A. Voronkov (Ed.)

Logic Programming

First Russian Conference on Logic Programming
Irkutsk, Russia, September 14-18, 1990
Second Russian Conference on Logic Programming
St. Petersburg, Russia, September 11-16, 1991
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Contents

Integrity Verification in Knowledge Bases

François Bry[1], Rainer Manthey[1], and Bern Martens[2]

[1]: *ECRC, Arabellastraße 17, D - 8000 München 81, West-Germany*
[2]: *Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B - 3030 Heverlee, Belgium*

fb@ecrc.de *rainer@ecrc.de* *bern@cs.kuleuven.ac.be*

ABSTRACT *In order to faithfully describe real-life applications, knowledge bases have to manage general integrity constraints. In this article, we analyse methods for an efficient verification of integrity constraints in updated knowledge bases. These methods rely on the satisfaction of the integrity constraints before the update for simplifying their evaluation in the updated knowledge base. During the last few years, an increasing amount of publications has been devoted to various aspects of this problem. Since they use distinct formalisms and different terminologies, they are difficult to compare. Moreover, it is often complex to recognize commonalities and to find out whether techniques described in different articles are in principle different. A first part of this report aims at giving a comprehensive state-of-the-art in integrity verification. It describes integrity constraint verification techniques in a common formalism. A second part of this report is devoted to comparing several proposals. The differences and similarities between various methods are investigated.*

1 Introduction

Knowledge bases extend databases by managing general, nonfactual information. Nonfactual knowledge is represented by means of deduction rules and integrity constraints [GMN84, Rei84, Ull88]. Deduction rules – called, in some cases, views – permit to derive definite knowledge from stored data. Typical examples of deduction rules define flight connections in terms of direct flights, and subordination of employees to managers from membership to and leadership of departments. In contrast, integrity constraints can be used for expressing indefinite and negative information. An integrity constraint can for example claim the existence of a manager in each department without choosing a definite person. Another constraint may require a direct flight on every day from Munich to Paris-Charles-de-Gaule or to Paris-Orly, without imposing the choice of one of the two airports. An integrity constraint can also forbid that someone manages two distinct departments. Restricted kinds of integrity constraints, called dependencies, have been often considered expressive enough for database applications [FV83]. However, as dependencies can neither express complex relationships, nor existence and disjunctive properties, more general integrity constraints are often needed for knowledge base applications.

Some integrity constraints express properties that must be fullfilled in each of the states resulting from updating a knowledge base. They are called 'state' or 'static constraints'. Other constraints refer to both the current and the updated knowledge base. They are called 'transition' or 'dynamic constraints'. Assuming that the current state satisfies the static constraints, those can be viewed as post-conditions for updates, i.e., as a simple form of dynamic constraints. As noted in [Ull88], static integrity constraints can be expressed using the knowledge base query language as yes/no queries, i.e., formally, as closed formulas [GMN84, Rei84]. For expressing dynamic constraints, however, a slight extension of the query language is needed: Statements referring to the current state must be distinguished from statements related to the updated knowledge base.

If a knowledge base is compelled to satisfy static integrity constraints, these constraints have to be verified after each update. This can be achieved by handling the static integrity constraints as post-conditions and simply re-evaluating them in the new state. This naive approach is however often inefficient, for most updates affect only part of the data and therefore only part of the static integrity constraints. Methods have been proposed that profit from the satisfaction of the static integrity constraints in the state prior to the update for simplifying their evaluation in the updated knowledge base. Consider for example a static integrity constraint requiring that all employees speak English. If it is satisfied in the current state, it is sufficient to check that every new employee speaks English for ensuring the desired property in all states. With more general integrity constraints, the definition and the computation of such an update-dependent necessary and sufficient condition is often less trivial. This article is devoted to analyzing methods that systematically generate such conditions from updates and general static integrity constraints.

These methods are presented in the literature from various points of view. In particular, some authors describe the simplification of a static constraint with respect to an update as the specialization of a logic programm – the static constraint – with respect to some input data – the update. The same problem has also been regarded as the generation of a dynamic constraint from a static one. Indeed, simplified forms of static constraints on the one hand impose conditions on the updated state, and on the other hand refer, through the update definition, to the knowledge base state prior to the update.

Processing static integrity constraints has been investigated since declarative query languages are considered, i.e., roughly since relational databases are studied. Early work on integrity verification has investigated whether simple updates could affect some static integrity constraints. The specialization of static integrity constraints with respect to updates has been proposed first by Nicolas in [Nic79] and by Blaustein in [Bla81]. These two seminal papers considered relational databases, i.e., knowledge bases without deduction rules. The necessary extensions needed in the presence of deduction rules have been investigated by several authors from various points of view during the mid-eighties. Since then, a considerable number of articles on integrity verification has been published. Over the last three years, more than fifteen approaches have been described. Since different terminologies and formalisms are used, it is rather difficult to compare the various proposals and to recognize whether methods from different authors significantly differ. This article aims at giving a comprehensive state-of-the-art in integrity verification.

With the aim of overcoming differences in form not reflecting differences of principles, we first describe the techniques involved in integrity constraint verification in a common formalism. This formalization gives rise to recognizing strong similarities between two aspects of integrity verification – the specialization of constraints and the computation of resulting changes. We show that, in fact, computing specialized constraints is a form of partial computation of resulting changes. Then, we consider several proposals. Relying on the formalization we introduced, we analyse their commonalities and differences.

This article consists of five sections, the first of which is this introduction. In Section 2, we present background notions. We propose a formalization of deduction rules, updates, and integrity constraints. Section 3 is devoted to describing the principles of integrity verification. We first consider specializing constraints with respect to updates, then the propagation of updates through deduction rules. Finally, we give a unified view of these two issues. Section 4 is an overview of the literature on integrity verification. We summarize the article and we indicate directions for further research in Section 5.

2 Knowledge Bases: A Formalization

Various languages can be adopted for expressing knowledge, but stylistic differences do not necessarily reflect differences of principles. A same deduction rule for example can be expressed as an SQL view or à la PROLOG – even recursive rules are not forbidden by the definition of SQL [Dat85], although they are rejected by the current implementations. The choice of either style does not impose any evaluation paradigm, though most current systems associate with each style given evaluation techniques. In this section, we introduce a formalism for describing the components of knowledge bases that are relevant to integrity verification: Deduction rules, updates, and integrity constraints. We emphazise that the formalism described here is not proposed as a language for knowledge base designers or users: Styles closer to natural or programming languages could be more convenient for users that are not familiar with mathematical notations.

We think that the formalism proposed below permits to describe large classes of systems. In particular, it can be used for formalizing the conventional update languages of databases as well as updating facilities more recently considered in logic programming, deductive databases, and knowledge bases.

We rely on the nowadays classical formalization of relational and deductive databases in logic [GMN84, Rei84, Ull88]. A knowledge base consists of finitely many relations. A relation is a set of tuples. Relations are not necessarily in first normal form, i.e., tuples may contain nested terms. Relation names are interpreted in logic as predicate symbols, and term constructors occurring in tuples as function symbols. Thus, a knowledge base KB defines a finite first-order logic language $\mathcal{L}(\text{KB})$. The variables of this language range over the components of tuples: This is a domain calculus. Though no more expressive than tuple calculi, domain calculi are more convenient for a simple description of integrity constraint specialization and computation of resulting changes. However, a tuple calculus may be more convenient in other contexts. We recall that information which is not provable is considered proven false *(closed world assumption).*

It is often convenient to strongly restrict the syntax of deduction rules, in order to focus on the core of some issue. In particular, many studies on query evaluation consider DATALOG [Ull88] or DATALOG$^{\text{NOT}}$ [AV88] rules, for investigating respectively the evaluation of recursive rules, or the processing of negation. Since integrity constraints are in general expressed in a more permissive syntax, it is sensitive to allow, in a study on integrity verification, a free syntax for deduction rules as well. We shall therefore not explicitly mention syntactical restrictions imposed by a query evaluator such as, for example, the exclusion of recursive predicate definitions in deduction rules, of explicit universal quantifications, or of function symbols. We assume to have a query evaluator at our disposal and we assume that deduction rules and integrity constraints fulfill the syntactical restrictions that are necessary for the soundness, completeness, and exhaustivity of the considered evaluator.

Integrity verification however does require a syntactical restriction, namely, that deduction rules and integrity constraints are 'range-restricted'. We first recall the definition of deduction rules and static integrity constraints. Then, we define ranges and range-restriction. We introduce a formalism for expressing general updates in terms of deduction rules. Finally, we give a definition of dynamic constraints.

2.1 Deduction Rules and Static Integrity Constraints

A deduction rule is an implicative formula which defines - part or all - of a relation.

> *Definition 1:*
> A rule defining an n-ary relation p is denoted by $p(t_1, ..., t_n) \leftarrow Q$ where Q is a well-formed formula [Smu88] - possibly containing quantifiers -, and where the t_is are terms. The atom $p(t_1, ..., t_n)$ is called the ***head*** of the rule, the formula Q its ***body***.

Formally, the rule $p(t_1, ..., t_n) \leftarrow Q$ denotes the formula $\forall x_1 ... \forall x_k\ [Q \Rightarrow p(t_1, ..., t_n)]$ where $x_1, ..., x_k$ are the variables occurring freely in $p(t_1, ..., t_n)$ or Q.

