Using Weakest Preconditions to Simplify Integrity Constraint Checking

Michael Lawley Rodney Topor
School of Computing and Information Technology
Griffith University, Nathan, Qld 4111, Australia

and

Mark Wallace
European Computer-Industry Research Centre
Arabellastraße 17, W-8000 München 81, Germany

ABSTRACT

We present a procedure for deriving the weakest precondition for a database update and an integrity constraint. We show how to simplify the weakest precondition to produce a condition to be evaluated before the update is performed. This provides an efficient means to ensure that database updates maintain integrity constraints.

Keywords Database, update, integrity constraint, weakest precondition

1 Introduction

In this paper we apply program derivation techniques [4, 5] to the task of efficiently checking the safety of database updates. Unlike many integrity constraint checking papers [2, 8, 10], we deal with the update "programs" themselves rather than just the resulting inserted and deleted tuples. In this sense our work is related to verification of transaction safety [11].

A database update is *safe* if it is guaranteed to maintain the truth of the database's integrity constraints [11]. We present a procedure for deriving the weakest precondition for an update and a constraint. This new condition can be combined with the update to form a conditional update. If the condition is true in the initial state, the constraint is guaranteed to be true after the update is performed. If the constraint is also true in the initial state, the conditional update is safe with respect to the constraint. In this case, we can further simplify the condition to be evaluated before the update is performed. Because the derivation of the weakest precondition and subsequent simplification is independent of the current database state, it only needs to be performed once, as a preprocessing step to produce the safe conditional update.

This provides a method for efficiently checking that updates maintain integrity constraints before performing them.

We consider a set oriented update language, based on that given by Wallace [12], and first-order conditions and integrity constraints. The update language is equivalent in power to SQL updates and more expressive than those considered by Bry et al. [2], Lloyd et al. [8], and Nicolas [10], We present a formal description of the effect of an update. We then show how to transform a constraint with respect to an update to produce a new condition. We prove that this new condition is the weakest precondition of the update and constraint, and show how to simplify the condition to be checked before performing the update.

2 Definitions

Let $\mathcal{U}$ be a *universal domain* consisting of a countably infinite set of constants. A *database* is a tuple $\langle R_1, \ldots, R_n \rangle$, where each R_i is a finite named relation over $\mathcal{U}^{k_i}$ for some $k_i \geq 0$. A *database update* U is a mapping from one database, $B = \langle R_1, \ldots, R_n \rangle$, to another, $U(B) = \langle R'_1, \ldots, R'_n \rangle$, where each R_i and R'_i have the same arity. We require the mapping to be partially recursive and C-generic for some finite set of constants C. (See [1] for an explanation of the C-genericity condition.)

A *formula* (resp., *sentence*) is a first-order, function-free formula (resp., sentence) in some fixed language. If R, $R_1, \ldots$ are relations, then r, $r_1, \ldots$ are the corresponding predicate symbols.

A *valuation* for a formula F is a mapping of the free variables of F to elements of $\mathcal{U}$. For a database B, formula F (resp., sentence G) and valuation ν for F, $B \models_\nu F$ (resp., $B \models G$) is defined as usual in model theory to mean F (resp., G) is true in B with respect to the valuation ν. We say a valuation ν' extends another valuation ν ($\nu' \geq \nu$) if the domain of ν' is a superset of the domain of ν and the restriction of ν' to the domain of ν is identical to ν.

A sentence F is a *precondition* for an update U and a sentence H if, for every database B, $B \models F$ implies $U(B) \models H$. A precondition F for U and H is a *weakest precondition* for U and H if, for every precondition G for U and H, and for every database B, $B \models G$ implies $B \models F$.

