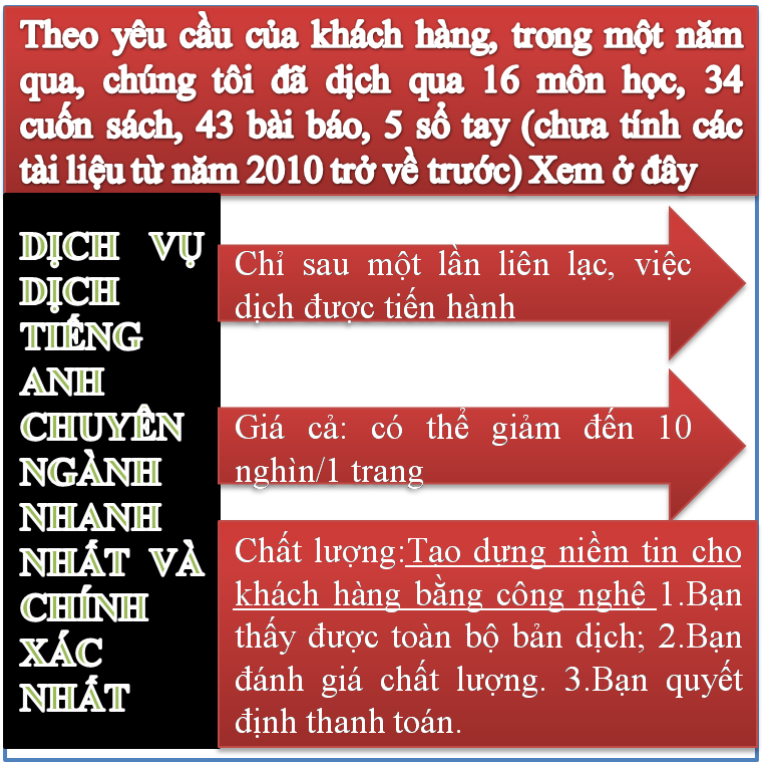

Tài liệu này được dịch sang tiếng việt bởi:

www.mientayvn.com

Tìm bản gốc tại thư mục này (copy link và dán hoặc nhấn Ctrl+Click):

https://drive.google.com/folderview?id=0B4rAPqlxIMRDSFE2RXQ2N3FtdDA&usp=sharing

Liên hệ để mua:

thanhlam1910_2006@yahoo.com hoặc frbwrthes@gmail.com hoặc số 0168 8557 403 (gặp Lâm)

Giá tiền: 1 nghìn /trang đơn (trang không chia cột); 500 VND/trang song ngữ

Dịch tài liệu của bạn: http://www.mientayvn.com/dich_tieng_anh_chuyen_nghanh.html

A BASIS FOR DEDUCTIVE DATABASE SYSTEMS

J. w. LLOYD AND R. w. TOPOR

This paper provides a theoretical basis for deductive database systems. A deductive database consists of closed typed first order logic formulas of the form A *- W, where A is an atom and W is a typed first order formula. A typed first order formula can be used as a query, and a closed typed first order formula can be used as an integrity constraint. Functions are allowed to appear in formulas. Such a deductive database system can be implemented using a PROLOG system. The main results are the soundness of the query evaluation process, the soundness of the implementation of integrity constraints, and a simplification theorem for implementing integrity constraints. A short list of open problems is also presented.

1. INTRODUCTION

In recent years, there has been a growing interest in deductive database systems [4-7,15]. Such systems have first order logic as their theoretical foundation. This approach has several desirable properties. Logic itself has a well-understood semantics. Furthermore, its use as a foundation for database systems means that we can employ logic as a uniform language for data, programs, queries, views, and integrity constraints.

One of the most promising approaches to implementing deductive database systems is to use a PROLOG system as the query evaluator [2,8,10,12,17,18]. This approach requires some restrictions on the kinds of formulas which can be used in the database. However, such deductive databases are

MỘT SỐ VẤN ĐỀ CƠ BẢN VỀ CƠ SỞ DỮ LIỆU SUY DIỄN

Bài báo này trình bày cơ sở lý thuyết của các hệ thống cơ sở dữ liệu suy diễn. Một cơ sở dữ liệu suy diễn bao gồm các công thức logic bậc nhất -đóng - định kiểu có dạng ….., trong đó A là một atom và W là một công thức bậc nhất định kiểu. Công thức bậc nhất định kiểu có thể được dùng như một truy vấn, và công thức bậc nhất - đóng - định kiểu có thể được dùng như một ràng buộc toàn vẹn. Trong công thức có thể có các hàm. Chúng ta có thể thực thi một hệ thống cơ sở dữ liệu suy diễn như thế bằng một hệ thống PROLOG. Các kết quả chính là tính đúng đắn của quá trình đáng giá truy vấn, tính đúng đắn của sự thực thi các ràng buộc toàn vẹn, và một định lý đơn giản hóa để thực thi các ràng buộc toàn vẹn. Chúng tôi cũng đưa ra một danh sách vài bài tập mở.