Intuitively, the evaluation of Q over the considered knowledge base KB returns values for the x_js. These instantiations affect those terms t_i in which some x_js occur and define p-tuples. Q can be viewed as a query with target list $(x_1, ..., x_k)$. More formally, if σ is an instantiation of the x_js such that $Q\sigma$ holds in KB - noted, $KB \vdash Q\sigma$ - then the instantiated term $p(t_1, ..., t_n)\sigma$ belongs to the extension of p - $KB \vdash p(t_1, ..., t_n)\sigma$.

> *Definition 2:*
> A static integrity constraint is a closed well-formed formula [Smu88] in the language of the knowledge base, i.e., a formula the variables of which are all quantified.

It is worth noting that some knowledge can be represented either by means of deduction rules, or as integrity constraints. This is the case, for example, for the statement:

$$\forall x\ [\ employee(x) \Rightarrow speaks(x, English)\ ]$$

Other statements, however, can only be used as integrity constraints, like, for example:

$$\forall x\ [\ employee(x) \Rightarrow (speaks(x, English) \vee speaks(x, German))\ ]$$
$$\forall x\ [\ project(x) \Rightarrow (\exists y\ employee(y) \wedge leads(y, x))\ ]$$

More generally, in the current state-of-the-art in query evaluation, deduction rules cannot serve for defining existential and disjunctive statements. Such properties must be expressed by means of integrity constraints.

For the sake of simplicity, we shall restrict ourselves to the connectives $\neg$, $\wedge$, $\vee$. We assume that $\Rightarrow$, $\Leftarrow$, and $\Leftrightarrow$ are expressed through $\neg$, $\wedge$, and $\vee$ according to the well-known equivalences: $(F \Rightarrow G) \Leftrightarrow (\neg F \vee G)$, $(F \Leftarrow G) \Leftrightarrow (F \vee \neg G)$, and $(F \Leftrightarrow G) \Leftrightarrow [(\neg F \vee G) \wedge (F \vee \neg G)]$. For readibility reasons, however, we shall permit an explicit use of $\Rightarrow$ in combination with universal quantifiers. Consider for example the following universal integrity constraint:

$$\forall x\ [\ \text{employee}(x) \Rightarrow \text{speaks}(x, \text{English})\ ]$$

Its meaning is rather clear: It requires to check that every employee speaks English. This meaning is somehow hidden in the equivalent form:

$$\forall x\ [\ \neg\ \text{employee}(x) \lor \text{speaks}(x, \text{English})\ ]$$

2.2 Range-Restriction

In order to ensure that deduction rules define ground tuples – i.e., tuples without unbound variables – the bodies of rules have to provide ranges for the free variables. Before formally defining ranges, we illustrate this notion with examples. The variables x and y in the rule $p(x, y) \leftarrow q(x) \lor r(y)$ are not both bound by an evaluation of the body $q(x) \lor r(y)$: This body is not a range for x and y. In contrast, evaluating the body of the rule $s(x, y) \leftarrow t(x) \land u(y) \land (q(x) \lor r(y))$ binds all variables. The following definition considers the various possible combinations of logical connectives that yield expressions – called 'ranges' – whose evaluations bind a set of variables.

Definition 3:
A *range* R for variables $x_1, ..., x_n$ is inductively defined as follows:

1. $p(t_1, ..., t_k)$ is a range for $x_1, ..., x_n$ if p is a k-ary predicate symbol and if the t_is are terms such that each variable x_j occurs in at least one of them.
2. $R_1 \land R_2$ is a range for $x_1, ..., x_n$ if R_1 is a range for $y_1, ..., y_k$, if R_2 is a range for $z_1, ..., z_h$ and if $\{y_1, ..., y_k\} \cup \{z_1, ..., z_h\} = \{x_1, ..., x_n\}$.
3. $R_1 \lor R_2$ is a range for $x_1, ..., x_n$ if R_1 and R_2 are both ranges for $x_1, ..., x_n$.
4. $R \land F$ is a range for $x_1, ..., x_n$ if R is a range for $x_1, ..., x_n$ and F is a formula with free variables in $\{x_1, ..., x_n\}$.
5. $\exists y_1...y_p\ R$ is a range for $x_1, ..., x_n$ if R is a range for $y_1, ..., y_p, x_1, ..., x_n$.

Integrity verification does not only require that the body of a deduction rule is a range for its free variables. Similar conditions, formalized by the following definition, must as well be imposed on the quantified variables that occur in bodies of deduction rules and in static integrity constraints.

These conditions are very similar to typed quantifications. Typed quantifications assign types to each quantified variable. If types are represented by unary predicates t_i, a typed quantification has one of the forms $\exists x\ t_i(x) \land F$ or $\forall x\ t_i(x) \Rightarrow F$. The definition of range-restricted formulas is based on the same idea, but allows to 'type' variables with ranges. For legibility reasons, we denote multiple quantifications $\exists x_1...\exists x_n$ and $\forall x_1...\forall x_n$ by $\exists x_1...x_n$ and $\forall x_1...x_n$, respectively.

Definition 4:
A closed formula F is *range-restricted* if all the quantified subformulas SF of F have one of the following forms:

- $\exists x_1...x_n\ R$
- $\exists x_1...x_n\ R \land G$
- $\forall x_1...x_n\ \neg R$

- $\forall x_1 \ldots x_n\ R \Rightarrow G$

where R is a range for $x_1, \ldots, x_n$ and G is a formula with all free variables in $\{x_1, \ldots, x_n\}$.

A deduction rule $p(t_1, \ldots, t_n) \leftarrow Q$ is *range-restricted* if the closed formula $\forall x_1 \ldots x_k [Q \Rightarrow p(t_1, \ldots, t_n)]$ is range-restricted, where $x_1, \ldots, x_k$ are the free variables of $[Q \Rightarrow p(t_1, \ldots, t_n)]$.

An open formula F with free variables $x_1, \ldots, x_k$ is *range-restricted* if its existential closure $\exists x_1 \ldots x_k\ F$ is range-restricted.

The notion 'range-restriction' has been defined first for formulas in clausal form in [Nic79]. The more general definition given above, as well as Definition 3, was given in [Bry89b]

Range-restricted formulas are definite [Kuh67] and domain independent [Fag80]. These two equivalent properties characterize the formulas whose evaluation is not affected by updates that have no effects on the extensions of the relations mentioned in the formulas. Although some domain independent formulas are not range-restricted, it is necessary in practice to consider that class of formulas since definiteness and domain independence are undecidable [Di 69]. Informally, the proof given in [Di 69] makes use of the fact that there is no decision procedure for recognizing tautologies in first-order logic. Tautologies are definite, indeed, for they are not affected by any update since they always evaluate to true.

In contrast with definiteness and domain independence, range-restriction is defined syntactically. It is therefore decidable. It is worth noting that all proposed query languages restrict themselves to range-restricted queries. Range-restricted deduction rules and integrity constraints are assumed in most studies on integrity verification. Although this property is not mentioned in other studies, the methods they describe are correct only if rules and constraints are range restricted. In more theoretical studies, other decidable classes of domain independent formulas have been proposed, e.g., the various classes of allowed formulas [LT86, VGT87, She88], and the formulas a bit misleadingly called 'definite' in [Ull88]. For each closed formula in one of these classes, an equivalent range-restricted formula can be constructed by applying equivalence preserving syntactical transformations [Bry89a].

2.3 Rule-Based Specification of Updates

In this section, we describe a formalism for defining set updates. Many studies on integrity verification mention only single tuple updates – i.e., the insertion or the removal of a given tuple. The reason for this apparent restriction is that the techniques involved in integrity verification can be extended without difficulties from single tuple updates to set updates. We first define set updates. Then, we show that more general updates – such as intensional and sequential updates – can be expressed in terms of deduction rules. Thus, an integrity verification method that takes deduction rules and set updates into account is in fact also applicable to more general updates.

Set updates are necessary since the modifications of a knowledge base that are meaningful from the application viewpoint, often affect more than one relation. If, for example, an employee leaves a company, one may have to remove a first tuple from a relation listing the affectation of employees to projects, and a second tuple from another relation with administrative data (address, salary, etc.). An update may affect deduction rules as well as factual data. Since relations are the usual way to represent sets in knowledge bases, we shall rely on two relations 'to-be-removed' and 'to-be-inserted' for describing set updates. Thus, the insertion of a fact p(a, b, c) and the removal of a deduction rule $q(x, y) \leftarrow r(x, z) \wedge s(z, y)$ will be denoted by the two facts:

to-be-inserted(p(a, b, c))
to-be-removed(q(x, y) $\leftarrow$ r(x, z) $\wedge$ s(z, y))

For the sake of simplicity, and without loss of generality, we assume that modifications are expressed by means of insertions and removals. Thus, changing the address of the employee John from Munich into Paris is expressed as:

to-be-inserted(address(John, Paris))
to-be-removed(address(John, Munich))

We assume a non-sequential, instantaneous semantics for set updates. Given a set update U to a knowledge base KB, the updated knowledge base KB_U is obtained by removing from KB all facts and rules marked in U as 'to-be-removed', and by inserting into KB all expressions 'to-be-inserted'. The following definition formalizes this semantics.

Definition 5:
An update U to a knowledge base KB consists of two (finite) unary relations 'to-be-inserted' and 'to-be-removed' ranging over deduction rules and ground facts in the language $\mathcal{L}$(KB) of KB.