For later use in describing the effects of statements in our update language, we introduce the following notation and two simple lemmas. Let R be a relation in a database B, and $\Phi(\overline{x})$ a first-order formula with free variables $\overline{x} = x_1, \ldots, x_n$. Then $B' = B[R \mapsto R']$ denotes the result of replacing the relation R in B by R'. In practice, R' is either $R \cup \{\overline{x} \mid B \models \Phi(\overline{x})\}$ or $R - \{\overline{x} \mid B \models \Phi(\overline{x})\}$, i.e. $\Phi(\overline{x})$ is evaluated in the current database B and the resulting tuples are added to (resp., deleted from) R.

Lemma 2.1 $B[R \mapsto R \cup \{\overline{x} \mid B \models \Phi(\overline{x})\}] \models_\nu r(\overline{u})$ if and only if $B \models_\nu r(\overline{u}) \vee \Phi(\overline{u})$.

Lemma 2.2 $B[R \mapsto R - \{\overline{x} \mid B \models \Phi(\overline{x})\}] \models_\nu r(\overline{u})$ if and only if $B \models_\nu r(\overline{u}) \wedge \neg\Phi(\overline{u})$.

3 Updates

Our update language is adapted from Wallace [12]. It is syntactically simpler but can be shown to have the same expressive power [7]. Let R be a named relation, and $\Phi(\overline{x})$ a first-order formula with free variables $\overline{x}$. Then **foreach** $\overline{x} : \Phi(\overline{x})$ **do** $\mathit{insert}_R(\overline{x})$, and **foreach** $\overline{x} : \Phi(\overline{x})$ **do** $\mathit{delete}_R(\overline{x})$ are statements in our language. If $S_1, \ldots, S_n$ are statements in our language then $(S_1 \ ; \ \ldots \ ; \ S_n)$ is a statement in our language. For example, adding the tuple $\overline{a}$ to the relation R would be written **foreach** $\overline{x} : \overline{x} = \overline{a}$ **do** $\mathit{insert}_R(\overline{x})$. Sometimes we will abbreviate this statement as $\mathit{insert}_R(\overline{a})$.

The effect of a statement in our language is given informally as follows. The statement **foreach** $\overline{x} : \Phi(\overline{x})$ **do** $\mathit{insert}_R(\overline{x})$ (resp. **foreach** $\overline{x} : \Phi(\overline{x})$ **do** $\mathit{delete}_R(\overline{x})$) is executed by evaluating $\Phi(\overline{x})$ to produce a set of bindings for $\overline{x}$ and then adding to (resp. deleting from) R each $\overline{x}$ tuple. The statement $(S_1 \ ; \ \ldots \ ; \ S_n)$ is executed by first executing S_1, then executing S_2, and so on.

Example 3.1 To give a salary increase of 10% to all employees who earn over \$10,000 we could write the following statement sequence.

foreach $e, s : \mathit{emp}(e, s) \wedge s > 10000$ **do** $\mathit{insert}_T(e, s)$;
foreach $e, s : t(e, s)$ **do** $\mathit{delete}_{\mathit{Emp}}(e, s)$;
foreach $e, s, \mathit{snew} : t(e, s) \wedge \mathit{snew} = s \times 1.1$ **do** $\mathit{insert}_{\mathit{Emp}}(e, \mathit{snew})$

where T is a suitable, initially empty, temporary relation. □

The update defined by a statement S in our update language is denoted $[\![S]\!]$ and is given as follows.

1. $[\![$**foreach** $\overline{x} : \Phi(\overline{x})$ **do** $\mathit{insert}_R(\overline{x})]\!](B) = B[R \mapsto R \cup \{\overline{x} \mid B \models \Phi(\overline{x})\}]$
2. $[\![$**foreach** $\overline{x} : \Phi(\overline{x})$ **do** $\mathit{delete}_R(\overline{x})]\!](B) = B[R \mapsto R - \{\overline{x} \mid B \models \Phi(\overline{x})\}]$
3. $[\![S_1 \ ; \ \ldots \ ; \ S_n]\!](\mathrm{B}) = [\![S_n]\!](\ldots[\![S_1]\!](\mathrm{B})\ldots)$

Clearly, for every statement S, the mapping $[\![S]\!]$ is a database update.

