(B)$ *for each attribute* B *other than* A*, and* r' *is the result of replacing the tuple* t *in* r *by* t'*, then* r' *is not an instance of* $\mathbf{R}$. $\Box$

We now show that our notions of a tuple being partly redundant or fully redundant give special cases of the notion of redundancy in Definition 1.16. In the case of "partly redundant", if r, t, and A are as in Definition 1.4, then it is straightforward to see that (t, A) is redundant in r. In the case of "fully redundant", if r and t are as in Definition 1.6, then it is straightforward to see that (t, A) is redundant in r for each attribute A.

Intuitively, Definition 1.16 says that the value of tuple t for attribute A is redundant if it is uniquely determined by the rest of the relation. Vincent defines a new normal form based on this notion of redundancy; we now give a definition equivalent to his.

Definition 1.17. [17] *A relation schema* $\mathbf{R}$ *is in* redundancy-free normal form (RFNF) *if it is not the case that there is an instance* r *of* $\mathbf{R}$*, a tuple* t *of* r*, and an attribute* A *such that* (t, A) *is redundant in* r *with respect to* $\mathbf{R}$. $\Box$

Vincent [17] shows that if $\mathbf{R}$ is a relation schema specified only by FDs and MVDs, then $\mathbf{R}$ is in RFNF if and only if it is in 4NF. The interesting case, which we now discuss, is when $\mathbf{R}$ is specified only by FDs and JDs.

Assume that $\mathbf{R}$ is specified only by FDs and JDs. Vincent then gives a syntactic characterization of RFNF. He refers to this syntactic characterization as *key-complete normal form (KCNF)*. We now give a definition equivalent to his.

Definition 1.18. [17] *A relation schema* $\mathbf{R}$ *that is specified only by FDs and JDs is in* key-complete normal form (KCNF) *if it is in BCNF, and if for every JD* J *of* $\mathbf{R}$ *(explicit or implicit), the union of the components of* J *that are superkeys contains every attribute of* $\mathbf{R}$. $\Box$

Vincent proves the following theorem, which shows that KCNF is a syntactic characterization of RFNF.

Theorem 1.19. [17] *Let* $\mathbf{R}$ *be a relation schema specified only by FDs and JDs. Then* $\mathbf{R}$ *is in RFNF if and only if it is in KCNF.*

Arenas and Libkin [2] define a numerical measure (based on entropy) of the redundancy of a relation schema. They refer to a relation schema with zero redundancy as being *well designed*. Vincent et al. [19] prove that if $\mathbf{R}$ is a relation schema specified only by FDs and JDs, then $\mathbf{R}$ is in RFNF if and only if it is well designed.

As we shall show in the proof of Theorem 6.1, the relation schema $\mathbf{R}'$ of Example 1.2 is in ETNF but not RFNF. It is instructive to see an example of a redundancy that keeps $\mathbf{R}'$ from being in RFNF. Let r be a relation that consists of the tuples (s, p, j), (s', p, j), and (s, p', j), with $s \neq s'$ and $p \neq p'$. Then r is an instance of $\mathbf{R}'$. Let t be the tuple (s, p, j). We now show that (t, J) (which corresponds to the entry j of tuple t) is redundant in r. This is because if we were to try to replace j with j' in the tuple (s, p, j), we would have the relation r' with tuples (s, p, j'), (s', p, j), and (s, p', j). As discussed in Example 1.2, the explicit JD and explicit FD of $\mathbf{R}'$ force $j' = j$, which proves redundancy. This redundancy implies that $\mathbf{R}'$ is not in RFNF.

In other related work, Thalheim [16] defines what he calls the *deductive normal form* of a relation r (with respect to a relation schema $\mathbf{R}$). It is not really a normal form in our sense, but is simply a minimal subset of the tuples of r that generates r via the chase. It is not hard to see that a relation schema has no instance with a fully redundant tuple if and only if the deductive normal form of every instance r is r itself. However, Thalheim does not define or study this case where the deductive normal form of every instance r is r itself.

1.2 Relationships between normal forms

We shall show that 5NF $\Rightarrow$ SKNF $\Rightarrow$ RFNF $\Rightarrow$ ETNF $\Rightarrow$ 4NF, and that none of the reverse implications hold.

Normann [14] shows that 5NF $\Rightarrow$ SKNF $\Rightarrow$ 4NF, and that neither reverse implication holds. Vincent [17] shows that 5NF $\Rightarrow$ SKNF $\Rightarrow$ RFNF $\Rightarrow$ 4NF, and that none of the reverse implications hold.

It is quite interesting to compare the syntactic conditions, which involve the interrelationship of superkeys with components of JDs of the schema, that characterize the intermediate normal forms SKNF, RFNF, and ETNF. Let $\mathbf{R}$ be a relation schema specified only by FDs and JDs. By Definition 1.15, we have that $\mathbf{R}$ is in SKNF if and only if for every irreducible JD J of $\mathbf{R}$, every component of J is a superkey. By Theorem 1.19, we have that $\mathbf{R}$ is in RFNF if and only if it is in BCNF and for every JD J of $\mathbf{R}$, the union of the components of J that are superkeys includes every attribute of $\mathbf{R}$. By Theorem 1.13, we have that $\mathbf{R}$ is in ETNF if and only if it is in BCNF and for every JD J of $\mathbf{R}$, some component of J is a superkey.

1.3 Summary of contributions

We introduce a new normal form, called *essential tuple normal form (ETNF)*, which lies strictly between 4NF and 5NF, and which is exactly what is needed to eliminate redundancy of tuples. Our main technical result is a syntactic characterization of ETNF: a relation schema is in ETNF if and only if it is in BCNF and some component of every explicitly declared join dependency of the schema is a superkey. We show the relationship between ETNF and other normal forms in the literature that are strictly between 4NF and 5NF, namely SKNF and RFNF. Specifically, we show that 5NF $\Rightarrow$ SKNF $\Rightarrow$ RFNF $\Rightarrow$ ETNF $\Rightarrow$ 4NF, and that none of the reverse implications hold. Interestingly, all three of these intermediate normal forms can be characterized by saying that the schema must be in BCNF and that there is some interrelationship of superkeys with components of JDs of the schema We also give a simple sufficient condition for ETNF: a relation schema is in ETNF if it is in BCNF and some key has only one attribute.

2. DEFINITIONS AND BACKGROUND

In this section, we give basic definitions and background needed for this paper.

We shall consider three types of *dependencies*, or *sentences*, in this paper, namely functional dependencies, multivalued dependencies, and join dependencies, each of which we shall define shortly. A set Σ of sentences *logically implies* a sentence τ (or τ is a *logical consequence of* Σ) if every relation that satisfies every sentence of Σ also satisfies τ. If Σ is a singleton set $\{\sigma\}$, then we say that σ logically implies τ. We say that two sentences are *logically equivalent* if each logically implies the other.

We formally defined *tuple* in the introduction. A *relation (with attributes $\mathcal{A}$)* is a finite set of tuples over $\mathcal{A}$. If $X \subseteq \mathcal{A}$, then $r[X]$ is the set of all tuples $t[X]$ where t is a tuple of r. The relation $r[X]$ is called the *projection of r onto X*.

The *active domain* of a relation r is the set of values $t(A)$ over all tuples t of r and all attributes A. Thus, the active domain of relation r is the set of members of the domain that actually appear in r. If an ordering on the set $\mathcal{A}$ of attributes is understood, and the number of attributes is n, then we may write a tuple as $(a_1, \ldots, a_n)$, where a_i is the value of the tuple on the ith attribute, for $1 \leq i \leq n$.

If A and B are attributes, we may write AB for the set $\{A, B\}$. If X and Y are sets of attributes, we may write XY for the set $X \cup Y$.