The updated knowledge base KB_U defined by U and KB is defined as

$$KB_U = (KB \setminus \{E \mid \text{to-be-removed}(E) \in U\}) \cup \{E \mid \text{to-be-inserted}(E) \in U\}$$

where a difference A $\setminus$ B of two knowledge bases A and B contains the facts of A that are not in B, and the deduction rules of A that are not variant of rules in B.

This definition calls for some comments. First, the formalism of Definition 5 only permits one to define set updates, not to specify when to perform them. This second issue requires to extend the declarative specification language we have considered above – for facts, deduction rules, integrity constraints, and update definitions – into a procedural, data manipulation language. Such an extension can be done in various ways. One of them is to keep clearly separated the declarative specification and query language from the procedural manipulation language [Dat85, MKW89]. Other approaches [NK87, NT89, dS88a, dS88b] do not really distinguish the two languages.

Second, as they are defined above, updates may contain both 'to-be-inserted(E)' and 'to-be-removed(E)' for a same fact or rule E. Such updates seem inconsistent. However, they are well-defined because Definition 5 gives priority to insertions over removals: In presence of conflicting updating requirements, the insertion prevails. One could question, whether the opposite priority would not be more convenient for some applications. One could even suggest to consider more sophisticated priorities, depending for example on the predicate defined by the fact or the rule. We do not want to argue in favor of any definition: Which one is more convenient is likely to be application dependent. Modifying Definition 5 appropriately does not affect the issues investigated in this paper.

Finally, Definition 5 interprets set updates as non-sequential, instantaneous changes: No semantics are associated with the intermediate states that might be temporarily reached during the computation of KB_U from KB. In other words, set updates are "atomic transactions": As far as integrity verification is concerned, they are indivisible. If U is a set update of a knowledge base KB, the static integrity constraints express conditions on KB and KB_U but on no intermediate state temporarily reached during the computation of KB_U from KB.

In the rest of this section, we propose to formalize more general updates than set updates in terms of deduction rules. We consider first intensional updates, then sequential updates.

One often needs the capability to refer to a set update by specifying the facts and rules 'to-be-inserted' or 'to-be-removed' through a query on the current knowledge base. For example, let us assume that flights are stored in a ternary relation 'flight' the attributes of which are, successively, the departure airport, the arrival airport, and the flight number. The query

$$\text{flight(Munich, y, z)} \land \text{in(y, France)}$$

characterizes the flights from Munich to French airports. Therefore, the removal of these flights is well characterized by the formula:

$$\forall yz \; [\; \text{flight(Munich, y, z)} \land \text{in(y, France)} \Rightarrow \text{to-be-removed(flight(Munich, y, z))} \;]$$

Such a universal formula is, in fact, a deduction rule for the meta-relation 'to-be-removed':

$$\text{to-be-removed(flight(Munich, y, z))} \leftarrow \text{flight(Munich, y, z)} \land \text{in(y, France)}$$

To make such rule-based expression of set updates possible, Definition 5 does not need to be modified, provided that the non-sequential, instantaneous semantics of set updates is not abandoned. This means that the non-sequential semantics of Definition 5 can be kept if deduction rules defining the relations 'to-be-inserted' and 'to-be-removed' are not seen as defining triggers or production rules. The non-sequential semantics of Definition 5 requires that bodies of rules for 'to-be-inserted' and 'to-be-removed' are all evaluated on the same knowledge base state, namely on the state prior to any change.

Other rule-based definitions of updates have been considered, e.g., in [AV87, AV88, Abi88], and in the languages LDL [NK87, NT89] and RDL1 [dS88a, dS88b]. In these studies, insertions and removals are immediately triggered, as soon as a rule body is satisfied. Therefore, bodies of updating rules are evaluated on different states. Such a processing leads to complex semantic issues: In particular, updates may be non-deterministic. In contrast, Definition 5 gives a deterministic semantics, for KB_U is always uniquely defined whatever are U and KB.

The notion of range-restricted rules – Definition 4 – must be extended to deduction rules defining updates. Clearly, it would not make sense to restrict rule-based definitions of updates to insertion and removal of ground deduction rules! Definition 6 formalizes range-restricted update rules.

Definition 6:
A rule to-be-inserted(E) $\leftarrow$ B or to-be-removed(E) $\leftarrow$ B is range-restricted if one of the following conditions is satisfied:

- E is an atom and the rule E $\leftarrow$ B is range-restricted.
- E = (H $\leftarrow$ C) and the rule H $\leftarrow$ (B $\land$ C) is range-restricted.

Intuitively, range-restriction for rule-defined updates of deduction rules ensures that those variables not bound by the evaluation of the rule-update body will be bound during the evaluation of the inserted or removed rule.

It is worth noting the close similarity between intensionally defined updates and updates of deduction rules. An intensional update permits one to explicitly affect all the employees with a certain skill to a certain project. Inserting a new rule allows to implicitly define these employees as affected to the considered project. The effect of both updates is the same as far as the semantics is concerned. However, the intensionally defined update results in the insertion of new facts, while these facts remain implicit by the rule insertion.

Updating transactions are often defined as sequences of changes, the subsequent changes being defined in terms of the knowledge base state resulting from the previous changes. If such a sequential transaction is atomic with respect to integrity, i.e., no intermediate state is compelled to satisfy integrity constraints, it cannot be formalized as a sequence of atomic set update transactions. We conclude this section by showing how deduction rules of a new type can be used to express sequential updates in terms of set updates.

In order to express sequential updates, one needs a formalism for distinguishing between various states of a knowledge base. We propose to annotate queries with states as follows. Given two non-sequential updates U_1 and U_2 and a query Q, the expression 'after(U_1, Q)' will denote the query Q against the knowledge base $KB_1 = KB_{U_1}$. Similarly, 'after(U_1, after(U_2, Q))' represents the query Q on the updated knowledge base obtained from KB by U_1 followed by U_2, i.e., on the knowledge base $KB_{U_1\ U_2}$.

> *Definition 7:*
> Let KB be a knowledge base, U an update of KB, and Q a query in the language $\mathcal{L}$(KB) of KB.
>
> The term 'after(U, Q)' denotes the query Q on the knowledge base KB_U.
>
> A sequential update S on KB is a finite ordered list [U_1, ..., U_n] such that U_1 is a non-sequential update of KB, and each U_i (i =2, ..., n) is a non-sequential update of $KB_{U_1 \ldots U_{i-1}}$.

Since Definition 7 makes use of a meta-predicate, namely 'after', it is a rather natural question whether the semantics of sequential updates is first-order or not. One can first remark that updates are as such not, strictly speaking, expressible in classical first-order logic. This logic is untemporal, indeed, and does not provide one with any means for expressing changes of theories. However, one can observe that a query 'after(U, Q)' has indeed a first-order semantics, for it represents the query Q – a first-order formula – on the knowledge base KB_U – a first-order theory. In other words, the meta-predicate 'after' does not need to be interpreted in second-order logic – see [End72] § 4.4, p. 281-289.

In this section, we have proposed a language for defining sequential and non-sequential update intentions by means of special facts and deduction rules. In Section 3.2, we show how to implement a hypothetical reasoning evaluator by means of deduction rules, in order to simulate query evaluation on an updated knowledge base, without actually updating the knowledge base.

2.4 Dynamic Integrity Constraints

The 'after' meta-predicate introduced in Definition 7 can be used for expressing dynamic integrity constraints. Given a – possibly sequential – update U of a knowledge base KB, let us extend the language $\mathcal{L}$(KB) of KB with 'after(U, .)' expressions. A dynamic integrity constraint of KB is simply a closed formula in this extended language. The following definition formalizes this remark.

Definition 9:
Let KB be a knowledge base and U an update of KB. The set of formulas $\mathcal{F}_U(KB)$ is inductively defined as follows:

- F $\in \mathcal{F}_U(KB)$ if F is a formula in $\mathcal{L}(KB)$
- after(U, F) $\in \mathcal{F}_U(KB)$ if F is a formula in $\mathcal{L}(KB)$
- $\neg$ F $\in \mathcal{F}_U(KB)$ if F $\in \mathcal{F}_U(KB)$
- F θ G $\in \mathcal{F}_U(KB)$ if F $\in \mathcal{F}_U(KB)$, if G $\in \mathcal{F}_U(KB)$, and if θ is a logical connective
- Qx F $\in \mathcal{F}_U(KB)$ if F is a formula in $\mathcal{L}(KB)$, x is a free variable in F, and Q denotes $\forall$ or $\exists$

A dynamic integrity constraint is a closed formula F in $\mathcal{F}_U(KB)$ which is not also a formula in the language $\mathcal{L}(KB)$.

Closed formulas in $\mathcal{L}(KB)$ are not dynamic integrity constraints, for they contain no occurrences of 'after' and therefore do not express relationships between two successive knowledge base states.

One could extend Definition 9 by defining constraints related to more than two knowledge base states. We do not give such a definition here because constraints referring to more than two states are not needed for expressing specialized forms of static integrity constraints with respect to updates.

3 Integrity Verification: Principles

In this section, we first consider specializing static integrity constraints with respect to (possibly sequential) set updates. Then, we show how deduction rules can serve to specify – and implement – a hypothetical reasoning that simulates an updated knowledge base. This reasoning propagates the updates through the deduction rules of the knowledge base. Finally, we give a unified view of the two issues: We show how constraint specialization can be viewed as update propagation.