1.GIỚI THIỆU

Trong những năm gần đây, các nhà nghiên cứu ngày càng quan tâm đến các hệ cơ sở dữ liệu suy diễn [4-7,15]. Những hệ như thế có cơ sở lý thuyết là logic bậc nhất. Cách tiếp cận này có một số tính chất đáng quan tâm. Bản thân logic có một ngữ nghĩa rõ ràng. Hơn nữa, việc sử dụng nó như một cơ sở cho các hệ cơ sở dữ liệu đồng nghĩa với việc chúng ta có thể sử dụng logic dưới dạng một ngôn ngữ thống nhất cho dữ liệu, chương trình, truy vấn, hiển thị và các ràng buộc toàn vẹn.

Một trong những cách tiếp cận có nhiều tiềm năng nhất trong việc thực thi các hệ cơ sở dữ liệu suy diễn là sử dụng các hệ PROLOG như một chương trình đánh giá truy vấn [2,8,10,12,17,18]. Phương pháp này áp đặt một số ràng buộc trên các loại công thức được dùng trong cơ sở dữ liệu. Tuy nhiên, các

substantially more general than relational databases and can still be implemented efficiently.	cơ sở dữ liệu suy diễn như thế tổng quát hơn đáng kể các cơ sở dữ liệu quan hệ và vẫn có thể được thực thi có hiệu quả.
This paper contains some basic theoretical results for such an approach to deductive database systems. In particular, it builds on earlier work in [10], which contains special cases of some of the results presented here. In [10], to simplify matters, we assumed that there were no functions in databases, integrity constraints, or queries. In this paper that restriction is removed. It turns out that the proof of a key lemma (Lemma 1 below) is considerably more difficult when functions are allowed.	Bài báo này trình bày một số kết quả lý thuyết về phương pháp tiếp cận các hệ cở sở dữ liệu suy diễn như thế. Đặc biệt, nó xây dựng trên công trình trước đây trong [10], công trình chứa các trường hợp đặc biệt của những kết quả được trình bày ở đây. Trong [10], để đơn giản hóa vấn đề, chúng tôi giả sử rằng không có các hàm trong cơ sở dữ liệu, các ràng buộc toàn vẹn hoặc các truy vấn. Trong bài báo này, những hạn chế đó được loại bỏ. Chúng ta sẽ thấy rằng việc chứng minh bổ đề chính (Bổ đề 1 bên dưới) khó hơn đáng kể khi có các hàm.
The major results of this paper are the soundness of query evaluation and the soundness of the implementation of integrity constraints. These results give a firm theoretical foundation in a general setting for the approach of implementing deductive database systems using PROLOG. Also presented is a simplification theorem for implementing integrity constraints which extends a similar result for relational databases given in [13]. In Section 2, we introduce the main concepts used in these results. In Section 3, the soundness of the query evaluation process is proved. In Section 4, we prove that the implementation of integrity constraints is sound and we prove the simplification theorem. The last section contains some open problems.	Các kết quả chính của bài báo này là tính đúng đắn của đánh giá truy vấn và tính đúng đắn của việc thực thi các ràng buộc toàn vẹn. Những kết quả này cho chúng ta một nền tảng lý thuyết vững chắc trong trường hợp tổng quát để tiếp cận với việc thực thi các hệ cơ sở dữ liệu suy diễn bằng PROLOG. Chúng tôi cũng trình bày một định lý đơn giản để thực thi các ràng buộc toàn vẹn, định lý này mở rộng kết quả tương tự cho các cơ sở dữ liệu quan hệ được trình bày trong [13]. Trong phần 2, chúng tôi đưa vào những khái niệm chính được sử dụng trong những kết quả này. Trong phần 3, tính đúng đắn của quá trình đánh giá truy vấn được chứng minh. Trong phần 4, chúng tôi chứng minh rằng việc thực thi các ràng buộc toàn vẹn là đúng đắn và chúng tôi chứng minh định lý đơn giản hóa. Trong phần cuối, chúng tôi đưa ra một số bài tập mở.
We assume familiarity with [10] and also the basic theoretical results of logic programming, which can be found in [9], The notation and terminology of this paper is consistent with [9] and [10],	Giả sử chúng ta đã biết [10] và các kết quả lý thuyết cơ bản của lập trình logic, có thể tham khảo trong [9], ký hiệu và thuật ngữ của bài báo này thống nhất với [9] và [10].