A *functional dependency or FD (over the set $\mathcal{A}$ of attributes)* is a sentence of the form $X \rightarrow Y$, where X and Y are subsets of the set $\mathcal{A}$ of attributes. If X is a singleton set $\{A\}$, we may write A for $\{A\}$ in the FD, and similarly for Y. We say that a relation r *satisfies* the FD $X \rightarrow Y$ (or the FD $X \rightarrow Y$ *holds for* r) if for each pair t_1, t_2 of tuples in r such that $t_1[X] = t_2[X]$, we have $t_1[Y] = t_2[Y]$. A *trivial* FD (over $\mathcal{A}$) is one that holds for every relation with attributes $\mathcal{A}$. It is easy to see that an FD $X \rightarrow Y$ is trivial if and only if $Y \subseteq X$. Functional dependencies were first considered by Codd [3], but not using this formalism.

A *multivalued dependency or MVD (over the set $\mathcal{A}$ of attributes)* [8] is a sentence of the form $X \rightarrow\rightarrow Y$, where X and Y are subsets of the set $\mathcal{A}$ of attributes. Let Z be the set difference $\mathcal{A} \setminus (X \cup Y)$. We may then also write the MVD $X \rightarrow\rightarrow Y$ as $X \rightarrow\rightarrow Y|Z$. We say that a relation r *satisfies* the MVD $X \rightarrow\rightarrow Y|Z$ (or the MVD $X \rightarrow\rightarrow Y|Z$ *holds for* r) if whenever t_1 and t_2 are tuples in r such that $t_1[X] = t_2[X]$, then there is a tuple t in r such that $t[X] = t_1[X]$, $t[Y] = t_1[Y]$, and $t[Z] = t_2[Z]$. A *trivial* MVD (over $\mathcal{A}$) is one that holds for every relation with attributes $\mathcal{A}$. The MVD $X \rightarrow\rightarrow Y$ is trivial if and only if either $Y \subseteq X$ or $X \cup Y = \mathcal{A}$ [8]. Without loss of generality, we can restrict our attention to MVDs $X \rightarrow\rightarrow Y$ where X and Y are disjoint (and so X, Y, and Z are pairwise disjoint), because of the simple result [8] that the MVD $X \rightarrow\rightarrow Y$ is logically equivalent to the MVD $X \rightarrow\rightarrow Y'$, where Y' is the set difference $Y \setminus X$.

A *join dependency or JD (over the set $\mathcal{A}$ of attributes)* [15] is a sentence of the form $\bowtie\{C_1, \ldots, C_k\}$, where $C_1 \cup \ldots \cup C_k = \mathcal{A}$. We call each C_i a *component* of the JD $\bowtie\{C_1, \ldots, C_k\}$. We say that a relation r *satisfies* the JD $\bowtie\{C_1, \ldots, C_k\}$ (or the JD $\bowtie\{C_1, \ldots, C_k\}$ *holds for* r) if whenever $t_1, \ldots, t_k$ are tuples in r, and there is a tuple t such that $t[C_i] = t_i[C_i]$, for $1 \leq i \leq k$, then t is a tuple in r. The name "join dependency" refers to the fact that a relation r satisfies the JD $\bowtie\{C_1, \ldots, C_k\}$ if and only if r is the join of its projections $r[C_1], \ldots, r[C_k]$. This is sometimes expressed by saying that r *decomposes losslessly* onto these projections. A *trivial* JD (over $\mathcal{A}$) is one that holds for every relation with attributes $\mathcal{A}$. It is easy to see that a JD is trivial if and only if some component is the set of all attributes.

A *dependency* is an FD, MVD, or JD. An MVD can be viewed as a JD, in that the MVD $X \rightarrow\rightarrow Y|Z$ is logically equivalent to the JD $\bowtie\{XY, XZ\}$.

A *relation schema* $\mathbf{R}$ is a pair consisting of a finite set of attributes and a set of constraints. It is often convenient to say that $\mathbf{R}$ is *specified by* these constraints. In the case that we are focusing on, where the constraints are FDs and JDs, we shall write a relation schema as a triple $(\mathcal{A}, \mathcal{F}, \mathcal{J})$, where $\mathcal{A}$ is a finite set of attributes, $\mathcal{F}$ is a set of FDs over $\mathcal{A}$, and $\mathcal{J}$ is a set of JDs over $\mathcal{A}$. We then say that $\mathbf{R}$ is specified by $\mathcal{F}$ and $\mathcal{J}$. Each member of $\mathcal{F}$ is an *explicit FD* of $\mathbf{R}$, and each member of $\mathcal{J}$ is an *explicit JD* of $\mathbf{R}$. *Implicit* dependencies are those not in $\mathcal{F}$ or $\mathcal{J}$ but which are logically implied by $\mathcal{F} \cup \mathcal{J}$. A relation r is an *instance* of the relation schema $\mathbf{R}$ if (a) $\mathcal{A}$ is the set of attributes of r and (b) r satisfies the dependencies in $\mathcal{F}$ and $\mathcal{J}$.

A *key* of the relation schema $\mathbf{R} = (\mathcal{A}, \mathcal{F}, \mathcal{J})$ is a subset of $\mathcal{A}$ such that (a) the FD $K \rightarrow \mathcal{A}$ is an explicit or implicit FD of $\mathbf{R}$ and (b) there is no proper subset K' of K such that the FD $K' \rightarrow \mathcal{A}$ is an explicit or implicit FD of $\mathbf{R}$. A *superkey* of $\mathbf{R}$ is a subset of $\mathcal{A}$ that is also a superset of a key of $\mathbf{R}$. When we say that a dependency σ is *logically implied by the keys*, we mean that σ is a logical consequence of the set of FDs of the form $K \rightarrow \mathcal{A}$, where K is a key.

We now define the "classic" normal forms that we shall make use of in this paper. Since we mention third normal form (3NF) [3] only in passing, we shall not give the definition here of 3NF.

The definition of BCNF appears in the introduction (Definition 1.3). We note for later use the following simple proposition from [9]. It helps clarify the definition of BCNF.

PROPOSITION 2.1. [9] *A nontrivial FD $X \rightarrow Y$ is logically implied by the keys of a relation schema $\mathbf{R}$ if and only if X is a superkey of $\mathbf{R}$.*

We now define 4NF and 5NF.

Definition 2.2. [8] *A relation schema* $\mathbf{R}$ *is in* fourth normal form (4NF) *if every MVD of* $\mathbf{R}$ *is logically implied by the keys of* $\mathbf{R}$. *(By an* MVD of $\mathbf{R}$, *we mean those explicit or implicit JDs of* $\mathbf{R}$ *that are logically equivalent to an MVD.)* $\square$

Definition 2.3. [9] *A relation schema* $\mathbf{R}$ *is in* fifth normal form *(*5NF*; also known as* projection-join normal form*, or* PJ/NF*) if every explicit or implicit JD of* $\mathbf{R}$ *is logically implied by the keys of* $\mathbf{R}$. $\square$

We shall make use of the *Membership Algorithm* of [9], which tells when a JD $\bowtie\{C_1, \ldots, C_k\}$ is logically implied by the keys.

Membership Algorithm [9] (Input is a finite set $\mathcal{A}$ of attributes, a JD $\bowtie\{C_1, \ldots, C_k\}$, and a set $\{K_1, \ldots, K_r\}$ of keys.)

1. Initialize the set S to be $\{C_1, \ldots, C_k\}$.
2. Apply the following rule until it can no longer be applied: if $K_i \subseteq Y \cap Z$ for some i (with $1 \leq i \leq r$) and some members Y, Z of S, then replace Y and Z in S by their union, that is, remove the sets Y and Z from S and add to S the single member $Y \cup Z$. (Note that the number of members of S then decreases by one.)
3. On termination, if $\mathcal{A}$ is a member of S, then accept, and otherwise reject.

Proposition 2.4. [9] *Let* $\mathbf{R}$ *be a relation schema with attributes* $\mathcal{A}$ *and with* $\{K_1, \ldots, K_r\}$ *as its set of keys. The JD* $\bowtie\{C_1, \ldots, C_k\}$ *over* $\mathcal{A}$ *is logically implied by the keys of* $\mathbf{R}$ *if and only if the Membership Algorithm accepts with inputs* $\mathcal{A}$, $\bowtie\{C_1, \ldots, C_k\}$, *and* $\{K_1, \ldots, K_r\}$.