3.1 Specialization of Static Integrity Constraints

For the sake of clarity, we introduce the specialization techniques stepwise, for static integrity constraints of increasing structural complexity. We consider first ground, quantifier-free static integrity constraints, then constraints with universal and existential quantifiers.

The quantifier-free constraints with simplest syntax are ground literals – i.e., facts or negated facts. Although very restricted, this type of constraint is sometimes useful. A factual static constraint like 'department(financial)' ensures, for example, that a company always has a financial department. Let $\neg$p(a, b) be such a constraint. Clearly, it is violated by all updates containing to-be-inserted(p(a, b)). It is also violated by updates U that do not explicitly require the insertion of p(a, b), but result in the truth of p(a, b), i.e., are such that $KB_U \vdash$ p(a, b). For example, if KB contains the deduction rule p(x, y) $\leftarrow$ q(x) $\wedge$ r(y), U = {to-be-inserted(q(a)), to-be-inserted(r(b))} is such an update. This is a general phenomenon: In order to determine the effect of an update on static integrity constraints, one has to take into account all changes resulting from the update. The following definition formalizes the notion of resulting change.

Definition 10:

Let KB be a knowledge base, U an update of KB, and F a ground fact.

F is an insertion resulting from U if $KB_U \vdash F$ and $KB \not\vdash F$. F is a removal resulting from U if $KB_U \vdash \neg F$ and $KB \not\vdash \neg F$ – i.e., $KB \vdash F$ and $KB_U \not\vdash F$.

In the same way as we defined updates by means of the two relations 'to-be-inserted' and 'to-be-removed', we shall consider two relations 'resulting-insertion' and 'resulting-removal' for denoting the changes resulting from an update. Using the meta-predicate 'after' of Definition 7 and the deduction rule formalism, one can reformulate Definition 10 as follows:

$$\begin{array}{lll} \text{resulting-insertion}(U, F) & \leftarrow & \text{after}(U, F) \wedge \neg F \\ \text{resulting-removal}(U, F) & \leftarrow & F \wedge \text{after}(U, \neg F) \end{array}$$

where F denotes a ground fact in the language $\mathcal{L}(KB)$ of the knowledge base KB under consideration. For negative ground literals ¬F, we have:

$$\begin{array}{lll} \text{resulting-insertion}(U, \neg F) & \leftarrow & \text{resulting-removal}(U, F) \\ \text{resulting-removal}(U, \neg F) & \leftarrow & \text{resulting-insertion}(U, F) \end{array}$$

Using this formalism, a static integrity constraint ¬p(a, b) which is satisfied by a knowledge base KB is violated by an update U if and only if resulting-insertion(U, p(a, b)) is true. Similarly, an update U violates a positive constraint q(b) satisfied in KB if and only if resulting-removal(U, q(b)). These conditions can be expressed as ¬resulting-insertion(U, p(a, b)) and ¬resulting-removal(U, q(b)), or by the following conditional statements:

$$\begin{array}{ll} \text{resulting-insertion}(U, p(a, b)) & \Rightarrow \text{false} \\ \text{resulting-removal}(U, q(b)) & \Rightarrow \text{false} \end{array}$$

These update dependent specialized forms of static integrity constraints are dynamic integrity constraints.

Since a conjunctive integrity constraint like p(a, b) ∧ q(c) can be represented by the two constraints p(a, b), q(c), there are no lessons to be learned from considering such compound expressions. Disjunctions are, however, more interesting. Assume that KB satisfies the disjunctive constraint C = p(a, b) ∨ q(c). C is violated in KB_U if both p(a, b) and q(c) are false in this knowledge base. Therefore, if one of the disjuncts of C is a resulting removal, the other disjunct p(a, b) must be true in KB_U. This is expressed by the following implications:

$$\begin{array}{ll} \text{resulting-removal}(U, p(a, b)) & \Rightarrow \text{after}(U, q(c)) \\ \text{resulting-removal}(U, q(c)) & \Rightarrow \text{after}(U, p(a, b)) \end{array}$$

Before considering quantified constraints, we formalize the specialization of static integrity constraints. The following definition derives dynamic integrity constraints from static integrity constraints.

Definition 11:

Let C be a ground formula. S(C) is the set of formulas obtained from C as follows:
For each atom A occurring in C, S(C) contains the implication

$$\begin{array}{ll} \text{resulting-removal}(U, A) & \Rightarrow \text{after}(U, C_{A/\text{false}}) \text{ if A has positive polarity in C} \\ \text{resulting-insertion}(U, A) & \Rightarrow \text{after}(U, C_{A/\text{true}}) \text{ if A has negative polarity in C} \end{array}$$

where $C_{A/x}$ is the formula obtained from C by replacing A by x.

The following classical implications permit one to simplify the formulas of a set S(C):
$F \vee \text{false} \Rightarrow F$, $F \wedge \text{false} \Rightarrow \text{false}$, $\neg\text{false} \Rightarrow \text{true}$, $F \wedge \text{true} \Rightarrow F$, $F \vee \text{true} \Rightarrow \text{true}$, $\neg\text{true} \Rightarrow \text{false}$, etc.

Consider for example the static integrity constraint: $C = (p(a) \vee \neg q(b)) \wedge \neg(r(c) \vee r(d))$. After simplification the set S(C) consists of:

$$\begin{array}{ll} \text{resulting-removal}(U, p(a)) & \Rightarrow \text{after}(U, \neg q(b) \wedge (\neg r(c) \vee r(d)) \\ \text{resulting-insertion}(U, q(b)) & \Rightarrow \text{after}(U, p(a) \wedge \neg(r(c) \vee r(d)) \,) \\ \text{resulting-insertion}(U, r(c)) & \Rightarrow \text{after}(U, \text{false}) \\ \text{resulting-insertion}(U, r(d)) & \Rightarrow \text{after}(U, \text{false}) \end{array}$$

A further simplification permits one to rewrite 'after(U, false)' as 'false': 'false' is provable in no knowledge base, indeed.

If C is a clause, i.e., if C is a literal or a disjunction of literals, S(C) is obtained by resolving out the literals of C. If L is such a positive literal and if R is the corresponding resolvent, S(C) contains the implication:

$$\text{resulting-removal}(U, L) \Rightarrow \text{after}(U, R)$$

If L is a negative literal $\neg A$, S(C) contains the implication:

$$\text{resulting-insertion}(U, A) \Rightarrow \text{after}(U, R)$$

Definition 11 may be seen as relying on an extension of unit resolution to general, nonclausal ground formulas.

The following Proposition establishes the correctness of the Definition 11.

Proposition 1:
Let KB be a knowledge base, let U be an update of KB, and let C be a static integrity constraint such that $KB \vdash C$.

$$KB_U \vdash C \text{ if and only if } KB \cup \text{after}(U, KB) \vdash F, \text{ for all } F \in S(C)$$

where 'after(U, KB)' denotes the extension $\{\text{after}(U, A) \mid KB_U \vdash A\}$ of the relation 'after'.

[*Proof:* By structural induction on C.]

Consider now the universally quantified static constraint: $C = \forall x\ p(x) \Rightarrow \neg q(x)$. If resulting-insertion(U, p(a)) holds, then the truth of C in KB_U requires that $\neg q(a)$ holds in KB_U. Similarly, if resulting-removal(U, $\neg q(c)$), i.e., resulting-insertion(U, q(c)) holds, then after(U, $\neg p(c)$) must hold as well. This suggests the following update dependent specialized forms for C:

$$\forall x\ [\ \text{resulting-insertion}(U,\ p(x)) \Rightarrow \text{after}(U,\ \neg q(x))\ ]$$
$$\forall x\ [\ \text{resulting-insertion}(U,\ q(x)) \Rightarrow \text{after}(U,\ \neg p(x))\ ]$$

Note that they are in principle obtained like ground static constraints are specialized. Definition 12 formalizes this observation.

Definition 12:

Let C be a universally quantified range-restricted formula $\forall x\ R[x] \Rightarrow F[x]$ such that R[x] is a range for x, F[x] is a formula containing x as free variable, and no existential quantifiers occur in C.

S(C) is the set of formulas obtained from C as follows. For each atom A occurring in C, S(C) contains a formula:

$$\forall x\ \text{resulting-removal}(U,\ A) \land \text{after}(U, R[x]_{A/\text{false}}) \Rightarrow \text{after}(U, F[x]_{A/\text{false}})$$
if A occurs in C with positive polarity
$$\forall x\ \text{resulting-insertion}(U,\ A) \land \text{after}(U, R[x]_{A/\text{true}}) \Rightarrow \text{after}(U, F[x]_{A/\text{true}})$$
if A occurs in C with negative polarity

where an expression $G_{A/x}$ is obtained from a formula G by replacing the atom A by x.

Applying this definition to the constraint $C = \forall x\ p(x) \Rightarrow \neg q(x)$ produces the formulas:

$$\forall x\ [\ \text{resulting-insertion}(U,\ p(x)) \Rightarrow \text{after}(U,\ \neg q(x))\ ]$$
$$\forall x\ [\ \text{resulting-insertion}(U,\ q(x)) \land \text{after}(U,\ p(x)) \Rightarrow \text{false}\ ]$$

Although the second formula is not identical with the one mentioned above, it is equivalent to it.