2. BASIC CONCEPTS

In this section, we introduce the concepts of a deductive database, query, and integrity constraint. We also give the definition of the completion of a database and a correct answer substitution.

We emphasize that, in contrast to [10], here we allow functions to appear in databases, queries, and integrity constraints. The introduction of functions does cause certain problems (see [14] for a discussion), and hence they are commonly excluded in the database context. The major reason for excluding functions is that they can cause the set of answers to a query to be infinite and hence affect the ability of the system to return all answers. However, as we show, having functions does not affect soundness in any way and, after all, soundness is the prime theoretical requirement of any database system. In any case, at this stage, it is important to push the theoretical developments as far as possible.

Underlying all the theoretical developments is a typed first order language. We assume that the language contains only finitely many constants, functions, and predicates.

Each predicate, function, constant, and variable is typed. Predicates have type denoted rxX ••• XV and functions have type denoted Ti X • • • x T,, -»r.

If / has type rx X • • • Xt,,-*t, we say / has range type r.

Terms in the language have a type induced in the obvious way. We assume that, for each type r, there is a ground term of type r.

We use the notation 'ix/rW and 3x/rW to indicate that the bound variable x of the quantifier is of type r. V(f) denotes the typed universal closure of the formula F.

We also use V to denote the ordinary type-free universal closure. It will always be clear from the context which is meant. The concepts of interpretation, model, logical consequence, and so on, are defined in the natural way for typed first order logic (also called many-sorted first order logic). Background material on types is contained in [3].
The reason for using a typed language is evident. Types provide a natural way of expressing the domain concept of relational databases. The requirement that formulas be correctly typed ensures that important kinds of integrity constraints are maintained.

Next we turn to the definitions of the main concepts. For examples of these concepts, see [10].

Definition. A database clause is a typed first order formula of the form A*- W
where A is an atom and W is a typed first order formula. A is called the head and W the body of the clause. The formula W may be absent. Any variables in A and any free variables in W are assumed to be universally quantified at the front of the clause.

Definition. A database is a finite set of database clauses.
Definition. A query is a typed first order formula of the form «- W

where W is a typed first order formula and any free variables of W are assumed to be universally quantified at the front of the query.

Definition. Let «- W be a query, where W has free variables xv..., x,,. An answer substitution is a substitution for some or all of the variables xlt..., x,,.
It is understood that substitutions are correctly typed in that each variable is bound to a term of the same type as the variable.

As in [10], our soundness results require the introduction of the completion of a database.

The definition of the completion given here is a generalization of the definition given in [10]. This generalization is needed because we are now allowing functions to appear in formulas. The definition of the completion requires the introduction of a typed equality predicate = T, for each type t.

These predicates are assumed not to appear in the original language. In particular, no database, query or integrity constraint contains any = T.
Definition. Let D be a database and p a predicate occurring in D. Suppose the predicate p has definition
where each Ai has the form p(tv..., t,,). Then the completed definition of p is the formula

where xly...,x,, are variables not appearing in any Ai«- Wn each Ei has the form

and yv..., yd are the variables of Ai *- Wi which are universally quantified at the front of the clause.

Definition. Let D be a database and p a predicate occurring in D. Suppose there is no clause in D with predicate p in its head. Then the completed definition of p is the formula

The equality theory for a database consists of all axioms of the following form:

(1) c =* T d, where c and d are distinct constants of type r.

(2) V(f(x1,...,x,,)* Tg(yv...,ym)), where / and g are distinct functions of range type r.

(3) V(/(X},..., x,,) ± T c), where c is a constant of type t and / is a function of range type t.

(4) V^x]^ Tx), where i[x] is a term of type t containing x and different from x.

(5) JOV ••• V(x,,*Tmya)^f(x1,...,xn)*rf(y1,...,yn)), where/is a function of type TX X • • ■

(6) Vx/r(x=Tx).

(7) A(x,, = rmyl,)^f(x1,...,xn) = Tf(y1,...,y,,)),

where/ is a function of type tx X • • • X t,, -* t.

(8) V((X!=^^)A A (JC,, = T>

where ^ (including every =T) is a predicate of type Tj X • ■ • XT,,.

(9) Vjc/t((Jc = Tfli) V • • • V (jc = Tak) V (Bjcj/tj • • ■ 3x,,/t,,(x = ...,x,,)))
V • • • V(a^Ai • • • = rfriJ'i,• • ■. Jm))))>
where alt...,ak are all the constants of type t and fx,...,fr are all the functions of range type t.