Lemma 3.1 *Let S be a statement and H a sentence. Then a sentence F is a weakest precondition for $[\![S]\!]$ and H if, for every database B, $B \models F$ if and only if $[\![S]\!](B) \models H$.*

Proof Suppose that, for every database B, $B \models F$ if and only if $[\![S]\!](B) \models H$. Clearly, F is a precondition for $[\![S]\!]$ and H. Further, for all databases B and preconditions G for $[\![S]\!]$ and H, $B \models G$ implies $[\![S]\!](B) \models H$, as G is a precondition for $[\![S]\!]$ and H, and hence, by assumption, $B \models F$. That is, F is a weakest precondition for $[\![S]\!]$ and H. □

4 Constraint transformation

In the context of program derivation, Dijkstra [4] defined a predicate transformer wp as follows. For any command S and predicate H, $wp(S, H)$ denotes another predicate F such that execution of S in a state satisfying F guarantees that S will terminate in a state satisfying H.

Similarly, in our database context, we define a constraint transformer wp for a statement S and a formula H. Informally, execution of S in a state satisfying $wp(S, H)$ guarantees that S will terminate in a state satisfying H.

Let H be a (possibly open) formula, Φ a formula with k free variables and r a k-ary predicate symbol, for some $k \geq 0$. We write $H[r \mapsto r \cup \Phi]$ (resp., $H[r \mapsto r - \Phi]$) for the formula resulting from the replacement of every occurrence of $r(\overline{s})$ in H by $(r(\overline{s}) \vee \Phi(\overline{s}))$ (resp., $(r(\overline{s}) \wedge \neg\Phi(\overline{s}))$).

Definition Let S be a statement in normal form and H a (possibly open) formula. Then $wp(S, H)$ is defined as follows:

1. $wp(\textbf{foreach } \overline{x} : \Phi(\overline{x}) \textbf{ do } \mathit{insert}_R(\overline{x}), H) = H[r \mapsto r \cup \Phi]$
2. $wp(\textbf{foreach } \overline{x} : \Phi(\overline{x}) \textbf{ do } \mathit{delete}_R(\overline{x}), H) = H[r \mapsto r - \Phi]$
3. $wp(S_1 \ ; \ \ldots \ ; \ S_n), H) = wp(S_1, \ldots wp(S_n, H) \ldots)$

Note that these equations define wp, and not the semantics of statements as in [5].

The following example illustrates how to construct $wp(S, H)$ according to the definition above.

Example 4.1 Let H be the constraint $\forall\overline{x}\ (p(\overline{x}) \rightarrow q(\overline{x}))$, and S the statement $S1 \ ; \ S2$, where $S1$ is the statement **foreach** $\overline{x} : r(\overline{x})$ **do** $\mathit{delete}_P(\overline{x})$, and $S2$ is the statement **foreach** $\overline{y} : s(\overline{y})$ **do** $\mathit{insert}_P(\overline{y})$. Then,

$$\begin{aligned}
wp(S, H) &= wp(S1; S2, H) \\
&= wp(S1, wp(S2, H)) \\
&= wp(S1, \forall\overline{x}\ ((p(\overline{x}) \vee s(\overline{x})) \rightarrow q(\overline{x})) \\
&= \forall\overline{x}\ (((p(\overline{x}) \wedge \neg r(\overline{x})) \vee s(\overline{x})) \rightarrow q(\overline{x}))
\end{aligned}$$

□

We now present a series of lemmas leading to a proof that $wp(S, H)$ is the weakest precondition for $[\![S]\!]$ and H.