Proposition 2 establishes the soundness of Definition 12.

Proposition 2:

Let KB be a knowledge base, let U be an update of KB, and let $C = \forall x\ R[x] \Rightarrow F[x]$ be a static integrity constraint such that KB $\vdash$ C.

$$KB_U \vdash C \text{ if and only if } KB \cup \text{after}(U, KB) \vdash F, \text{ for all } F \in S(C)$$

[*Proof:* By structural induction on R and F.]

Consider finally an existential statement $C = \exists x\ p(x)$. Intuitively, any resulting removal of a p fact may affect it, since we do not know for which values C is true in the knowledge base prior to the update. Therefore, the dynamic constraint associated with C is:

$$\forall x\ \text{resulting-removal}(U,\ p(x)) \Rightarrow \text{after}(U,\ \exists x\ p(x))$$

Similarly, if $D = \exists x\ p(x) \land \neg q(x)$ is a static integrity constraint satisfied in KB, the following statements are necessary conditions for the truth of D in the updated knowledge base KB_U:

(1) $\forall x\ \text{resulting-removal}(U,\ p(x)) \Rightarrow \text{after}(U,\ \exists x\ p(x) \land \neg q(x))$
$\forall x\ \text{resulting-insertion}(U,\ q(x)) \Rightarrow \text{after}(U,\ \exists x\ p(x) \land \neg q(x))$

In this case, however, more restrictive premisses can be considered. Assume for example that p(c) is a resulting removal for an update U. If $\neg q(c)$ does not hold in KB, then the removal of p(c) cannot induce a violation of D in KB_U. Therefore, $KB_U \vdash D$ if and only if:

(2) $\forall x$ resulting-removal(U, p(x)) $\wedge \neg q(x) \Rightarrow$ after(U, $\exists x\ p(x) \wedge \neg q(x)$)
$\forall x$ resulting-insertion(U, q(x)) $\wedge\ p(x) \Rightarrow$ after(U, $\exists x\ p(x) \wedge \neg q(x)$)

It is not an easy matter to decide whether (1) or (2) is more efficient to evaluate. The overhead introduced by computing more restrictive premisses may very well be significantly bigger than the gain it brings. It is beyond the scope of this article to propose a strategy or a heuristic for making this choice. Moreover it seems preferable to define such a choice method for general queries, not only for integrity verification.

In order to be capable to express possible choices, without committing ourselves to one of them, we rely on a meta-predicate 'option'. Our example becomes:

$\forall x$ resulting-removal(U, p(x)) $\wedge$ option($\neg q(x)$) $\Rightarrow$ after(U, $\exists x\ p(x) \wedge \neg q(x)$)
$\forall x$ resulting-insertion(U, q(x)) $\wedge$ option(p(x)) $\Rightarrow$ after(U, $\exists x\ p(x) \wedge \neg q(x)$)

Definition 13 formalizes this notation:

Definition 13:
In a theory T, if $F \Rightarrow H$ holds if and only if $F \wedge G \Rightarrow H$ holds, then the expression $F \wedge$ option(G) $\Rightarrow H$ denotes either formula.

We shall use the notation of Definition 13 with a theory T consisting of a knowledge base KB which satisfies its static integrity constraints. Definition 12 can be generalized as follows:

Definition 14:
Let C be a range-restricted closed formula. S(C) is the set of formulas obtained from C as follows. For each atom A occurring in C, S(C) contains a formula:

$\forall^*$ resulting-removal(U, A) $\Rightarrow$ after(U, $C_{A/\text{false}}$)
if A has positive polarity in C and does not occur within the scope of an existential quantifier

$\forall^*$ resulting-insertion(U, A) $\Rightarrow$ after(U, $C_{A/\text{true}}$)
if A has negative polarity in C and does not occur within the scope of an existential quantifier

$\forall^*$ resulting-removal(U, A) $\wedge$ option($C_{A/\text{true}}$) $\Rightarrow$ after(U, C)
if A has positive polarity in C and occurs in the scope of an existential quantifier

$\forall^*$ resulting-insertion(U, A) $\wedge$ option($C_{A/\text{false}}$) $\Rightarrow$ after(U, C)
if A has negative polarity in C and occurs in the scope of an existential quantifier

where $C_{A/x}$ denotes the formula obtained from C by replacing A by x, and where $\forall^*$ F denotes the universal closure of a formula F.

In some cases, the generated option does not constrain the variables occurring in the resulting insertion or resulting removal. This is the case, for example, for $C = \exists x\ p(x) \land \forall y\ q(y) \Rightarrow s(x, y)$. One of the formulas obtained from C according to Definition 14 is:

$$\forall z\ \text{resulting-insertion}(U, q(z)) \land \text{option}(\exists x\ p(x)) \Rightarrow \text{after}(U, \exists x\ p(x) \land \forall y\ q(y) \Rightarrow s(x, y))$$

Since z does not occur in option($\exists$x p(x)), this option expression does not constrain z and is therefore not worth considering.

The following result generalizes Propositions 1 and 2.

Proposition 3:
Let KB be a knowledge base, let U be an update of KB, and let C be a range-restricted static integrity constraint such that KB $\vdash$ C.

$$KB_U \vdash C \text{ if and only if } KB \cup \{\text{after}(U, A) \mid KB_U \vdash A\} \vdash F, \text{ for all } F \in S(C)$$

[*Proof:* By structural induction on C.]

Query and integrity constraint languages are usually implicitly assumed to be 'rectified' – i.e., they forbid to use the same symbol for denoting two distinct quantified variables in two different parts of a same formula. Rectification forces to rewrite the following static integrity constraint

$$[\forall x\ \text{employee}(x) \Rightarrow \text{speaks}(x, \text{English})] \land [\exists x\ \text{employee}(x) \land \text{speaks}(x, \text{German})]$$

as, for example:

$$[\forall x\ \text{employee}(x) \Rightarrow \text{speaks}(x, \text{English})] \land [\exists y\ \text{employee}(y) \land \text{speaks}(y, \text{German})]$$

We implicitly assume here that static integrity constraints are rectified.

3.2 Computation of Resulting Changes

In this section, we show how one can implement a hypothetical reasoning evaluator by means of deduction rules, in order to simulate query evaluation on an updated knowledge base, without actually performing the update. Such a query evaluator is needed, especially if concurrent accesses to the knowledge base are allowed. Indeed, an update might be rejected – for example if it violates some integrity constraints. In order to avoid, or at least to minimize, periods of time during which a knowledge base may be inconsistent and therefore not accessible to other users, one has to delay performing updates until they are proven acceptable. Classical database systems rely on special procedures for simulating updated states. We show in this section, that this is not needed for knowledge bases.

As already observed in Section 3.1, resulting changes can be defined by the following meta-rules:

$$\begin{array}{lcl}
\text{resulting-insertion}(U, F) & \leftarrow & \text{after}(U, F) \land \neg F \\
\text{resulting-removal}(U, F) & \leftarrow & F \land \text{after}(U, \neg F) \\
\text{resulting-insertion}(U, \neg F) & \leftarrow & \text{resulting-removal}(U, F) \\
\text{resulting-removal}(U, \neg F) & \leftarrow & \text{resulting-insertion}(U, F)
\end{array}$$

where F denotes a ground fact in the language $\mathcal{L}$(KB) of the knowledge base KB under consideration and U the considered update of KB.

These rules may, however, be sometimes rather inefficient. The following proposition suggests an interesting improvement.

> *Proposition 4:*
> Let KB be a knowledge base and U an update of KB. Let A and B be two formulas in the language $\mathcal{L}$(KB) of KB.
>
> The conjunction A ∧ B is a resulting insertion if A (B, resp.) is a resulting insertion and B (A, resp.) is true in KB_U.
>
> The conjunction A ∧ B is a resulting removal if A (B, resp.) is a resulting removal and B (A, resp.) is true in KB.

[*Proof:* By definition, A ∧ B is a resulting insertion (removal, resp.) if A ∧ B hold (does not hold, resp.) in the updated database KB_U but does not hold (holds, resp.) in KB. Proposition 4 follows.]

By Proposition 4, one can define resulting insertions by means of the following meta-rules:

```
resulting-insertion(U, F)         ← to-be-inserted(F)
resulting-insertion(U, F)         ← after(U, rule(F ← B)) ∧ resulting-insertion(B)
resulting-insertion(U, A ∧ B) ← resulting-insertion(U, A) ∧ after(U, B)
resulting-insertion(U, A ∧ B) ← after(U, A) ∧ resulting-insertion(U, B)
```

where the meta-predicate 'rule' ranges over the deduction rules. In a similar way, one can refine the definition of resulting removals, as well as resulting insertions and removals of general formulas containing disjunctions, quantifiers, etc.

3.3 Specialization as Partial Computation of Changes

In general, static integrity constraints cannot be represented by means of deduction rules since they may express conditions like disjunctive, existential, and negative statements that are not expressible in the formalism of deduction rules. However, the negation of such conditions can be expressed in terms of deduction rules. In other words, deduction rules can be used for expressing forbidden properties.