Axioms 1 to 8 are the typed versions of the usual equality axioms for a program [9], The axioms 9 are the domain closure axioms. This

equality theory generalizes the equality theory given in [10] for the function-free case.

Definition. Let D be a database. The completion of D, denoted comp(D), is the collection of completed definitions for each predicate in D together with the above equality theory.
Definition. Let D be a database and Q a query <- W.

A correct answer substitution for comp(D) U {£?} is an answer substitution 0 such that V(W0) is a logical consequence of comp(D).

The concept of a correct answer substitution gives a declarative understanding of the desired output from a query to a database. In the next section, we prove the soundness of an implementation of this concept.

Next we turn to integrity constraints.

Definition. An integrity constraint is a closed typed first order formula.

Intuitively, an integrity constraint should be an invariant of the database. This leads us to make the following definition.

Definition [15]. Let D be a database such that comp(Z)) is consistent, and let W be an integrity constraint. We say D satisfies W if W is a logical consequence of comp(D); otherwise, we say D violates W.
Finally we define a class of databases that has several important properties.

Definition. A database is called hierarchical if its predicates can be partitioned into levels so

that the definitions of level 0 predicates consist solely of database clauses A «- and the bodies of the clauses in the definitions of level j predicates (j > 0) contain only level i predicates, where i < j.

Such a database is more general than a relational database, but does not allow recursively defined predicates. Related definitions are given in [1] and [16].

3. QUERIES

In this section, we shall prove that our query evaluation process is sound. To prove this result, we only have to prove a generalization of Lemma 4 of [10] for which functions are allowed. The remainder of the argument given in [10] is valid in this more general context. The generalization of Lemma 4 of [10] which we require is given by Lemma 1 below.

The precise details of the query evaluation process are given in [10]. Fortunately, most of the details are not needed to understand Lemma 1. Thus we only present here an overview of query evaluation. The first step of the query evaluation process transforms typed first order formulas into corresponding type-free first order formulas. For this, we use a standard transformation [3],

Definition. Let W be a typed first order formula. For each type t, we associate a unary type predicate also denoted by t.

Then the type-free form W* of W is the first order formula obtained from W by applying the following transformations to subformulas of W of the form Vx/tF and 3X/TV:

(a) Replace Vx/rV by Vx(V*- T(X)).

(b) Replace 3X/TV by 3x(V A r(x)).

We will also require the usual type theory [3],

Definition, The type theory O consists of all axioms of the following form:

(1) r(a), where a is a constant of type t.

(2) Vxj • • • Vxn(,r(f{x-i,x,,))«- A • • • A r,,(x,,)),
where / is a function of type X • • • X t,, -* t.

Now we can give an overview of query evaluation. To answer a query Q to a database D, we first transform Q and D to their type-free forms Q* and D*,

where D* = {c* : Ce D).

We then transform Q* and Z)*U$ into an ordinary PROLOG goal G and program P (which generally may include negations) by successively applying some of the 10 transformation rules given in [10], which eliminate universal quantifiers, implications, and so on, in the bodies of clauses. A computed answer to the query Q for the database D is then defined to be a computed answer to the goal G for the program P. Note that, due to the presence of the type predicates, every computed answer is a ground substitution for all free variables in the body of the query. As we explained in [10], to ensure that the negations are handled properly, it is essential that the PROLOG system use a safe computation rule (that is, one which only selects negative literals that are ground). If R is a safe computation rule, then an R-computed answer substitution for

DU (0) is an /^-computed answer substitution for P U {G}.

Since we are allowing functions, a query can have infinitely many answers. However, under a reasonable restriction on the type theory <!>, we can ensure that each query can have at most finitely many answers. As with databases, we say that <E> is hierarchical if there are some types whose type axioms are only of the form (1) above (that is, these types do not have any function of that range type), there are some further types whose axioms of the form (2) above can only refer to the first set of types in their bodies, and so on. In particular, this restriction bans recursion in 0.

For such a type theory, it is clear that there are only finitely many ground terms of each type. Consequently, each query can have at most finitely many answers. We emphasize that it is not so much the presence of functions which causes queries to have infinitely many answers, but rather the presence of a "recursive" type theory.

With this background, we now proceed with the proof of Lemma 1. The lemma is a technical one which is only concerned with the first step of query evaluation, where we transform typed formulas into type-free ones. In this lemma, D* U $ is essentially a type-free database (called an extended program in [10]), and its completion, comp(D* u <i>), is essentially a type-free version of the completion of a database given above, without the domain closure axioms. We refer

the reader to [10] for the precise definitions.