Lemma 4.1 *Let S be the statement* **foreach** $\overline{x} : \Phi(\overline{x})$ **do** $\mathit{insert}_R(\overline{x})$*, H a formula with free variables $\overline{z}$, and ν a valuation for $\overline{z}$. Then, for every database B, $B \models_\nu wp(S, H)$ if and only if $[\![S]\!](B) \models_\nu H$.*

Proof The proof is by induction on the structure of H. Lemma 2.1 is used in the base case, $H = r(\overline{x})$. Simple induction is used when $H = H_1 \wedge H_2$, $H = H_1 \vee H_2$, or $H = \neg H_1$. The only nontrivial case is $H = \forall \overline{y}\ H'$, which is given in full below. (By renaming, we can assume that $\overline{x}$ and $\overline{y}$ are disjoint.)

$$\begin{array}{lll}
B \models_\nu wp(S, \forall \overline{y}\ H') & & \\
\iff & B \models_\nu (\forall \overline{y}\ H')[r \mapsto r \cup \Phi] & \text{(definition of } wp\text{)} \\
\iff & \text{for all } \nu' \geq \nu,\ B \models_{\nu'} H'[r \mapsto r \cup \Phi] & \text{(semantics of } \forall\text{)} \\
\iff & \text{for all } \nu' \geq \nu,\ B \models_{\nu'} wp(S, H') & \text{(definition of } wp\text{)} \\
\iff & \text{for all } \nu' \geq \nu,\ [\![S]\!](B) \models_{\nu'} H' & \text{(induction hypothesis)} \\
\iff & [\![S]\!](B) \models_\nu \forall \overline{y}\ H' & \text{(semantics of } \forall\text{)}
\end{array}$$

□

Lemma 4.2 *Let S be the statement* **foreach** $\overline{x} : \Phi(\overline{x})$ **do** $delete_R(\overline{x})$*, H a formula with free variables $\overline{z}$, and ν a valuation for $\overline{z}$. Then, for every database B, $B \models_\nu wp(S, H)$ if and only if $[\![S]\!](B) \models_\nu H$.*

Proof As for Lemma 4.1, but using Lemma 2.2 in place of Lemma 2.1. □

Lemma 4.3 *Let S be the sequential statement $(S_1\ ;\ \ldots\ ;\ S_n)$ and H a sentence. Then, for every database B, $B \models wp(S, H)$ if and only if $[\![S]\!](B) \models H$.*

Proof Assume inductively that, for every database B, sentence H', and statement sequence $S_1\ ;\ \ldots\ ;\ S_i$, $1 \leq i < n$, we have $B \models wp((S_1\ ;\ \ldots\ ;\ S_i), H')$ if and only if $[\![S_1\ ;\ \ldots\ ;\ S_i]\!](B) \models H'$. Then

$$\begin{array}{lll}
B \models wp((S_1\ ;\ \ldots\ ;\ S_n), H) & & \\
\iff & B \models wp((S_1\ ;\ \ldots\ ;\ S_{n-1}), wp(S_n, H)) & \text{(definition of } wp\text{)} \\
\iff & [\![S_1\ ;\ \ldots\ ;\ S_{n-1}]\!](B) \models wp(S_n, H)) & \text{(induction hypothesis)} \\
\iff & [\![S_n]\!]([\![S_1\ ;\ \ldots\ ;\ S_{n-1}]\!](B)) \models H & \text{(induction hypothesis)} \\
\iff & [\![S_1\ ;\ \ldots\ ;\ S_n]\!](B) \models H & \text{(definition of update)}
\end{array}$$

Lemmas 4.1 and 4.2 (with ν empty) provide the base cases. □

Theorem 4.1 *Let S be a statement, and H a sentence. Then $wp(S, H)$ is the weakest precondition for $[\![S]\!]$ and H.*

Proof The result follows immediately from Lemmas 3.1, 4.1, 4.2,and 4.3. □

To apply Theorem 4.1 we construct $wp(S, H)$ and evaluate it before performing the statement S. In the next section we describe how $wp(S, H)$ can be simplified before it is evaluated.