Consider for example a negative integrity constraint C = ¬p(a) precluding that p(a) holds. Relying on the propositional variable 'false', on could express it by means of the deduction rule:

$$\text{false} \leftarrow p(a)$$

C is violated as soon as 'false' is derivable. Similarly, a disjunctive integrity constraints q(b) ∨ r(c) could be represented by the following deduction rule:

$$\text{false} \leftarrow \neg q(b) \wedge \neg r(c)$$

Relying on negation in the same way gives rise to expressing existential constraints. For example, the integrity constraint $\exists x\ p(x)$ could be represented by:

$$\text{false} \leftarrow \neg\ [\exists x\ p(x)]$$

If integrity constraints are represented in such a manner as conditions for 'false', detecting if an update U to a knowledge base KB violates integrity constraints corresponds to determining whether 'false' is an insertion resulting from U. Moreover, the specialization of integrity constraints with respect to updates described in Section 3.1 reduces – under this representation of integrity constraints – to a computation of resulting changes.

4 Literature Overview

4.1 From Early Work to General Methods

In this section we want to illustrate the main developments in the field of integrity verification between 1974 and 1983. An evolution can be noticed from the realization that it might be a good idea to centrally impose the integrity of a database, via languages to do so, over special purpose techniques to efficiently test some specific constraints, to generally applicable methods to automatically simplify and evaluate constraints when updates are carried out.

One of the earliest papers addressing the issue of integrity constraints in databases is [Flo74]. The author notices that in computerized databases, the consistency of the stored data with the world that is represented, can best be preserved through the statement of logical constraints. He discerns between static and dynamic integrity constraints and remarks that integrity can best be maintained by checking the constraints when an update is carried out. Moreover, he indicates the possibility of using a kind of theorem prover to derive simplified constraints, but rejects the feasibility of such an approach in practice. Instead, the database designer is given the responsibility for formulating simple tests that can be efficiently evaluated. This process can also involve restructuring the database in such a way that checking certain constraints becomes easier.

Similar ideas can be found in [HM75]. In this paper, a distinction is made between domain and relation constraints. For each type, a rather abstract language, which is independent of any particular database system, is proposed in which constraints can be expressed. In this language, it is possible to state conditions that must be enforced, when to do so and which actions have to be taken when a violation occurs. Interesting for us is the fact that a large part of the responsibility for the decision when a particular constraint must be tested, rests with the person that formulates it. Moreover, no automatic simplification of constraints is incorporated. They must simply be formulated in a way that enables efficient testing.

In [Wil80], an approach is described to enhance existing databases with integrity checking capabilities. Again, the core of the methodology is a language in which integrity constraints, violation actions, costs and several other aspects can be specified. It incorporates facilities to indicate by which updates a constraint can be violated and how it must be instantiated when it is evaluated. It is the constraint designer who uses these facilities and who has to make sure that the actual behaviour of the system is efficient and in agreement with what was intended.

A different approach is taken in [Sto75]. Here, the formalism of the query language QUEL is used to formulate a way of treating a restricted class of integrity constraints with more automatization.

The database designer formulates constraints as universally quantified QUEL assertions. These can be divided in classes of increasing complexity. For each class, Stonebraker gives algorithms to modify insert- or replace-statements in such a way that the resulting database satisfies the constraints (if they were satisfied before the change was effected). Removals do not need attention because they cannot violate the constraints of the restricted form treated (except in the most complex case, and then no transaction modification is proposed anyway). These transformations can be executed automatically and result in quite efficiently executable tests, at least in the less complex cases. Finally, it must be noted that a replace-statement, although specified separately, is in fact more or less treated as a removal followed by an insertion.

It is interesting to compare the point of view taken in [Sto75] with the one we find in [GM79]. In the latter article, a procedural update language is proposed and integrity constraints are formulated as logic statements. Supposing they hold at the start of the transaction program, it is then proposed to use Hoare's program proving logic to investigate whether they can be demonstrated to hold at the end. In other words, at compile-time, transaction specifications would be checked for compliance with the integrity constraints, but they would not be modified. If a transaction program does not pass the test, it is simply not executed. A drawback of such a method seems to be the fact that each user writing transaction programs must explicitly pay attention to all integrity constraints that might be relevant. If he does not, it will be impossible for the system to prove that his program cannot violate integrity. In [Sto75] on the other hand, transaction programs are automatically transformed in such a way that they do not violate constraints. Of course, it must be noted that the class of transactions and constraints treated in [GM79] is more general than those addressed in [Sto75].

Finally, we can also mention [CD83]. In this paper, a more elaborate constraint checking facility for the database system INGRES is discussed and compared with the technique proposed in [Sto75].

An early paper about integrity constraints in deductive databases is [NY78]. Static and dynamic integrity constraints in the form of clauses are discussed. And the issues involved in working with (static) constraints together with derivation rules in one system are addressed. Two types of derivation rules are considered : Query-rules, which leave the defined facts implicit, and generation-rules, which are used in such a way that all derivable information is explicitly stored. As far as checking integrity is concerned, results are stated that allow to determine which constraints have to be checked when updates are performed. But nothing is said about possible simplifications of the identified constraints.

Some authors have considered restricted classes of constraints. And developed specialized techniques to test them efficiently.

[BBC80], e.g., addresses the problem of maintaining constraints that are closed tuple calculus formulas involving two tuple variables, at most one of which is existentially quantified. Moreover, the main part of the constraint is supposed to be a comparison ($\leq$) between some attributes of the tuples involved. As updates, single tuple insertions or removals are considered. The use of Hoare's program verification logic is proposed to verify that certain updates cannot violate certain constraints. For the remaining update-constraint pairs easily verifiable sufficient conditions for consistency are derived. These involve keeping extra aggregate data (minima and maxima) about the data stored in the database. A method is given to decide which data must be stored, how integrity constraints can be tested and how the stored data can be kept up to date.

The subject of [Rei81] is also rather specialized : How to treat type constraints in (a kind of) deductive databases. The author describes how typed formulas can be transformed into a set of formulas which is more manageable in the database. In this process, the constraints on types are also checked.

Finally, [Fdd81] can be mentioned as an example of a paper where an extremely specialized technique is used to treat one particular kind of constraint. Within an entity-relationship framework, it is described how one can efficiently test constraints demanding that for every entity of a certain type E1, there must exist an entity of type E2 that has a certain relation with it.

Other papers address the problem of finding general methods which can be applied to simplify and evaluate a broad class of constraints, independently from any particular system.

A first step in this direction seems to have been taken in [HK78]. There, it is proposed to analyze the influence that given updates can have on given integrity constraints. The result of this analysis would be conditions that can efficiently be tested at run-time. Different tests would be generated and a choice which alternative it is best to actually evaluate under the run-time circumstances would be made by a query optimizer.

Two articles that present more details are [WSK83] and [FW83]. In [WSK83], an extensive classification of integrity constraints is given. All constraints are (universally quantified) tuple calculus formulas. They are classified according to criteria such as : What is constrained, when they have to be invoked, how strict the conditions are they impose, how many tuples are involved. Algorithms are given to decide on whether constraints need testing given certain updates, to facilitate testing of constraints with functions such as sum, average, maximum. Finally, much attention is given to the subject of testing constraints at the end of complex transactions. Different methods to do this, involving early testing and/or protocolling are described and compared. We find different aspects treated in [FW83]. A general purpose declarative constraint language is proposed which would enable formulating (universally quantified) static integrity constraints in a kind of function-free binary predicate logic. The wish to keep the consistency of a set of formulas decidable is the reason for these strong restrictions. The main part of the paper addresses the question of how one can identify and instantiate constraints that need testing when an update is carried out.

But the two probably most influential papers that addressed these issues around the same time, are [Nic79] and [Bla81]. Nicolas treats integrity constraints formulated as first order logic formulas in prenex conjunctive normal form. He is the first one to point out the importance of having the range restricted property for integrity constraints. He proposes algorithms to decide which constraints must be checked when an insertion or removal is effected. And he shows how an affected constraint can often be instantiated and simplified in order to produce conditions which can, in the updated database, be more efficiently checked than the original constraint. Several alternative algorithms are given that produce either more conditions that are more instantiated or less conditions that are less instantiated. And a generalization of the method to transactions consisting of several removals and/or insertions is discussed. Although this paper presented a major step towards clear, general methods for the efficient treatment of integrity constraints, it does of course have its limitations. The main one is probably the fact that no methods are given to derive simplified tests for constraints that are affected through existentially quantified variables (or universally quantified variables within the scope of existentially quantified ones). Or, more precisely stated, Nicolas remarks that it is in general not possible to derive necessary and sufficient conditions for the truth of the constraint by simply instantiating these variables with values from the update. But he does not consider other techniques which might be useful. [Bla81] presents a quite comprehensive approach in a tuple calculus setting. Basically, the proposed techniques are similar to what can be found in [Nic79]. Some methods to treat existential quantifiers are also given. However, in [HI85], it is shown that they are not always correct in all circumstances. Furthermore, the work is restricted to constraints that do not contain several variables ranging over the same relation and only single tuple removals and insertions are considered. As a complement to this work, we can mention [BB82]. In this paper, a query optimization technique is presented. It is particularly useful for evaluating the kind of expressions that are typically produced as a result of the simplification techniques described in [Bla81].