Assume that relations r and r' have the same attributes. A *homomorphism* from r to r' is a function h from the active domain of r to the active domain of r' such that whenever $(b_1, \ldots, b_n)$ is a tuple of r, then $(h(b_1), \ldots, h(b_n))$ is a tuple of r'. For convenience, if $b = (b_1, \ldots, b_n)$, then we denote $(h(b_1), \ldots, h(b_n))$ by $h(b)$.

3. IMPLICIT DEPENDENCIES

We now discuss an example that shows the importance of considering implicit dependencies.

Example 3.1. Let $\mathbf{R}$ be the relation schema with

$$\mathcal{A} = \{A, B, C\},$$

$$\mathcal{F} = \{AB \rightarrow C\},$$

$$\mathcal{J} = \{\bowtie\{AB, AC\}\}.$$

We shall show that $A \rightarrow C$ is an implicit FD, and A is not a key. Hence, this relation schema is not in BCNF. This example shows that to determine BCNF, we must consider not only the explicit FDs, but also the implicit FDs. Thus, the only explicit FD is $AB \rightarrow C$, whose left-hand side is a key. So if we were to consider only the explicit FDs, then we would believe that $\mathbf{R}$ is in BCNF, even though it is not.

We first show that $A \rightarrow C$ is an implicit FD. Let r be an instance of the relation schema. Assume that (a, b_1, c_1) and (a, b_2, c_2) are tuples of r. To show that $A \rightarrow C$ is an implicit FD, we need only show that the dependencies of $\mathbf{R}$ imply that $c_1 = c_2$. From the JD, we have that (a, b_1, c_2) is a tuple of r. Since (a, b_1, c_1) and (a, b_1, c_2) are tuples of r, it follows from the FD $AB \rightarrow C$ that $c_1 = c_2$, as desired.

We now show that $A \rightarrow B$ is not an implicit FD, and so A is not a key. Let r consist of the tuples (a, b_1, c) and (a, b_2, c), where $b_1 \neq b_2$. It is easy to verify that r satisfies the (explicit) dependencies of the relation schema, and so is an instance of the relation schema, but the FD $A \rightarrow B$ fails. $\square$

As an interesting aside, we note that this example can be turned into a necessary and sufficient condition for an extended version of Heath's Theorem. Heath's Theorem [11] says that if X and Y are subsets of the set $\mathcal{A}$ of attributes, and Z is the set difference $\mathcal{A} \setminus (X \cup Y)$, then the FD $X \rightarrow Y$ implies the JD $\bowtie\{XY, XZ\}$. It is easy to see that the converse is false. However, an extended version of Heath's Theorem does have a valid converse. Specifically, we have the following proposition.

Proposition 3.2. *Assume that* X *and* Y *are subsets of the set* $\mathcal{A}$ *of attributes, and* Z *is the set difference* $\mathcal{A} \setminus (X \cup Y)$ *The following are equivalent:*

1. $X \rightarrow Y$
2. $\bowtie\{XY, XZ\}$ *and* XZ *is a superkey (i.e.,* $XZ \rightarrow Y$*).*

Proof. Assume first that (1) holds. Then $\bowtie\{XY, XZ\}$ holds, by Heath's Theorem. Also, since $X \rightarrow Y$ holds, so does $XZ \rightarrow Y$.

Conversely, assume that (2) holds. Then by the same argument as we gave in Example 3.1, where the roles of A, B, and C of Example 3.1 are played by X, Z, and $Y \setminus X$, respectively, it follows that (1) holds. $\square$

We will not make explicit use of this section later. However, it is important for the reader to keep in mind that, as Example 3.1 shows, we cannot simply look only at the explicit dependencies in order to determine whether a relation schema is in some specific normal form; instead, we must take into account the implicit dependencies also

4. SYNTACTICAL CHARACTERIZATION OF ETNF: PROOF

In this section, we present our proof of Theorem 1.13, which gives a syntactic characterization of ETNF, and which is our main technical result. As the first step in the proof, we give an algorithm that determines, given a relation schema $\mathbf{R}$, whether $\mathbf{R}$ has an instance with a fully redundant tuple. This algorithm is based on the chase [1, 13]. As is usual with algorithms that do the chase using JDs, the algorithm runs in time exponential in the size of the dependencies. The only reason we are presenting the algorithm is that we use it as a tool in our proofs, and in particular to help prove Theorem 1.13.

4.1 Full nonredundancy algorithm

Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be the set of attributes. For each attribute A_i, we create a *distinguished variable* a_i. We assume that there are an arbitrary number of nondistinguished variables for each attribute (none of which equals the distinguished variable).

As we mentioned, the algorithm we shall give is based on the chase. For convenience, instead of using a tableau as in [1, 13], we shall use a relation, whose entries consist not of constants but of variables (distinguished and nondistinguished).[8] During the course of the chase, we may replace certain variables that appear as entries in tuples of the relation by other variables, and we may add new tuples to the relation.

Full Nonredundancy Algorithm (Input is a finite set $\mathcal{A} = \{A_1, \ldots, A_n\}$ of attributes, a set $\mathcal{F}$ of FDs over $\mathcal{A}$, and a set $\mathcal{J}$ of JDs over $\mathcal{A}$.)

1. For each JD J in $\mathcal{J}$, do the following.
   (a) If J is $\bowtie\{C_1, \ldots, C_k\}$, then create a relation r_J with k tuples $t_1, \ldots, t_k$ defined as follows. For $1 \leq i \leq k$, let $t_i(A_j)$ be the distinguished variable a_j if $A_j \in C_i$, and otherwise let $t_i(A_j)$ be a new nondistinguished variable.
   (b) Create a total ordering $<$ over all of the variables (distinguished and nondistinguished) that appear in r_J, where every distinguished variable is smaller in the total ordering than every nondistinguished variable (thus, the distinguished variables form an initial prefix of the total ordering).
   (c) Chase r_J with $\mathcal{F}$ and $\mathcal{J}$. During this chase, whenever an FD forces two variables x and y to be equated, if $x < y$ then replace every occurrence of y by x, and otherwise replace every occurrence of x by y. In particular, whenever a distinguished variable x and a nondistinguished variable y are to be equated, then every occurrence of y is replaced by x. Note that no two distinguished variables are ever forced to be equated during the chase, since a column (that is, the set of values $t(A)$ for a fixed attribute A) cannot contain two different distinguished variables, and the chase forces only variables in the same column to be equated.
   (d) If at the end of the chase, one of the original tuples t_p of r_J has been converted into the tuple t_D of all distinguished variables, then call this process a "success", and proceed to the next JD in $\mathcal{J}$. If not, then halt and reject.
2. If there is success for each of the JDs in $\mathcal{J}$, then accept.

Note in particular that in the Full Nonredundancy Algorithm, we need to consider only explicit JDs J (those in the set $\mathcal{J}$), not implicit JDs (those that are logically implied by the explicit dependencies).

The next theorem shows that the Full Nonredundancy Algorithm is sound and complete.