5 Simplification

To simplify the condition $wp(S, H)$ we take advantage of the fact that H is already true in the initial database B. In this case we show how to construct a simpler formula wp' such that wp' is true in B if and only if $wp(S, H)$ is true in B.

The following example illustrates the derivation of a weakest precondition and its subsequent simplification.

Example 5.1 Let H be the constraint $\forall\overline{x}\ (p(\overline{x}) \rightarrow q(\overline{x}))$, and S the statement **foreach** $\overline{x} : r(\overline{x})$ **do** $insert_P(\overline{x})$. Then,

$$\begin{aligned} wp(S, H) &= \forall\overline{x}\ (p(\overline{x}) \vee r(\overline{x}) \rightarrow q(\overline{x})) \\ &\equiv \forall\overline{x}\ (p(\overline{x}) \rightarrow q(\overline{x}))\ \wedge\ \forall\overline{x}\ (r(\overline{x}) \rightarrow q(\overline{x})) \\ &\equiv H\ \wedge\ \forall\overline{x}\ (r(\overline{x}) \rightarrow q(\overline{x})) \end{aligned}$$

Given that H is true in B, we thus need only check $wp' = \forall\overline{x}\ (r(\overline{x}) \rightarrow q(\overline{x}))$ to determine whether $wp(S, H)$ is true in B and, hence, whether performing S will cause the new database to satisfy the constraint H. Checking $\forall\overline{x}\ (r(\overline{x}) \rightarrow q(\overline{x}))$ in the current database is simpler than checking H in the new database because we only need to check $q(\overline{x})$ for the new values of $\overline{x}$ that will be added to P. $\square$

We have the following results describing some circumstances under which a constraint of a certain form is simplifiable with respect to an update of a certain form, given that the constraint is true in the current database. Without loss of generality we consider constraints in prenex conjunctive normal form.

Clearly, an update which only inserts into (resp., deletes from) predicates appearing positively (resp., negatively) in a constraint cannot cause the constraint to be violated in the updated database if it was true in the current database.

More interestingly, the following lemma gives a further condition under which $wp(S, H)$ can be simplified.

Lemma 5.1 *Let B be a database, D_i a disjunction of literals l_{ij}, H a constraint of the form $\forall\overline{x}\ (D_1 \wedge \ldots \wedge D_n)$, and S a statement. Suppose that the constraint H is true in B and that some predicate that S inserts (resp., deletes) occurs negatively (resp., positively) in H. Then there exists a simpler formula, wp' such that wp' is true in B if and only if $wp(S, H)$ is true in B.*

Proof For simplicity of presentation, we assume S is a single *foreach* statement and the updated predicate appears only once in the constraint H. Let S be the statement **foreach** $\overline{x}' : r(\overline{x}')$ **do** $insert_P(\overline{x}')$, and l_{ij} be $\neg p(\overline{s})$. Then,

$$\begin{aligned} wp(S, H) &= \forall\overline{x}\ (D_1 \wedge \ldots \wedge (l_{i1} \vee \ldots \vee \neg(p(\overline{s}) \vee r(\overline{s})) \vee \ldots \vee l_{ik_i}) \wedge \ldots \wedge D_n) \\ &\equiv \forall\overline{x}\ (D_1 \wedge \ldots \wedge (l_{i1} \vee \ldots \vee (\neg p(\overline{s}) \wedge \neg r(\overline{s})) \vee \ldots \vee l_{ik_i}) \wedge \ldots \wedge D_n) \\ &\equiv \forall\overline{x}\ (D_1 \wedge \ldots \wedge D_n \wedge (l_{i1} \vee \ldots \vee l_{ij-1} \vee \neg r(\overline{s}) \vee l_{ij+1} \vee \ldots \vee l_{ik_i})) \\ &\equiv H \wedge \forall\overline{x}\ (l_{i1} \vee \ldots \vee l_{ij-1} \vee \neg r(\overline{s}) \vee l_{ij+1} \vee \ldots \vee l_{ik_i}) \\ &\equiv \forall\overline{x}\ (r(\overline{s}) \rightarrow l_{i1} \vee \ldots \vee l_{ij-1} \vee l_{ij+1} \vee \ldots \vee l_{ik_i}) \end{aligned}$$

given that H is true in B. This last formula is simpler to check than $wp(S, H)$ because it consists of only one of the two conjuncts in a formula equivalent to $wp(S, H)$. $\Box$