Lemma 1. Let D be a database and W a closed typed first order formula. Let D* and W* be the type-free forms of D and W. If W* is a logical consequence of comp(D* U $), then W is a logical consequence of comp(Z)).

PROOF. The proof is rather long and requires some preparation. Given a model M for comp(D), we have to construct a model M* for comp(D* U$). The complexity of the construction of M* which we use is needed to ensure that the equality axioms are satisfied.

Let M be a model for comp(D). Using (the typed version of) [11, p. 83], we can assume without loss of generality that M is normal, that is, each = T is assigned the identity relation on the domain CT of type r. We can also suppose the CT's are disjoint. Put C = U TCT.

The underlying language L* for the interpretation M* includes all the constants, functions, and (nonequality) predicates of the underlying language L for M. L* differs from L in that all type information is suppressed, the various typed equality predicates = T are replaced by a single equality predicate = , and there is a unary predicate r for each type r.

Let F' be the set of mappings on the CT assigned by M to the functions in L. Let T be the set of all (free) terms that can be formed using elements of C as primitive terms and elements of F' as functions. (Note that the type restrictions are ignored in forming these terms).

The domain of M* will be the set of equivalence classes of a particular equivalence relation A on 7.

To define A, we introduce a reduction operation on T7.

We write f'(dx,dn) —> d if / has type tx x ■ • ■ XT,, -» t, /' is the mapping assigned to / by M, dt e CT., ¿eCT, and f\dx,..., d,,) = d.

For s, t^T, we write s => t if t is the result of replacing some (not necessarily proper) subterm f'(dv..., d,,) of s by d,

where f(dx,..., dn) —> d. We say that seT is irreducible if there is no t e 7 such that s => t.

Finally, for s, t e 7, we say that s reduces to t if there exists r0, r1,...,r,,e7' such that s = r0 => rx =» • • • =* rn = t.

Now we can define the equivalence relation A on 7. Let 5, t e T.

Then sAr if there exists u e 7 such that s reduces to u and t reduces to u. To prove that A is an equivalence relation, we use the following lemma.

Lemma 2. Let s&T. Then there exists a unique irreducible t e T such that s reduces to t. (We say that t is the irreducible form of s.)

PROOF OF LEMMA 2. That there exists an irreducible form of each s e 7 is immediate, since in each reduction u => v, v has fewer subterms than u.

To prove that irreducible forms are unique, first note that if /'(Ji, • • •,■?,,) reduces to g'(*i»..., tm), then f = g', and that the last step in any reduction of f'(s1, to an element d^C hence has the form f'(d1,..., dn)^> d.

Structural induction can then be used to show that the assumption that 5 has two distinct irreducible forms leads to a contradiction. □

Lemma 3. A is an equivalence relation.

PROOF OF LEMMA 3. Clearly, A is reflexive and symmetric. That A is transitive follows immediately from Lemma 2. □

We now define the domain of the model M* to be 7/A, the set of A-equivalence classes in 7. If t e 7, we let [f] denote the A-equivalence class containing t. Note that 7/A contains a copy of C via the injective mapping d —> [d]. Thus, in essence, we have simply enlarged C in a particular way to obtain a domain for M*.

If c is a constant in L* and M assigns c'eC to c, then M* assigns [c'] in 7/A to c. Let /gL* be an w-ary function. Suppose M assigns the mapping /' to /.

Then M* assigns the mapping from (7/A)" into 7/A defined by ([ij,...,[i,,])-» to /. It is easy to see that this mapping is well defined. Note that this mapping is an extension of /'.

Suppose p is an n-ary predicate in L*. If M assigns the relation p' to p, then M* assigns the relation {([d1],...,[dn]):(d1,...,dn)^p'} on (7/A)" to p.
To a type predicate t, M* assigns the unary relation {[d]: de Cr). In essence, M* assigns CT to r. Finally, M* assigns the identity relation on 7/A to = .

This completes the definition of the interpretation M* for comp(D* U 4>). We now check that M* is a model for comp(Z)* U $). Much of the verification is routine, and we take the liberty of omitting some details.

We first check that M* is a model for the equality theory of comp(D* U $). The eight axioms of the equality theory are given in [10] or [9, p. 70]. Apart from axiom(4), these axioms are easily seen to be satisfied. Axiom (4) is

V(t[x] # x), where t[x] is a term containing x and different from x.

That this axiom is satisfied follows immediately from the next lemma.

Lemma 4. Let r, s e T. If r is a proper subterm of s, then rQis.