Theorem 4.1. *The Full Nonredundancy Algorithm accepts if and only if* $\mathbf{R} = (\mathcal{A}, \mathcal{F}, \mathcal{J})$ *has no instance with a fully redundant tuple.*

Proof. Assume first that the algorithm rejects; we shall show that $\mathbf{R}$ has an instance with a fully redundant tuple. Let $\bowtie\{C_1, \ldots, C_k\}$ be the JD J that causes the algorithm to reject, and let r_J and $t_1, \ldots, t_k$ be as in part (a) of step (1) of the algorithm. Let t'_i (for $1 \leq i \leq k$) be what the tuple t_i was converted into (by variables being equated) at the end of the chase of r_J with $\mathcal{F}$ and $\mathcal{J}$. Note that if $t_i(A)$ is distinguished, then so is $t'_i(A)$. Let r_1 be the relation consisting of the tuples $t'_1, \ldots, t'_k$. Since r_1 does not contain the tuple t_D of all distinguished variables, and since whenever $t_i(A)$ is distinguished, then so is $t'_i(A)$, it follows that r_1 does not satisfy the JD $\bowtie\{C_1, \ldots, C_k\}$. Let r_2 be the relation that is the result of the chase of r_J with $\mathcal{F}$ and $\mathcal{J}$. In particular, r_2 contains the tuple t_D of all distinguished variables, because of the JD $\bowtie\{C_1, \ldots, C_k\}$ and because r_2 contains the tuples t'_i, where $t'_i[C_i]$ consists of all distinguished variables. Since r_1 does not contain t_D while r_2 does contain t_D, we have $r_1 \subsetneq r_2$. By the standard theory of the chase [1, 13], r_2 satisfies the dependencies of $\mathbf{R}$. It now follows from Lemma 1.8 that $\mathbf{R}$ has a deletion anomaly. So by Proposition 1.9, we know that $\mathbf{R}$ has an instance with a fully redundant tuple, which was to be shown.

Assume now that the algorithm accepts, but that $\mathbf{R}$ has an instance with a fully redundant tuple; we shall derive a contradiction. Since $\mathbf{R}$ has an instance with a fully redundant tuple, we know by Proposition 1.9 that $\mathbf{R}$ has a deletion anomaly. So there are r_1 and r_2 such that $r_1 \subsetneq r_2$, and such that r_1 does not satisfy the dependencies of $\mathbf{R}$ but r_2 satisfies the dependencies of $\mathbf{R}$. Since $r_1 \subsetneq r_2$, and r_2 satisfies the FDs of $\mathbf{R}$ (because r_2 satisfies all of the dependencies of $\mathbf{R}$), it follows that r_1 satisfies the FDs of $\mathbf{R}$.

Let $\mathcal{D}$ be the active domain of r_1. Thus, $\mathcal{D}$ is the set of all values $t(A_i)$, where t is a tuple of r_1 and A_i is an attribute. We now show that if x and y are distinct members of $\mathcal{D}$, then x and y can never be forced to be equal during a chase of r_1 by the dependencies of $\mathbf{R}$. This is because if this were to happen, then by the standard theory of the chase [1, 13], there could be no relation that includes r_1 as a subrelation and satisfies the dependencies of $\mathbf{R}$. This would contradict the facts that $r_1 \subsetneq r_2$, and r_2 satisfies the dependencies of $\mathbf{R}$.

We now show that r_1 satisfies the explicit JDs of $\mathbf{R}$ (which is a contradiction, since r_1 does not satisfy all of the dependencies of $\mathbf{R}$, and r_1 satisfies the FDs of $\mathbf{R}$). Let J be an explicit JD of $\mathbf{R}$; we must show that r_1 satisfies J. Assume not; we shall derive a contradiction. Since r_1 does not satisfy J, the result of chasing r_1 with J produces a new tuple $\hat{t}$ not in r_1.

Assume that J is the JD $\bowtie\{C_1, \ldots, C_k\}$. Let r_J be as in part (a) of step (1) of the Full Nonredundancy Algorithm, and let t_p and t_D be as in part (d) of step (1) of the Full Nonredundancy Algorithm. Since the result of chasing r_1 with J produces $\hat{t}$, it follows immediately that there is a homomorphism h from r_J to r_1 such that $h(t_D) = \hat{t}$.

Consider the sequence of chase steps on r_J that eventually converts t_p into t_D. Assume that there are m chase steps. Let $r_J^{(i)}$ be the relation after applying chase step i to r_J, for $0 \leq i \leq m$. In particular, $r_J^{(0)}$, the result of doing no chase steps, is r_J.

Let us mimic these chase steps on r_1 to thereby obtain a chase on r_1 with the dependencies of $\mathbf{R}$, as we now describe. We shall let $r_1^{(i)}$ be the result of applying chase step i to r_1, for $0 \leq i \leq m$. Thus, $r_1^{(0)} = r_1$, and $r_1^{(i+1)}$ is the result of doing the same chase step to $r_1^{(i)}$ as that used to obtain $r_J^{(i+1)}$ from $r_J^{(i)}$ (we shall clarify this shortly). We shall maintain the invariants that (1) if tuple s is in $r_J^{(i)}$, then tuple $h(s)$ is in $r_1^{(i)}$, and (2) if tuple s of r_J has been converted by

[8]This abuse of the convention that the entries of a relation are only constants is quite common.

chase step i to tuple s' of $r_J^{(i)}$, then tuple $h(s)$ of r_1 has been converted by chase step i to tuple $h(s')$ of $r_1^{(i)}$. We now clarify how our mimicking process works.

If on chase step $i+1$ the JD $\bowtie\{C'_1, \ldots, C'_q\}$ is applied to the tuples $s_1, \ldots, s_q$ of $r_J^{(i)}$ to obtain the new tuple s that is added to $r_J^{(i)}$ to obtain $r_J^{(i+1)}$, then $r_1^{(i+1)}$ is obtained from $r_1^{(i)}$ by adding the tuple $h(s)$ to $r_1^{(i)}$ (if the tuple $h(s)$ is not already present in $r_1^{(i)}$). It is straightforward to verify, using the first invariant, that this is a legal application of applying the JD $\bowtie\{C'_1, \ldots, C'_q\}$ to the tuples $h(s_1), \ldots, h(s_q)$ of $r_J^{(i)}$. If on chase step $i+1$ an FD F converts x in $r_J^{(i)}$ to y, we mimic this by converting $h(x)$ in $r_1^{(i)}$ into $h(y)$. As before, it is straightforward to verify, using the first invariant, that this is a legal application of the FD F. Further, it is straightforward to verify that in both cases, the invariants are maintained. Since the tuple t_p of r_J is eventually converted into t_D in the chase on r_J, it follows from the second invariant that the tuple $h(t_p)$ of r_1 is eventually converted into $h(t_D)$ in the chase on r_1 we have described that mimics the chase on r_J.

But it is not possible for a chase to convert the tuple $h(t_p)$ of r_1 into $h(t_D)$, since (1) $h(t_p) \neq h(t_D)$ (because $h(t_p)$ is in r_1, whereas $h(t_D)$ is not in r_1, since $h(t_D) = \hat{t}$, and $\hat{t}$ is not in r_1), and (2) every entry of $h(t_p)$ and $h(t_D)$ is a member of $\mathcal{D}$, and we showed that whenever x and y are distinct members of $\mathcal{D}$, then x and y can never be forced to be equal during a chase of r_1 by the dependencies of $\mathbf{R}$. This contradiction completes the proof. $\square$

4.2 Completion of the proof of Theorem 1.13

We now complete the proof of Theorem 1.13, by making use of Theorem 4.1, which shows the soundness and completeness of the Full Nonredundancy Algorithm.

Assume first that (1) $\mathbf{R}$ is in BCNF, and (2) for every JD in $\mathcal{J}$, some component is a superkey of $\mathbf{R}$. We must show that $\mathbf{R}$ is in ETNF. We first show that the Full Nonredundancy Algorithm accepts. Let J be the JD $\bowtie\{C_1, \ldots, C_k\}$ of $\mathbf{R}$. By condition (2), there is q such that the component C_q of J is a superkey of $\mathbf{R}$. Let $t_1, \ldots, t_k$ be as in part (a) of step (1) of the Full Nonredundancy Algorithm, and let t_D be as in part (d) of step (1) of the Full Nonredundancy Algorithm. In particular, t_q is the tuple corresponding to the component C_q that is a superkey. By construction, $t_q[C_q] = t_D[C_q]$. Applying the JD $\bowtie\{C_1, \ldots, C_k\}$ to the tuples $t_1, \ldots, t_k$ in the chase produces the tuple t_D. Since C_q is a superkey, and since $t_q[C_q] = t_D[C_q]$, the chase causes the tuple t_q to be converted into the tuple t_D. So the Full Nonredundancy Algorithm succeeds on the JD J. Since J is an arbitrary JD of $\mathbf{R}$, the Full Nonredundancy Algorithm accepts, as desired. Therefore, by Theorem 4.1, we have that $\mathbf{R}$ has no instance with a fully redundant tuple. Further, since $\mathbf{R}$ is also in BCNF, we know by Proposition 1.5 that $\mathbf{R}$ has no instance with a partly redundant tuple. It follows that every tuple in every instance of $\mathbf{R}$ is essential. Therefore, $\mathbf{R}$ is in ETNF, which was to be shown.