Both referential integrity constraints and functional dependencies can be expressed as constraints in the form required by Lemma 5.1. Hence they can be simplified in this manner.

Depending on the form of S and H, greater simplifications than Lemma 5.1 suggests may be possible.

Example 5.2 Let H be the constraint $\forall \overline{x}\ (p(\overline{x}) \to q(\overline{x}))$, and S the statement **foreach** $\overline{x} : r(\overline{x})$ **do** $\mathit{insert}_P(\overline{x})$; **foreach** $\overline{x} : r(\overline{x})$ **do** $\mathit{insert}_Q(\overline{x})$. Then,

$$\begin{aligned}
wp(S, H) &= \forall \overline{x}\ (p(\overline{x}) \vee r(\overline{x}) \to q(\overline{x}) \vee r(\overline{x})) \\
&\equiv \forall \overline{x}\ (p(\overline{x}) \to (q(\overline{x}) \vee r(\overline{x})))\ \wedge\ \forall \overline{x}\ (r(\overline{x}) \to (q(\overline{x}) \vee r(\overline{x}))) \\
&\equiv \forall \overline{x}\ (p(\overline{x}) \to (q(\overline{x}) \vee r(\overline{x}))) \\
&\equiv \mathit{true}
\end{aligned}$$

given that H is true in B. Thus S may be performed safely without checking $wp(S, H)$ at all. $\Box$

It is worth noting that for this example, the methods described in [2], [8], and [10] would all require checks equivalent to $\forall \overline{x}\ (r(\overline{x}) \to q(\overline{x}))$.

Under certain circumstances, notably when inserting and deleting facts, we may also simplify the weakest precondition when the constraint contains an existential quantifier following the outermost universal quantifier.

Example 5.3 (See [9, p. 58].) Let H be the constraint $\forall x \exists y\ (p(x) \to q(x, y))$, and S the statement $\mathit{delete}_Q(a, b)$. Then,

$$\begin{aligned}
wp(S, H) &= \forall x \exists y\ (p(x) \to q(x, y) \wedge x, y \neq a, b) \\
&\equiv \forall x \exists y\ (x \neq a \to (p(x) \to q(x, y) \wedge x, y \neq a, b)) \wedge \\
&\quad\ \forall x \exists y\ (x = a \to (p(x) \to q(x, y) \wedge x, y \neq a, b)) \\
&\equiv \forall x \exists y\ (x \neq a \to (p(x) \to q(x, y))) \wedge \exists y\ (p(a) \to q(a, y) \wedge y \neq b) \\
&\equiv \forall x \exists y\ (p(x) \to q(x, y)) \wedge \exists y\ (p(a) \to q(a, y) \wedge y \neq b) \\
&\equiv H \wedge \exists y\ (p(a) \to q(a, y) \wedge y \neq b) \\
&\equiv \exists y\ (p(a) \to q(a, y) \wedge y \neq b)
\end{aligned}$$

given that H is true in B. $\Box$

Furthermore, because we treat updates as a whole, we can also simplify some constraints whose outermost quantifier is existential.