PROOF OF LEMMA 4. Suppose r A s. Then there exists an irreducible (e i such that r reduces to t and s reduces to t. Let i/eTbe the result of replacing the occurrence of r in s by t. Then t is a proper subterm of u, and u reduces to t. If t e C, then we obtain a contradiction using axiom (4) of the equality theory for D. Otherwise, t has the form in which case we again have a contradiction, since it isimpossible for u to reduce to t. □

The remainder of the verification that M* is a model for comp(Z>* U$) depends on another lemma. For this we need a definition. A variable assignment V wrt M is an assignment to each variable x in L of an element d e CT, where t is the type of x. Corresponding to V, there is a variable assignment V* wrt M* which assigns [d] to x.

Lemma 5. Let W be a (not necessarily closed) typed first order formula, V a variable assignment wrt M, and V* the corresponding variable assignment wrt M*. Then W is true wrt M

and V iff W* is true wrt M*
and V*.

This lemma is a variant of a well-known result (Lemma 43A in [3]). The proof is a straightforward induction argument on the structure of W.

Using Lemma 5, it can now be checked that M* is a model for the remainder of comp(£)*

U $). The domain closure axioms for D are used to show that M* is a model for the only-if halves of the completed definitions of the type predicates.

We have now finally shown that M* is a model for comp(D* U 0). Since W* is a logical consequence of comp(Z)* UO), we have that M* is a model for W*. Using Lemma 5 again, we obtain that M is a model for W. Thus W is a logical consequence of comp(D). This completes the proof of Lemma 1. □

We can use Lemma 1 in place of Lemma 4 of [10] to obtain the following theorem, which is a generalization of Theorem 3 of [10].

Theorem 1. Let D be a database, Q a query, and R a safe computation rule. Then every R-computed answer substitution for DU {Q} is a correct answer substitution for comp(D)U {Q}.

Theorem 1 is the fundamental result which guarantees the soundness of our query evaluation process. The proof of this theorem (which includes Lemma 1 and several lemmas and theorems in [10]) is indeed long and complicated. However, it would be a mistake to conclude that the implementation of our query evaluation process is correspondingly complicated. In fact, the opposite is the case. The main part of the implementation concerns the 10 transformations given in [10]. These can be implemented in a PROLOG program which contains one clause for each transformation plus a short procedure for locating free

variables. Also, it is easy to avoid the explicit introduction of new predicates which was formally required in [10]. A direct implementation of types would also be easy. However, such an implementation would be inefficient, and hence some optimizations would be required.

4. INTEGRITY CONSTRAINTS

In this section, we prove that our implementation of integrity constraints is sound. We also prove that our simplification method for implementing integrity constraints is sound.

The standard method for determining whether a database satisfies or violates an integrity constraint W is by evaluating the query «- W. The idea is as follows. We evaluate the query *- W. If this query succeeds (that is, if we obtain an SLDNF-re- futation), then Theorem 2 below shows that D satisfies W.

Similarly, if the query fails finitely (that is, if we obtain a finitely failed SLDNF-tree), then Theorem 2 shows that D violates W. For the precise definitions of these concepts, we refer the reader to [10].

Theorem 2 below generalizes Theorems 4 and 5 of [10]. The proof is exactly as in [10], except the Lemma 1 above is used instead of Lemma 4 of [10]

Theorem 2. Let D be a database, W an integrity constraint, and R a safe computation rule. Suppose comp(D) is consistent.

(a) If there exists an SLDNF-refutation of D U {<- W} via R, then D satisfies W.

(b) If D U {«- W} has a finitely failed SLDNF-tree via R, then D violates W.

Next we turn to the simplification method for implementing integrity constraints. Let D be a database. Suppose a user requests that some fact A be deleted from D. Since D is a deductive database, A may not be explicitly present in D, but instead be a logical consequence of D. Thus, to perform the user's request, the system may instead delete some other fact (or facts) explicitly present in the database. This will result in A no longer being a logical consequence of D. Intuitively, we expect the deleted fact (or facts) to be "minimal", that is, their deletion should change D as little as possible. In relational database terminology, finding the right fact (or facts) to delete is called the view update problem. For an addition to a deductive database the situation is much simpler, since we can explicitly add the fact.

In fact, we are not directly concerned with this problem here. We assume that, for whatever reason, the system has to either add a clause to a database or delete an (explicitly present) clause from the database. Such an update can cause an integrity constraint to be violated.

The simplification method is concerned with

the problem of checking with the least amount of work that all the integrity constraints are still satisfied. The key idea is to use the fact that an integrity constraint was satisfied before the update was made either to eliminate the integrity constraint from further consideration or to construct simplified versions of it which must then be checked. The intention is that the simplified versions will be easier to check than the original constraint. This idea is well known in the context of relational databases (see [13] and the references therein). We prove that this simplification method is also sound for deductive databases. In this context, matters are greatly complicated by the presence of rules.