We now show the converse. Assume that $\mathbf{R}$ is in ETNF. So in particular, $\mathbf{R}$ has no instance with a partly redundant tuple. Therefore, by Proposition 1.5, we know that $\mathbf{R}$ is in BCNF. Let J be an explicit JD $\bowtie\{C_1, \ldots, C_k\}$ of $\mathbf{R}$. We must show that some C_i is a superkey.

Since $\mathbf{R}$ is in ETNF, it has no instance with a fully redundant tuple. Therefore, by Theorem 4.1, we know that the Full Nonredundancy Algorithm accepts when run on $\mathbf{R}$. Hence, step (1) of the Full Nonredundancy Algorithm succeeds when run on J. Let r_J and $t_1, \ldots, t_k$ be as in part (a) of step (1) of the Full Nonredundancy Algorithm, and let t_p and t_D be as in part (d) of step (1) of the Full Nonredundancy Algorithm. We now show that the corresponding component C_p of J is a superkey of $\mathbf{R}$.

Let r_1 be a relation consisting only of the two tuples t_p and t_D. Note that t_p and t_D agree precisely on the set C_p of attributes. Therefore, it follows from the standard theory of the chase [1, 13] that to show that C_p is a superkey of $\mathbf{R}$, we need only show that during a chase of r_1 with the dependencies of $\mathbf{R}$, the tuple t_p is eventually converted into the tuple t_D. Let h be a function defined on the distinguished and nondistinguished variables of r_J, defined as follows. First, we let $h(x) = x$ when x is a variable in t_D (that is, a distinguished variable) or a variable in t_p. For each remaining nondistinguished variable, if x is a nondistinguished variable for the attribute A_i, let $h(x)$ be the distinguished variable a_i for attribute A_i. It follows from the definition of h that $h(t_p) = t_p$, and $h(t_i) = t_D = h(t_D)$ for $i \neq p$. In particular, $h(t) \in r_1$ for each tuple t of r_J. So h is a homomorphism mapping r_J into r_1.

Then exactly as in the proof of Theorem 4.1, we can take the chase of r_J that converts t_p into t_D, and mimic this chase in r_1, to get a chase that converts $h(t_p)$ into $h(t_D)$. But $h(t_p) = t_p$, and $h(t_D) = t_D$. So this chase of r_1 converts t_p into t_D. This is exactly what was to be shown.

5. SIMPLE SUFFICIENT CONDITION FOR ETNF

In [7], some simple conditions, which we now describe, are given that guarantee higher normal forms. It is shown in [7] that if a relation schema $\mathbf{R}$ is in 3NF, and every key has only one attribute, then $\mathbf{R}$ is in 5NF. It is also shown that if a relation schema $\mathbf{R}$ is in BCNF and some key has only one attribute, then $\mathbf{R}$ is in 4NF. We now show that the conclusion of this latter result can be strengthened to say that $\mathbf{R}$ is in ETNF.

THEOREM 5.1. *Let* $\mathbf{R}$ *be a BCNF relation schema specified only by FDs and JDs, such that some key has only one attribute. Then* $\mathbf{R}$ *is in ETNF.*

PROOF. Let A be an attribute that is a key of $\mathbf{R}$, and let J be a JD of $\mathbf{R}$. Then A is contained in some component C of J, since the union of the components of J is the set of all attributes. Therefore, C is a superkey. It then follows from Theorem 1.13 that $\mathbf{R}$ is in ETNF. $\square$

The next example shows that we cannot strengthen Theorem 5.1 by replacing "ETNF" in the conclusion by "RFNF".

EXAMPLE 5.2. Let $\mathbf{R}$ be the relation schema with

$$\mathcal{A} = \{A, B, C, D\}\,,$$

$$\mathcal{F} = \{A \to BCD, BC \to AD\}\,,$$

$$\mathcal{J} = \{\bowtie\{ABC, CD, BD\}\}\,.$$

We shall show that $\mathbf{R}$ satisfies the hypotheses of Theorem 5.1, but $\mathbf{R}$ is not in RFNF.

We first show that $\mathbf{R}$ is in BCNF. Let $X \to Y$ be a nontrivial FD of $\mathbf{R}$. We shall show that X is a superkey, by showing that X necessarily contains either A or BC.

If X has exactly 3 members, then either X contains A, or X is BCD. In both cases, X contains either A or BC, as desired.

Assume now that X has exactly 2 members. It is easy to see that for X not to contain either A or BC, necessarily X must be one of BD or CD. We shall show that X cannot be BD; a symmetric argument shows that X cannot be CD. Let r be the relation with tuples (a, b, c, d) and (a', b, c', d), where $a \neq a'$ and $c \neq c'$. It is straightforward to verify that r satisfies the explicit dependencies of $\mathbf{R}$. However, r does not satisfy any nontrivial FD with left-hand side BD. This shows that X cannot be BD, as desired.

Assume now that X has exactly one member. If X is A, we are done, so assume that X is B, C, or D. We first show that X cannot be B; a symmetric argument shows that X cannot be C. Let r be the relation with tuples (a, b, c, d) and (a', b, c', d'), where $a \neq a'$, $c \neq c'$, and $d \neq d'$. It is straightforward to verify that r satisfies the explicit dependencies of $\mathbf{R}$. However, r does not satisfy any nontrivial FD with left-hand side B. This shows that X cannot be B, as desired. Now consider the case where X is D. Here we let r be the relation with tuples (a, b, c, d) and (a', b', c', d), where $a \neq a'$, $b \neq b'$, and $c \neq c'$. It is straightforward to verify that r satisfies the explicit dependencies of $\mathbf{R}$. However, r does not satisfy any nontrivial FD with left-hand side D. This shows that X cannot be D, as desired.

Assume now that X is the empty set. We can assume without loss of generality that the right-hand side Y is a singleton. If this singleton is B, so that $\emptyset \to B$ is an implicit FD, then so is $C \to B$, whereas we showed that there is no nontrivial implicit FD with C as the left-hand side. If this singleton is not B, then $B \to Y$ is a nontrivial implicit FD, whereas we showed that there is no nontrivial implicit FD with B as the left-hand side. This concludes the proof that $\mathbf{R}$ is in BCNF.

So $\mathbf{R}$ satisfies the hypotheses of Theorem 5.1. We conclude this example by showing that $\mathbf{R}$ is not in RFNF. Let J be the JD $\bowtie\{ABC, CD, BD\}$ of $\mathbf{R}$. From what we have shown, it follows that the only component of J that is a superkey is ABC. So D is an attribute of $\mathbf{R}$ that is not in any component of J that is a superkey. This shows that J is not in the syntactically defined normal form KCNF of Definition 1.18. So by Theorem 1.19, it follows that $\mathbf{R}$ is not in RFNF, which was to be shown. $\square$

6. RELATIONSHIPS BETWEEN NORMAL FORMS

In this section, we prove the following theorem, which gives the relationships among the normal forms. As we noted in Section 1.2, parts of this theorem were already known, but we are including proofs of those parts to help make this paper more self-contained.

Theorem 6.1. *5NF ⇒ SKNF ⇒ RFNF ⇒ ETNF ⇒ 4NF. None of the reverse implications hold.*

Proof. **5NF ⇒ SKNF**: Assume that $\mathbf{R}$ is in 5NF. We now show that $\mathbf{R}$ is in SKNF. Let $\bowtie\{C_1, \ldots, C_k\}$ be an irreducible JD of $\mathbf{R}$; we must show that every C_i is a superkey of $\mathbf{R}$.