To conclude this section, we give a few comments on [Laf82]. The subject of this paper is not the simplification of integrity constraints but when to test them. And, in this respect, a somewhat special view is taken. "Integrity dependencies" are introduced. These signalize for constraints which relations can be changed and which not, when they are violated (to recover from the violation). And these dependencies can be used for improving the efficiency of automatic integrity checking. For in some cases the verification of a constraint can be delayed until dependent relations occurring in it, are queried. In this way some of the checking work is done at data access time instead of at update time. Some simulations are discussed and the conclusion is that the proposed technique is particularly suitable for applications with a high update rate.

4.2 Specialization Methods in Deductive Databases

In this section, we present a comparative overview of papers presenting methods to deal with integrity constraints in deductive databases. We have made this selection because most of these papers are closely related to each other and together are a good illustration of a number of issues we wish to address. However, it should have become clear to the reader from the rest of this paper that we do not believe there exists any real difference between handling integrity constraints in relational and in deductive databases. Therefore, through future work, certainly other relevant recent papers will be included in this discussion.

In [AMM85], two methods to treat integrity constraints in (negation-free) deductive databases are proposed. The first is simply to evaluate the formulas: They must give "true" as a result. The second consists in modifying the facts and rules in the database in such a way that no information that violates the constraints can be derived. Notice that this can have remarkable consequences. Take, e.g., the constraint that every department must have a manager. If we then have a department in the database for which no manager is known, this method will not signal this fact, but simply pretend that department is not existing!

Most other papers contain a generalization of the method described in [Nic79]: They describe techniques to simplify integrity constraints with respect to updates and evaluate these simplified formulas to determine whether the updates violate constraints. We believe the main differences between most of these methods can be found in two aspects:

- How are resulting changes computed?
- Is constraint simplification and evaluation treated as part of the computation of resulting changes or is it regarded as a separate process?

Therefore, in our discussion, we will concentrate on these two issues. Like in earlier sections of this paper, we will speak about an update U. The database before the update is effected will be denoted KB. And the one that would result from performing the update, KB_U.

[Dec86] presents a method where only real (ground) changes are propagated. For every possible removal or insertion, a check is carried out whether they are in fact realized in KB_U (what we call phantomness test) and not in KB (what we call idleness test). Resulting insertions and removals are all computed, regardless whether they may affect static integrity constraints or not. For each computed resulting change, the relevant static constraints are suitably instantiated, simplified and evaluated.

[TKW85] and [LT85] contain elements of an integrity checking method that has been fully worked out in [LST87]. The method presented here, differs from the one described in [Dec86] in that it does not consider the factbase (the explicitly defined ground facts in KB) while deriving resulting changes. It propagates updates through rules, trying to instantiate resulting changes with constants or terms from the explicitly given update. No idleness or phantomness tests are performed, therefore less constraining premisses can be generated than possible. Consider for example a static integrity constraint $C = \forall x\ p(x) \Rightarrow q(x)$. In presence of the rule $p(x) \leftarrow q(x, y) \wedge r(y)$ and of the insertion of r(a), the method evaluates C in the updated database KB_U, instead of an expression equivalent to $\forall x$ resulting-insertion(U, p(x)) $\Rightarrow$ after(U, q(x)). Resulting change patterns - p(x) in our example - are taken to indicate which constraints must be (possibly) instantiated and evaluated. The authors propose a subsumption test for derived change patterns in order to avoid infinite loops. And they notice even this might not be completely satisfactory when complex terms with functors occur as arguments of predicates.

In [BD88] and [BDM88], an approach is described which has much in common with the previous one. But, for derived change patterns, affected constraints are not simply evaluated. Instead, there is a check first to see whether any instances of the pattern are indeed changes which are phantom nor idle. And only for each such instance is the simplified constraint evaluated. In the example considered above, a dynamic constraint similar - up to the names of predicates - to $\forall x$ resulting-insertion(U, p(x)) $\Rightarrow$ after(U, q(x)) is tested. A further distinguishing characteristic of this approach is the fact that its presentation relies heavily on the use of meta-programs. A short Prolog program is given for generating dynamic constraints from static ones. The hypothetical reasoning of Section 3.2 is given in these articles. The use of meta-programs adds an extra amount of flexibility, for one can imagine different strategies to execute these programs - see further. Finally, we can notice that compilation issues are given a lot of attention. Although only single tuple updates are explicitly cited in [BDM88], [BD88] shows how the very approach and programs given in [BDM88] also apply to set updates.

Another variant is presented in [AIM88]. Here, change patterns which affect constraints are tested for phantomness when they are insertions and for idleness in case of removals. In this way, they get a number of ground updates which are used to instantiate the constraints. For insertions, the instantiated constraints can be further simplified before being evaluated. In [AIM88], the claim is made that this method is superior to both the one proposed in [Dec86] and the one described in [LST87]. Although it is easy to attain a better performance than one of these two on a number of examples, and there might be even cases where both methods are beaten, it should be clear from this paper that we do not believe such a claim can hold true in general.

[DW89] shows yet another possible variant. When changes are propagated, there is always an immediate phantomness test for insertions. Derived deletions are checked for phantomness when it is discovered that they are involved in causing an addition. No idleness tests are performed, and therefore all computation can be done in the updated database KB_U. An interesting aspect of the method is the rewriting of integrity constraints. The negations of integrity constraints are rewritten as derivation rules (according to the method proposed in [LT84]). And when we have an insertion of the head of such a rule, integrity is violated. (This explains their particular choice concerning when to perform phantomness tests.) In this way, the computation of resulting changes and the simplification and evaluation of integrity constraints is unified in the way addressed in Section 3.3.

One of the first approaches with this latter characteristic is described in [SK87] and [KSS87]. Integrity constraints are written as denials and a modified version of SLDNF-resolution is introduced to enable reasoning forwards from updates. Removals are always immediately checked for phantomness and idleness. A phantomness test for insertions is also included. But exactly how and when it is performed depends on the computation rule which would be used. This introduces some flexibility that is lacking in most other proposals.

[MB88] combines the unified view on integrity checking and change propagation from the previous approach with the change propagation method proposed in [Dec86]. Moreover, the method is formulated in a set-oriented query evaluation framework. And compilation aspects are rather extensively addressed.

Within the same unified view, [MK88] stresses the fact that it is impossible to do well in all cases if you fix a global strategy for propagating changes. The authors illustrate with examples that it is necessary to be able to locally (at each particular step) choose between using a form of reasoning like in [Dec86] or like in [LST87] or somewhat in between. And they propose a formalism which would make this possible.

Most methods we discussed here, involve reasoning forwards from updates. Starting from explicitly given insertions and/or removals, resulting changes or patterns of possible resulting changes are computed. And finally, integrity constraints that might be influenced by these changes are evaluated. Whether or not a unified view on these two aspects is taken does of course not make a difference here. The method proposed in [ABI89] on the other hand, at run-time avoids all reasoning starting from updates. Predicate dependencies are pre-compiled and used at run-time to make a first selection of constraints that might be violated. For only those constraints that contain a predicate (with the right polarity) dependent on an updated predicate can be affected. The thus identified constraints are then basically evaluated as any other query. There is one difference: The computation rule used is such that as soon as possible the given update is taken into account. And when there is a branch without the possibility for using the update clause (the goal does not contain any literal with a predicate depending on the update predicate), it is not further elaborated. In fact, the method as it is presented is rather limited and suffers from some inefficiencies. But it does remind one of the fact that it is not always a good idea to reason forwards from updates. In fact, it is preferable not to make a fixed commitment to either way of reasoning. In [BDM88] and [BD88] this independence is achieved by relying on meta-rules that can be evaluated backward as well as forward. The choice of the one or the other evaluation strategy – or of a combination of both – is left to the query evaluator.

5 Conclusion

In this report, we aimed at giving a comprehensive state-of-the-art in integrity verification. A first part of the report was devoted to describe in a unified formalism features that have been previously proposed in different papers and formalized in different ways. We have shown that the notion of deduction rules permits one to express general updates – in particular set and sequential updates. Deduction rules can also be used for describing the hypothetical reasoning necessary for efficiently processing integrity constraints in concurrent knowledge base systems. Finally, we have shown that the specialization of static integrity constraints can be viewed as a partial computation of the meta-rules defining the hypothetical reasoning. A second part of this report was devoted to comparing most of the proposals for integrity verification that have been published over the last years. This study still needs to be refined and, maybe, extended. It is a first attempt towards a better understanding of the extensive amount of literature which has been published on integrity verification.

Acknowledgements

The research reported in this article has been partially supported by the European Community in the framework of the ESPRIT Basic Research Action *Compulog* No. 3012. Bern Martens is supported by the Belgian National Fund for Scientific Research.

We are indebted to our collegues of *Compulog,* ECRC, and the University of Leuven for helpful discussions.

References

[Abi88] S. Abiteboul. Updates, a New Frontier. In *Proc.* 2^{nd} *Int. Conf. on Database Theory (ICDT)*, 1988.

[ABI89] P. Asirelli, C. Billi, and P. Inverardi. Selective Refutation of Integrity Constraints in Deductive Databases. In *Proc.* 2^{nd} *Int. Symp. on Math. Fundamentals of Database Theory (MFDBS)*, June 1989.

[AIM88] P. Asirelli, P. Inverardi, and A. Mustaro. Improving Integrity Constraint Checking in Deductive Databases. In *Proc. Int. Conf. on Database Theory (ICDT)*, Sept. 1988.