To cover the most general situation with a single theorem, we use the concept of a transaction. A transaction is a finite sequence of additions of clauses to a database and deletions of clauses from a database. If D is a database and t is a transaction, then the application of t to D produces a new database D', which is obtained by applying in turn each of the deletions and additions in t.

We assume that, in any transaction, we do not have the addition and deletion of the same clause. As the deletions and additions in a transaction can then be performed in any order, we assume that all the deletions are performed before the additions. With regard to integrity constraint checking, a transaction is indivisible, so we need only check the constraints at the end of the transaction. Note that we can use a single transaction to pass from any database D to any other database D'.

The results which follow all concern databases, which, by definition, are based on a typed language. The proofs of these results use various definitions and results from [9] . In fact, we will actually require the typed versions of these definitions and results. In all cases, the required modifications to what appears in [9] are very simple.

In what follows, any reference to a definition or result in [9] involving a language actually refers to the appropriate typed version.

To obtain the simplification theorem, we have found it necessary to restrict D to be a definite database. A definite database clause is a database clause that has the form A*-A1 A ••• AA,,,
where A1,...,An are atoms. A definite database is a database that consists of definite database clauses only. The reason for this restriction is that the proof depends crucially on the monotonicity of the mapping TD (defined below) associated with D. Note that, by Propositions 5.1 and 14.3 of [9], comp(D) is consistent if D is definite.

Suppose L is the typed language underlying the database D. We make the assumption throughout that, whatever changes D may undergo, L remains fixed. Thus, for example, adding a new clause to D does not introduce new constants into the language. This is effectively the assumption that is made in [13],
Implementing the simplification method involves computing two sets of atoms, computing two sets of substitutions by unifying atoms in the sets with atoms in an

integrity constraint, and evaluating corresponding instances of the integrity constraint. We begin with the definition of the appropriate sets of atoms.

Definition. Let D and D' be definite databases such that D c D'. We define the set atom0 D, inductively as follows:

atom0D D. = {A : A <— Ax A ■ ■ ■ A Am e D' \ Z>},
atom^i,' = {Ad: A <^AX A ■ ■ ■ AAm e D, B e atom”c D,,

6 is the mgu of some At and B },
atomfl Z). = (J atom''fl D,
n> 0

To motivate the above definition, consider the case when we add a fact A to a database D to obtain a database D'. An important task of the simplification method is to capture the difference between a model for comp(D') and a model for comp(D). In the case that D is a relational database, we see that atomfl D, is {A}, which is precisely the difference between a model for comp(D) and a model for comp(Z)'). (In this case the models are essentially unique [15].) For a deductive database, the presence of rules means that the difference between the models could be larger. However, as we shall see, atom^, D, can still be used to capture the difference between the two models.

A preinterpretation of a database D is an

interpretation of D that omits the assignments of relations to predicates [9, p. 71].

Definition. Let / be a preinterpretation of a database D, V a variable assignment wrt J, and A an atom.

Suppose A is p(tx,..., tn), and d1,...,dn are the term assignments of tv..., t,, wrt J and V. We call Aj v = p(dl,..., d,,) the J-instance of A wrt V. Let [A\j= {AJV: F is a variable assignment wrt /}.

We call each element of [A\s a J-instance of A.
We also call each p(d1,..., d,,) a J-instance. Each interpretation based on J can now be identified with a subset of /-instances as in [9, p. 72],

Definition. Let D and D' be definite databases such that DÇ.D' and J a preinterpretation of D. We define instB fl, y = U/4eatomD DI^l]y.

The essential property of inst^ D, j is presented in Lemma 6 and used in Theorem 3 to capture the difference between models of comp(D) and comp(D').

We now define a monotonie mapping Tjl, from the lattice of interpretations based on J to itself as in [9, p. 72],

Definition. Let / be a preinterpretation of a definite database D.

Let I be an interpretation based on J.

Then 7^(7)= (Ajy: A *—A1 A ■ ■ ■ A A,,eD, F is a variable assignment wrt J, and

{(Ar)J V,... ,(An)j y} c I). It is convenient to suppress the J and denote this mapping by TD.
We also define E = UT[= T(*> x)]7. Subsequent use of E ensures that all models considered are normal.

Lemma 6. Let D and D' be definite databases such that D ç D'. Let J be a preinterpretation of D.

(a) Let M' be an interpretation based on J for D' such that M' U E is a model for comp(D'). Then we have M' \ Tg(M') ç instD D, j for every ordinal a.

(b) Let M be an interpretation based on J for D such that MU E is a model for comp(D). Then we have Tg,(M) \M Q instD D, j for every ordinal a.