Since $\mathbf{R}$ is in 5NF, every JD of $\mathbf{R}$ is logically implied by the keys, and so by Proposition 2.4, the Membership Algorithm accepts for the set $\mathcal{A}$ of attributes, the JD $\bowtie\{C_1, \ldots, C_k\}$, and the set $\{K_1, \ldots, K_r\}$ of keys of $\mathbf{R}$. Let $C_{i_1}, \ldots, C_{i_s}$ be the components of the JD $\bowtie\{C_1, \ldots, C_k\}$ that, during the course of running the Membership Algorithm, were removed from S and eventually unioned together to obtain $\mathcal{A}$ as a final member of S. Then each of $C_{i_1}, \ldots, C_{i_s}$ is a superkey, as we see from the specifications of the Membership Algorithm. Now $\bowtie\{C_{i_1}, \ldots, C_{i_s}\}$ is a JD of the schema $\mathbf{R}$, since it is logically implied by the keys (because the Membership Algorithm accepts for the set $\mathcal{A}$ of attributes, the JD $\bowtie\{C_{i_1}, \ldots, C_{i_s}\}$, and the set $\{K_1, \ldots, K_r\}$ of keys of $\mathbf{R}$), Therefore, since $\bowtie\{C_1, \ldots, C_k\}$ is an irreducible JD of $\mathbf{R}$ necessarily every C_i (for $1 \leq i \leq k$) is a member of $\{C_{i_1}, \ldots, C_{i_s}\}$. Since each of $C_{i_1}, \ldots, C_{i_s}$ is a superkey, it follows that each of $C_1, \ldots, C_k$ is a superkey. Hence, $\mathbf{R}$ is in SKNF, as desired.

SKNF ⇒ RFNF: Assume that $\mathbf{R}$ is in SKNF. To show that $\mathbf{R}$ is in RFNF, we must show (by Theorem 1.19) that $\mathbf{R}$ is in KCNF. We first show that $\mathbf{R}$ is in BCNF. Assume that $\mathbf{R}$ is not in BCNF; we shall show that $\mathbf{R}$ is not in SKNF, which is a contradiction. Since $\mathbf{R}$ is not in BCNF, there is an explicit or implicit FD $X \to A$ of $\mathbf{R}$ that is not implied by the keys, and so X is not a superkey of $\mathbf{R}$. Without loss of generality, we can assume that A is a single attribute. Let Y be the set of attributes not in $X \cup \{A\}$. Now Y is nonempty, since X is not a superkey. Since $X \to A$ is an FD of $\mathbf{R}$, we know that $\bowtie\{XA, XY\}$ is a JD of $\mathbf{R}$ (by Heath's Theorem [11]). This JD is irreducible, since if it were reducible, then either $\bowtie\{XA\}$ or $\bowtie\{XY\}$ would be an explicit or implicit JD; however, neither is a valid JD, since neither contains all of the attributes (XA does not contain the attributes of Y, and XY does not contain the attribute A). Now XA is not a superkey of $\mathbf{R}$, since if it were, then X would be a superkey of $\mathbf{R}$, because $X \to A$ is an FD of $\mathbf{R}$. So $\bowtie\{XA, XY\}$ is an irreducible JD of $\mathbf{R}$ where the component XA is not a superkey of $\mathbf{R}$. Hence, $\mathbf{R}$ is not in SKNF, as desired. This completes the proof that $\mathbf{R}$ is in BCNF.

Let J be an arbitrary JD of $\mathbf{R}$. To complete the proof that $\mathbf{R}$ is in KCNF (and hence RFNF), we must show that the union of the components of J that are superkeys includes every attribute of $\mathbf{R}$. It is easy to see that there is an irreducible JD J' of $\mathbf{R}$ such that every component of J' is a component of J (J' could be an explicit or implicit JD of $\mathbf{R}$, and J' could be J itself). Since J' is a JD of $\mathbf{R}$, it follows by definition of SKNF that every component of J' is a superkey. Hence, the union of the components of J' that are superkeys includes every attribute of $\mathbf{R}$. Since every component of J' is a component of J, it follows that the union of the components of J that are superkeys includes every attribute of $\mathbf{R}$. This was to be shown.

RFNF ⇒ ETNF: Assume that $\mathbf{R}$ is in RFNF. So by Theorem 1.19, it follows that $\mathbf{R}$ is in KCNF. Therefore $\mathbf{R}$ is in BCNF. Hence, by Theorem 1.13, to show that $\mathbf{R}$ is in ETNF, we need only show that some component of every explicit JD of $\mathbf{R}$ is a superkey. Let J be an explicit JD of $\mathbf{R}$. Since $\mathbf{R}$ is in KCNF, the union of the components of J that are superkeys includes every attribute of $\mathbf{R}$. In particular, some component of J is a superkey. This was to be shown.

ETNF ⇒ 4NF: Assume that $\mathbf{R}$ is in ETNF. Let $X \to\to Y|Z$ be a nontrivial MVD of $\mathbf{R}$. As we discussed earlier, we can assume without loss of generality that X, Y and Z are pairwise disjoint (and have union the set of all attributes), and Y and Z are nonempty. It is sufficient to show that X is a superkey, since then the keys of $\mathbf{R}$ logically imply

$X \to\to Y|Z$.

The MVD $X \to\to Y|Z$ is logically equivalent to the JD $\bowtie\{XY, XZ\}$. Let t_1 be a tuple with a distinguished variable for each member of XY, and a new nondistinguished variable for each attribute in Z. Similarly, let t_2 be a tuple with a distinguished variable for each member of XZ, and a new nondistinguished variable for each attribute in Y. Let r be the relation consisting of the tuples t_1 and t_2. Since the JD $\bowtie\{XY, XZ\}$ is one of the JDs of $\mathbf{R}$, we can assume without loss of generality that it is one of the explicit JDs of $\mathbf{R}$ (we can add it in if needed to the explicit JDs). Since there is a success when step (1) of the Full Nonredundancy Algorithm is applied to the JD $\bowtie\{XY, XZ\}$, we know that at the end of the chase, either t_1 or t_2 has been converted into the tuple t_D of all distinguished variables. Assume without loss of generality that t_1 has been converted into the tuple t_D of all distinguished variables. It follows from the standard theory of the chase [1, 13] that this implies that the FD $X \to Z$ is an explicit or implicit FD of the relation schema. Since $\mathbf{R}$ is in BCNF (because it is in ETNF), it follows that this FD $X \to Z$ is logically implied by the keys. But then, by Proposition 2.1, it follows that X is a superkey, as desired.

This completes the proof of the implications. We now show that none of the reverse implications hold.

SKNF $\not\Rightarrow$ 5NF: Let $\mathbf{R}$ be the relation schema with

$$\mathcal{A} = \{A, B, C\},$$

$$\mathcal{F} = \{AB \to C, AC \to B, BC \to A\},$$

$$\mathcal{J} = \{\bowtie\{AB, AC, BC\}\}.$$

We shall show that $\mathbf{R}$ is in SKNF but not 5NF.

We first show that $\mathbf{R}$ is in SKNF. Let $\bowtie\{C_1, \ldots, C_k\}$ be an irreducible JD of $\mathbf{R}$ where some C_i (say C_1, without loss of generality) is not a superkey of $\mathbf{R}$; we shall derive a contradiction. Since C_1 is not a superkey of $\mathbf{R}$, and since $C_1 \neq \emptyset$ (because the JD is irreducible), we see (by looking at $\mathcal{F}$) that C_1 is a singleton, say A (without loss of generality). The only possibilities for this JD are either $\bowtie\{A, B, C\}$ or $\bowtie\{A, BC\}$. Since $\bowtie\{A, B, C\}$ logically implies $\bowtie\{A, BC\}$, we need only show that $\bowtie\{A, BC\}$ is not an implicit JD of $\mathbf{R}$ to derive our desired contradiction. Let t_1 be the tuple (a_1, b_1, c_1) and let t_2 be the tuple (a_2, b_2, c_2), where $a_1 \neq a_2$, $b_1 \neq b_2$, and $c_1 \neq c_2$. Let r be the relation consisting of the two tuples t_1 and t_2. It is straightforward to verify that r satisfies the dependencies in $\mathcal{F}$ and $\mathcal{J}$. However, r does not satisfy the JD $\bowtie\{A, BC\}$, since (a_1, b_2, c_2) is not a tuple of r. So $\bowtie\{A, BC\}$ is not an implicit JD of $\mathbf{R}$, as desired.