[AMM85] P. Asirelli, De Santis M., and M. Martelli. Integrity Constraints in Logic Data Bases. *Journal of Logic Programming*, 2(3), 1985.

[AV87] S. Abiteboul and V. Vianu. A Transaction Language Complete for Update and Specification. In *Proc.* 6^{th} *ACM Symp. on Principles of Database Systems (PODS)*, 1987.

[AV88] S. Abiteboul and V. Vianu. Procedural and Declarative Database Update Languages. In *Proc.* 7^{th} *ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, Austin, Texas, March 1988.

[BB82] P. A. Bernstein and B. T. Blaustein. Fast Methods for Testing Quantified Relational Calculus Assertions. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD)*, June 1982.

[BBC80] B. Bernstein, B. T. Blaustein, and E. M. Clarke. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. In *Proc.* 6^{th} *Int. Conf. on Very Large Data Bases (VLDB)*, 1980.

[BD88] F. Bry and H. Decker. Préserver l'Integrité d'une Base de Données Déductive: une Méthode et son Implémentation. In *Proc.* $4^{\grave{e}mes}$ *Journéees Bases de Données Avancées (BDA)*, May 1988.

[BDM88] F. Bry, H. Decker, and R. Manthey. A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases. In *Proc.* 1^{st} *Int. Conf. Extending Database Technology (EDBT)*, March 1988.

[Bla81] B. T. Blaustein. *Enforcing Database Assertions: Techniques and Applications.* PhD thesis, Harvard Univ., Comp. Sc. Dept., Cambridge, Mass., Aug. 1981.

[Bry89a] F. Bry. Logical Rewritings for Improving the Evaluation of Quantified Queries. In *Proc. 2nd Int. Symp. on Mathematical Fundamentals of Data Base Theory (MFDBS)*, Visegrád, Hungary, June 1989. Springer-Verlag LNCS 364.

[Bry89b] F. Bry. Towards an Efficient Evaluation of General Queries: Quantifier and Disjunction Processing Revisited. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD)*, Portland, Oregon, May-June 1989.

[CD83] C. Cremers and G. Domann. AIM – An Integrity Monitor for the Database System Ingres. In *Proc. 9th Int. Conf. on Very Large Data Bases (VLDB)*, 1983.

[Dat85] C. J. Date. *A Guide to DB2.* Addison-Wesley, Reading, Massachusetts, 1985.

[Dec86] H. Decker. Integrity Enforcement on Deductive Databases. In *Proc. 1st Int. Conf. Expert Database Systems (EDS)*, April 1986.

[Di 69] R. A. Di Paola. The Recursive unsolvability of the Decision Problem for the Class of Definite Formulas. *Jour. of the ACM*, 16(2), 1969.

[dS88a] C. de Maindreville and E. Simon. A Production Rule Based Approach to Deductive Databases. In *Proc. 4th Int. Conf. on Data Engineering*, Los Angles, Calif., Feb. 1988.

[dS88b] C. de Maindreville and E. Simon. Modelling Queries and Updates in a deductive Database. In *Proc. 14th Int. Conf. on Very Large Data Bases (VLDB)*, Los Angles, Calif., Aug. 1988.

[DW89] S. K. Das and M. H. Williams. A Path Finding Method for Constraint Checking in Deductive Databases. *Data & Knowledge Engineering*, 4(3), 1989.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, 1972.

[Fag80] R. Fagin. Horn Clauses and Data Dependencies. In *Proc. 12th Annual ACM Symp. on Theory of Computing*, pages 123–134, 1980.

[Fdd81] A. L. Furtado, C. S. dos Santos, and J. M. V. de Castilho. Dynamic Modelling of a Simple Existence Constraint. *Information Systems*, 6, 1981.

[Flo74] J. J. Florentin. Consistency Auditing of Databases. *The Computer Journal*, 17(1), 1974.

[FV83] R. Fagin and M. Y. Vardi. Armstrong Databases for Functional and Inclusion Dependencies. *Information Processing Letters*, 16:13–19, Jan. 1983.

[FW83] R. A. Frost and S. Whittaker. A Step towards the Automatic Maintenance of the Semantic Integrity of Databases. *The Computer Journal*, 26(2), 1983.

[GM79] G. Gardarin and M. Melkanoff. Proving Consistency of Database Transactions. In *Proc. 5th Int. Conf. on Very Large Data Bases (VLDB)*, Sept. 1979.

[GMN84] H. Gallaire, J. Minker, and J.-M. Nicolas. Logic and Databases: A Deductive Approach. *ACM Computing Surveys*, 16(2):153–185, June 1984.

[HI85] A. Hsu and T. Imielinski. Integrity Checking for Multiple Updates. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD)*, 1985.

[HK78] M. Hammer and S. K. Karin. Efficient Monitoring of Database Assertions. In *Proc. Int. Conf. on Management of Data (SIGMOD)*, 1978.

[HM75] H. H. Hammer and D. J. McLeod. Semantic Integrity in a Relational Data Base System. In *Proc.* 1st *Int. Conf. on Very Large Data Bases (VLDB)*, 1975.

[KSS87] R. Kowalski, F. Sadri, and P. Soper. Integrity Checking in Deductive Databases. In *Proc.* 13th *Int. Conf. on Very Large Data Bases (VLDB)*, Brighton, UK, Sept. 1987.

[Kuh67] J. L. Kuhns. Answering Questions by Computer: A Logical Study. Technical Report RM-5428-PR, Rand Corp., 1967.

[Laf82] G. M. E. Lafue. Semantic Integrity Dependencies and Delayed Integrity Checking. In *Proc.* 8th *Int. Conf. on Very Large Data Bases (VLDB)*, 1982.

[LST87] J. W. Lloyd, E. A. Sonenberg, and R. W. Topor. Integrity Constraint Checking in Stratified Databases. *Jour. of Logic Programming*, 4(4), 1987.

[LT84] J. W. Lloyd and R. W. Topor. Making Prolog more Expressive. *Jour. of Logic Programming*, 1(3), 1984.

[LT85] J. W. Lloyd and R. W. Topor. A Basis for Deductive Database Systems. *Jour. of Logic Programming*, 2(2), 1985.

[LT86] J. W. Lloyd and R. W. Topor. A Basis for Deductive Database Systems II. *Jour. of Logic Programming*, 3(1):55–67, 1986.

[MB88] B. Martens and M. Bruynooghe. Integrity Constraint Checking in Deductive Databases Using a Rule/Goal Graph. In *Proc.* 2nd *Int. Conf. Expert Database Systems (EDS)*, April 1988.

[MK88] G. Moerkotte and S. Karl. Efficient Consistency Control in Deductive Databases. In *Proc.* 2nd *Int. Conf. on Database Theory (ICDT)*, 1988.

[MKW89] R. Manthey, V. Küchenhoff, and M. Wallace. KBL: Design Proposal for a Conceptual Language of EKS. Research Report TR-KB-29, ECRC, 1989.

[Nic79] J.-M. Nicolas. Logic for Improving Integrity Checking in Relational Databases. Technical report, ONERA-CERT, Feb. 1979. Also in *Acta Informatica 18(3), Dec. 1982, 227-253.*

[NK87] S. Naqvi and R. Krishnamurthy. Database Updates in logic Programming. In *Proc.* 7th *ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, pages 251–262, Austin, Texas, March 1987.

[NT89] S. Naqvi and S. Tsur. *A Logical Language for Data and Knowledge Bases.* Computer Science Press, New-York, 1989.

[NY78] J.-M. Nicolas and K. Yazdanian. *Logic and Data Bases*, chapter Integrity Checking in Deductive Databases. Plenum Press, New York, 1978.

[Rei81] R. Reiter. *Advances in Data Base Theory*, volume 1, chapter On the Integrity of Typed First-Order Data Bases. Plenum Press, New York, 1981.

[Rei84] R. Reiter. *On Conceptual Modelling*, chapter Towards a Logical Reconstruction of Relational Database Theory. Springer-Verlag, Berlin, New York, 1984.

[She88] J. C. Shepherdson. *Foundations of Deductive Databases and Logic Programming*, chapter Negation in Logic Programming, pages 19–88. Morgan Kaufmann, Los Altos, Calif., 1988.

[SK87] F. Sadri and R. Kowalski. A Theorem-Proving Approach to Database Integrity. In *Proc. Workshop on Foundations of Deductive Databases and Logic Programming*, 1987.

[Smu88] R. M. Smullyan. *First-Order Logic.* Springer-Verlag, Berlin, New-York, 1988.

[Sto75] M. Stonebraker. Implementation of Integrity Constraints and Views by Query Modification. In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, May 1975.

[TKW85] R. W. Topor, T. Keddis, and D. W. Wright. Deductive Database Tools. Technical Report 84/7, University of Melbourne, 1985.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, Rockville, Maryland, 1988.

[VGT87] A. Van Gelder and R. W. Topor. Safety and Correct Translation of Relational Calculus Formulas. In *Proc. 6th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, pages 317–327, San Diego, Calif., March 1987.

[Wil80] G. A. Wilson. A Conceptual Model for semantic Integrity Checking. In *Proc. 6th Int. Conf. on Very Large Data Bases (VLDB)*, Oct. 1980.

[WSK83] W. Weber, W. Stucky, and J. Karszt. Integrity Checking in Data Base Systems. *Information Systems*, 8(2), 1983.