PROOF, (a): First note that M' is a fixpoint of TD,, by Proposition 14.3 of [9]. Hence ToiM^QM', and so Tg(M') is defined for every ordinal a. The proof is by transfinite induction. We consider the following two cases.

Case 1: a is a limit ordinal. The case a = 0 is trivial. Otherwise, M' \ 7S(M') — M' \ Dp< aTg(M') = \Jp<a[M' \ T^(M')\ c inst^ ^ y, by the induction hypothesis.

Case 2: a is a successor ordinal. The case a = 1 is trivial. Otherwise, note that M' \ Tg(M') = [M' \ Td(M')] U [Td(M') \ Tg(M')].

Suppose that B g Td(M')\ Tg(M'). Then there exists a clause A*-Ax A ••• A A,, in D such that, for some variable assignment V wrt / and for some i, B is the /-instance of A wrt V, B, is the /-instance of Al wrt V, and Bl 6M'\ T£~l(M').
Thus, by the induction hypothesis, B,. e instD D, }. Hence Bi is also a /-instance of some C g atomD D,. By Lemma 15.1(a) of [9], A, and C are unifiable with mgu 0 = { xx/rv..., xm/rm}, say.

Since C g atom D D, and Afi = CO, we have that Ad g atom0 D,, By Lemma 15.1(b) of [9], the variable assignment that maps A t and C to Bi also maps Xj and to the same domain element, for each j.
Hence B is also a /-instance of AO, and so B g instfl Ly j.

(b) : The proof is similar to part (a). □

Let W be a formula in prenex conjunctive normal form. If a negative literal — A appears in some conjunct of the matrix of W, we say that A is a negated atom in W. If a positive literal A appears in some conjunct of the matrix of W, we say that A is an atom in W.

The addition of a clause C to a database D may cause a /-instance of a negated atom in W that is not in a model M of comp(D) to be in a model M' of comp(Z) U {C}), and thus cause W to be false wrt M'. The set 9 in Theorem 3 below describes all the ways in which this may occur, and which instances of W must hence be checked.

A similar comment applies to deletions and the set 'S'. We now state the simplification theorem for integrity constraints.

Theorem 3. Let D and D' be definite databases, and t a transaction whose application to D produces D'. Suppose t consists of a sequence of deletions followed by a sequence of additions and that the application of the sequence of deletions to D produces the intermediate database D".

Let W = Vjc1 • • • Vx^lV' be an integrity constraint in prenex conjunctive normal form. Suppose D satisfies W.
Let 0 = {0:0 is the restriction to x1,...,x„ of an mgu of a negated atom in W and an atom in atom0.- D.} and = {¡p: \p is the restriction to x±,..., xn of an mgu of an atom in W and an atom in atomD» D). Then we have the following properties:

(a) D' satisfies W iff D' satisfies \f(W'4>) for all <f> G 0 U

(b) If D'U {*-V(W'<j>)} has an SLDNF-refutation for all c|) g 0 U t, then D' satisfies W.

(c) If D' U {«- has a finitely failed SLDNF-tree for some <j> G 0 U 'S', then D' violates W.

Proof, (a): Suppose D' satisfies V(W'fy) for all <j> g 0 u 'S'. Let M' be an interpretation for D' based on / such that M' u£ is a model for comp(D').

Since Td„(M') c M' and TD„ is monotonic, by Propositions 5.3 and 14.3 of [9] there exists an ordinal a such that M" U E is a model for comp(Z>"), where M" = Tg„(M'). Similarly, there exists an ordinal /3 such that MU E is a model for comp(D), where M = T%(M"). By

supposition, W is true wrt Mu E. Let V be a variable assignment wrt /. We have to prove that W' is true wrt M' U E and V. If V* is a variable assignment that agrees with V on xl5..., x,,, then we say V* is compatible with V. We consider the following two cases.

Case 1: For every negated atom A in W and for every V* compatible with V, the J-instance p(dx,..., d,,) of A wrt V* is not in M' \ M,

and for every atom B in W and for every V* compatible with V, the J-instance q(ex,... ,em) of B wrt V* is not in M \ M'. Let A be a negated atom in W, and suppose that, for some V* compatible with V, the /-instance p(dl,...,dn) of A wrt V* is not in M. By the condition of case 1, we have that p(dY,..., dn) £ M' \ M. Hence p(dv..., dn) £ M'.

Let B be an atom in W, and suppose that, for some V* compatible with V, the /-instance q(e1,...,em) of B wrt V* is in M. By the condition of case 1, we have that q(ev..., em) iM\ AT. Hence q(ex,..., em) G M'. It follows from this that W' is true wrt M' U E and V.