We have shown that $\mathbf{R}$ is in SKNF. We now show that $\mathbf{R}$ is not in 5NF. We begin by showing that the only keys of $\mathbf{R}$ are the doubletons AB, AC, and BC. We need only show that none of the singletons A, B, or C is a superkey (this implies also that the empty set is not a superkey). By symmetry in the roles of A, B, and C, we need only show that A is not a superkey. Let t_1 be the tuple (a, b_1, c_1) and let t_2 be the tuple (a, b_2, c_2), where $b_1 \neq b_2$ and $c_1 \neq c_2$. Let r be the relation consisting of the two tuples t_1 and t_2. It is straightforward to verify that $\mathbf{R}$ satisfies the dependencies in $\mathcal{F}$ and $\mathcal{J}$. However, r does not satisfy the FD $A \to B$. It follows that $A \to B$ is not an FD of $\mathbf{R}$, so A is not a superkey, as desired.

To show that $\mathbf{R}$ is not in 5NF, we need only show that the JD $\bowtie\{AB, AC, BC\}$ of $\mathbf{R}$ is not logically implied by the keys of $\mathbf{R}$. We have already shown that the only keys of $\mathbf{R}$ are AB, AC, and BC, so we need only show that the JD $\bowtie\{AB, AC, BC\}$ of $\mathbf{R}$ is not logically implied by the FDs $AB \to C$, $AC \to B$, and $BC \to A$. Let r be the relation consisting of the three tuples $(a, b, c_1), (a, b_1, c), (a_1, b, c)$, where $a \neq a_1$, $b \neq b_1$, and $c \neq c_1$. Clearly r satisfies the FDs $AB \to C$, $AC \to B$, and $BC \to A$. However, since r does not contain the tuple (a, b, c), it follows that r does not satisfy the JD $\bowtie\{AB, AC, BC\}$. Hence, this JD is not logically implied by the FDs $AB \to C$, $AC \to B$, and $BC \to A$, as desired.

RFNF $\not\Rightarrow$ SKNF: Let $\mathbf{R}$ be the relation schema with

$$\mathcal{S} = \{S, P, J\},$$

$$\mathcal{F} = \{SP \to J, PJ \to S\},$$

$$\mathcal{J} = \{\bowtie\{SP, PJ, JS\}\}.$$

We shall show that $\mathbf{R}$ is in RFNF but not SKNF.

We first show that $\mathbf{R}$ is in RFNF. By Theorem 1.19, we need only show that $\mathbf{R}$ is in KCNF. We first show that $\mathbf{R}$ is in BCNF. It is easy to see that to show this, we need only show that if $X \to Y$ is a nontrivial implicit FD of $\mathbf{R}$, then X is not a singleton (this implies also that X cannot be the empty set). We now show that X cannot be $\{S\}$; similar arguments show that X cannot be either $\{P\}$ or $\{J\}$. The relation that consists of the tuples (s, p, j) and (s, p_1, j_1) with $p \neq p_1$ and $j \neq j_1$ satisfies the dependencies of $\mathbf{R}$, but does not satisfy either of the FDs $S \to P$ or $S \to J$. So indeed, X cannot be $\{S\}$. This completes the proof that $\mathbf{R}$ is in BCNF.

We now show that there are no nontrivial implicit JDs of $\mathbf{R}$. We need only show that there are no nontrivial *irreducible* implicit JDs of $\mathbf{R}$. Let K be a nontrivial irreducible implicit JD of $\mathbf{R}$ (we use K instead of J, since J is being used as an attribute). The only candidate for K with no doubleton component is $\bowtie\{S, P, J\}$. The only candidates for K with one doubleton component are $\bowtie\{SP, J\}$, $\bowtie\{SJ, P\}$ and $\bowtie\{PJ, S\}$. The only candidates for K with two doubleton components are $\bowtie\{SP, PJ\}$, $\bowtie\{SJ, PJ\}$ and $\bowtie\{SP, SJ\}$. We shall show that none of the three candidates with two doubleton components are an implicit JD of $\mathbf{R}$. This implies that none of the other four candidates are implicit JDs of $\mathbf{R}$, since each of the other four candidates implies one of the candidates with two doubleton components (for example, $\bowtie\{SP, J\}$ implies $\bowtie\{SP, PJ\}$). We now show that $\bowtie\{SP, PJ\}$ is not an implicit JD of $\mathbf{R}$; symmetric arguments show that neither of the other two candidates with two doubleton components are implicit JDs of $\mathbf{R}$. The relation that consists of the tuples (s, p, j) and (s_1, p, j_1) with $s \neq s_1$ and $j \neq j_1$ satisfies the dependencies of $\mathbf{R}$, but does not satisfy $\bowtie\{SP, PJ\}$, so indeed $\bowtie\{SP, PJ\}$ is not an implicit JD of $\mathbf{R}$. Hence, there are no nontrivial implicit JDs of $\mathbf{R}$, but only the explicit JD $\bowtie\{SP, PJ, JS\}$. Since SP and PJ are superkeys, the union of the components of this JD that are superkeys includes every attribute of $\mathbf{R}$. So $\mathbf{R}$ is in KCNF, as desired.

We now show that $\mathbf{R}$ is not in SKNF. It follows from what we have shown that $\bowtie\{SP, PJ, JS\}$ is irreducible, and so we need only show that JS is not a superkey of $\mathbf{R}$. The fact that JS is not a superkey of $\mathbf{R}$ follows from the fact that the relation that consists of the tuples (s, p, j) and (s, p_1, j) with $p \neq p_1$ satisfies the dependencies of $\mathbf{R}$, but not the FD $JS \to P$.

ETNF $\not\Rightarrow$ RFNF: The alert reader will observe that this was already demonstrated by the relation schema of Example 5.2, which we showed satisfies the hypotheses of Theorem 5.1 (and hence, by Theorem 5.1, is in ETNF), but which we showed in Example 5.2 is not in RFNF.

Since we promised in Section 1.1 that we would show that the relation schema $\mathbf{R}'$ in Example 1.2 is another relation schema in ETNF but not RFNF, we now do so.

Let $\mathbf{R}'$ be this relation schema. Thus, $\mathbf{R}'$ is the relation schema with

$$\mathcal{S} = \{S, P, J\}\,,$$

$$\mathcal{F} = \{SP \rightarrow J\}\,,$$

$$\mathcal{J} = \{\bowtie\{SP, PJ, JS\}\}\,.$$

We shall show that $\mathbf{R}'$ is in ETNF but not RFNF.

We first show that $\mathbf{R}'$ is in ETNF. We begin by showing that $\mathbf{R}'$ is in BCNF. To prove this, we need only show that there are no nontrivial implicit FDs of $\mathbf{R}'$ (so that the only FD is the explicit FD $SP \rightarrow J$). Since SP is a key, this will show that $\mathbf{R}'$ is in BCNF. The only FDs with a doubleton left-hand side that we need to consider (that is, to show are not implicit FDs) are $PJ \rightarrow S$ and $JS \rightarrow P$. In addition, we then need only consider the FDs $S \rightarrow J$ and $P \rightarrow J$, since the other nontrivial FDs (such as $S \rightarrow P$) logically imply one of the FDs $PJ \rightarrow S$ and $JS \rightarrow P$. Thus, to show that $\mathbf{R}'$ is in BCNF, we need only show that none of the FDs $PJ \rightarrow S$, $JS \rightarrow P$, $S \rightarrow J$, and $P \rightarrow J$ are FDs of $\mathbf{R}'$. Therefore, by symmetry in the roles of S and P, we need only show that neither of the FDs $PJ \rightarrow S$ and $S \rightarrow J$ are FDs of $\mathbf{R}'$. Now $PJ \rightarrow S$ is not an FD of $\mathbf{R}'$, since the relation consisting of the tuples (s, p, j) and (s_1, p, j), where $s \neq s_1$, satisfies the dependencies of $\mathbf{R}'$ but not the FD $PJ \rightarrow S$. And $S \rightarrow J$ is not an FD of $\mathbf{R}'$, since the relation consisting of the tuples (s, p, j) and (s, p_1, j_1), where $p \neq p_1$ and $j \neq j_1$, satisfies the dependencies of $\mathbf{R}'$ but not the FD $S \rightarrow J$. This completes the proof that $\mathbf{R}'$ is in BCNF. Hence, by Proposition 1.5, $\mathbf{R}'$ has no instance with a partly redundant tuple.

We now show that $\mathbf{R}'$ has no instance with a fully redundant tuple, which completes the proof that $\mathbf{R}'$ is in ETNF. By Theorem 4.1, we need only show that the Full Nonredundancy Algorithm accepts. Let K be the JD $\bowtie\{SP, PJ, JS\}$, let t_1 be the tuple (s, p, j_1), let t_2 be the tuple (s, p_1, j), and let t_3 be the tuple (s_1, p, j), where s, p, and j are distinguished variables, and where s_1, p_1, and j_1 are new nondistinguished variables. Then r_K (which plays the role of r_J in part (a) of step (1) of the Full Nonredundancy Algorithm) is the relation with tuples t_1, t_2, and t_3. When we chase r_K with K, we obtain the tuple $t_D = (s, p, j)$ of all distinguished variables. When we then chase with the FD $SP \rightarrow J$, the tuple t_1 is converted to the tuple t_D. So the Full Nonredundancy Algorithm accepts, as desired.

We have shown that $\mathbf{R}'$ is in ETNF. We now show that $\mathbf{R}'$ is not in RFNF. By Theorem 1.19, we need only show that $\mathbf{R}'$ is not in KCNF. Again, let J be the JD $\bowtie\{SP, PJ, JS\}$ of $\mathbf{R}'$. To show that $\mathbf{R}$ is not in KCNF, we need only show that neither PJ nor JS is a superkey of $\mathbf{R}'$. But we already noted that neither $PJ \rightarrow S$ nor $JS \rightarrow P$ are FDs of $\mathbf{R}'$. So indeed, $\mathbf{R}'$ is not in KCNF.

4NF $\not\Rightarrow$ ETNF: Let $\mathbf{R}'$ be the relation schema with

$$\mathcal{A} = \{A, B, C\}\,,$$

$$\mathcal{F} = \emptyset,$$

$$\mathcal{J} = \{\bowtie\{AB, AC, BC\}\}\,.$$

We shall show that $\mathbf{R}'$ is in 4NF but not ETNF.

It was shown in [9] that $\mathbf{R}'$ is in 4NF but not 5NF. Furthermore, $\mathbf{R}'$ is not in ETNF, since the Full Nonredundancy Algorithm rejects $\mathbf{R}'$ (because there are no FDs to equate variables). This concludes the proof. □

7. CONCLUSIONS

We have introduced a new normal form, called *essential tuple normal form (ETNF)*, which is between 4NF and 5NF. We argue that if the goal of the database designer is to prevent redundant tuples, then contrary to common belief, 5NF is not needed, but instead ETNF is what is called for. ETNF is defined semantically, in terms of lack of redundancy of tuples. We prove a syntactic characterization of ETNF, which says that a relation schema is in ETNF if and only if it is in BCNF and some component of every JD is a superkey. Interestingly, this syntactic characterization is very similar to that of two other intermediate normal forms. The first such normal form arose as an erroneous definition in various textbooks (including [6]) of 5NF; the normal form given by this erroneous definition of 5NF is called *superkey normal form (SKNF)* by Normann [14] and *5NFR* by Vincent [18]. It says that every component of every irreducible JD is a superkey. Our syntactic characterization of ETNF is also similar to the syntactic characterization of a normal form due to Vincent [17] called *redundancy-free normal form (RFNF)*, that captures an even stronger notion of lack of redundancy than what ETNF captures. This syntactic characterization of RFNF says that a relation schema is in RFNF if and only if it is in BCNF and for every JD J of $\mathbf{R}$, the union of the components of J that are superkeys includes every attribute of $\mathbf{R}$. We show that 5NF $\Rightarrow$ SKNF $\Rightarrow$ RFNF $\Rightarrow$ ETNF $\Rightarrow$ 4NF, and that none of the reverse implications hold.

We also give a simple sufficient condition for ETNF. Specifically, we show that if a relation schema $\mathbf{R}$ is in BCNF and some key has only one attribute, then $\mathbf{R}$ is in ETNF. This is nice, because it gives a simple, natural condition involving FDs only (that is, not involving JDs) that guarantees ETNF. We show that this condition is not strong enough to guarantee RFNF.

Acknowledgments

The authors are grateful to Marcelo Arenas, Leonid Libkin, and Millist Vincent for valuable discussions. We also thank Ragnar Normann for some helpful comments.

8. REFERENCES

[1] A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems (TODS)*, 4(3):297–314, 1979.

[2] M. Arenas and L. Libkin. An information-theoretic approach to normal forms for relational and XML data. *Journal of the Association for Computing Machinery (JACM)*, 52(2):246–283, 2005.

[3] E. F. Codd. Further normalization of the database relational model. In R. Rustin, editor, *Courant Computer Science Symposium*, volume 6 - Data Base Systems, pages 33–64. Prentice-Hall, 1972.

[4] E. F. Codd. Recent investigations in relational data base systems. In *Proc. IFIP Congress 74*, pages 1017–1021. North-Holland, 1974.

[5] E. F. Codd and C. J. Date. Interactive support for non-programmers: the relational and network approaches. In R. J. Rustin, editor, *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control: Data Models: Data-Structure-Set versus Relational*, SIGFIDET '74, pages 11–41, New York, NY, USA, 1975. ACM.

[6] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, eighth edition, 2004.

[7] C. J. Date and R. Fagin. Simple conditions for guaranteeing higher normal forms for relational databases. *ACM Transactions on Database Systems (TODS)*, 17(3), 1992. Reprinted in "Relational database, writings 1989 – 1991" by C. J. Date with H. Darwen, Addison Wesley, 1992.

[8] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems (TODS)*, 2(3):262–278, Sept. 1977.

[9] R. Fagin. Normal forms and relational database operators. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 153–160, 1979.

[10] R. Fagin. A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems (TODS)*, 6(3):387–415, 1981.

[11] I. J. Heath. Unacceptable file operations in a relational data base. In *Proc. ACM SIGFIDET Workshop on Data Description, Access and Control*, pages 19–33, 1971.

[12] D. Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[13] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems (TODS)*, 4(4):455–469, 1979.

[14] R. Normann. Minimal lossless decompositions and some normal forms between 4NF and PJ/NF. *Information Systems*, 23(7):509–516, 1998.

[15] J. Rissanen. Theory of relations for databases–a tutorial survey. In *7th Symposium on Mathematical Foundations of Computer Science,* Lecture Notes in Computer Science, 64, pages 537–551, 1978.

[16] B. Thalheim. Deductive normal forms of relations. In W. Bibel and K. P. Jantke, editors, *Mathematical Methods of Specification and Synthesis of Software Systems*, volume 215 of *Lecture Notes in Computer Science*, pages 226–230. Springer, 1985.

[17] M. W. Vincent. Redundancy elimination and a new normal form for relational database design. In L. Libkin and B. Thalheim, editors, *Semantics in Databases*, volume 1358 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 1995.

[18] M. W. Vincent. A corrected 5NF definition for relational database design. *Theoretical Computer Science (TCS)*, 185(2):379–391, 1997.

[19] M. W. Vincent, J. Liu, and M. K. Mohania. On the equivalence between FDs in XML and FDs in relations. *Acta Informatica*, 44(3–4):207–247, 2007.