Example 5.4 Let H be the constraint $\exists\overline{x}\ p(\overline{x})$, and S the statement sequence $\mathit{delete}_P(\overline{a})$; $\mathit{insert}_P(\overline{c})$. Then,

$$\begin{array}{rcl} wp(S,H) & = & \exists\overline{x}\ (p(\overline{x}) \wedge \neg(\overline{x}=\overline{a})) \vee \overline{x}=\overline{c} \\ & \equiv & \exists\overline{x}\ (p(\overline{x}) \wedge \neg(\overline{x}=\overline{a})) \vee \exists\overline{x}\ \overline{x}=\overline{c} \\ & \equiv & \exists\overline{x}\ (p(\overline{x}) \wedge \neg(\overline{x}=\overline{a})) \vee \mathit{true} \\ & \equiv & \mathit{true} \end{array}$$

given that H is true in B. $\square$

Of course, it is not always possible to simplify $wp(S,H)$.

Example 5.5 Let H be the constraint $\exists\overline{x}\ p(\overline{x})$, and S the statement $\mathit{delete}_P(\overline{a})$. Then,

$$wp(S,H) = \exists\overline{x}\ (p(\overline{x}) \wedge \neg(\overline{x}=\overline{a}))$$

which cannot be simplified. $\square$

At present, we are unable to completely characterise the conditions under which a constraint is simplifiable with respect to an update. However, we can identify certain important special cases including constraints expressing referential integrity and functional dependencies. Also, we note that the problem of simplifying $wp(S,H)$ given that H is true in B, is a special case of semantic query optimisation [3]. Hence, the techniques applicable to semantic query optimisation are also applicable here and offer further avenues for simplification.

6 Conclusion

The ability to compute the weakest precondition of a database update and a first-order sentence leads to an efficient method to ensure that database updates maintain the truth of integrity constraints.

In the context of relational databases, our method of constraint simplification is at least as good as, and offers the following advantages over, the methods described in [2], [8], and [10].

- The simplified constraints can be evaluated before the update is performed.
- The update is considered as a whole rather than just a set of separate updates. This allows simpler constraints to be generated.
- A more powerful update language is considered which allows parameterised, set at a time, updates.

In contrast to the method described in [6] and [9] we describe a simple method for constructing the weakest precondition (for a more expressive update language) and explicit conditions under which it can be simplified. These include the common constraints expressing referential integrity and functional dependencies.

McCune and Henschen [9] describe a method for integrity constraint simplification which effectively calculates $wp(S, H)$. An incomplete method for simplifying $wp(S, H)$ is described but no claims about its effectiveness are made.

Stemple and Sheard [11] describe a method of integrity constraint simplification which also operates on the update procedure. Their update language is less powerful than ours and it is not clear how much simplification is possible with their method which is based around a modified Boyer-Moore theorem prover.

We believe our approach to be both simpler in presentation and more general in scope than the above methods. We treat a more powerful update language and derive simpler conditions to check. Our method can be applied without change to derived relations defined by non-recursive rules.

Wallace [12] describes a method for compiling integrity constraints into update procedures. He deals with an equivalent update language and also handles recursive rules. His method extends the methods based on "recursive update consequences" [2] by considering a more powerful update language and a transformation similar to wp. It produces simplifications similar to our method.

Extending the generation of weakest preconditions to deductive databases with recursive rules and integrity constraints is a straightforward task. It remains to be done to apply the semantic query optimisation techniques extended for recursive rules described in [3] to help simplify weakest preconditions with recursion.

References

[1] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences*, 41(1):181–229, 1990.

[2] F. Bry, H. Decker, and R. Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *Proc. First International Conference on Extending Database Technology*, pages 488–505, Venice, Italy, Feb. 1988.

[3] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[4] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, Aug. 1975.

[5] D. Gries. *The Science of Programming.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1981.

[6] L. J. Henschen, W. W. McCune, and S. A. Naqvi. Compiling constraint-checking programs from first-order formulas. In H. Gallaire, J. Minker, and J.-M. Nicolas, editors, *Advances in Data Base Theory*, pages 145–169. Plenum Press, New York, 1984.

[7] M. Lawley. On the power of database update languages. In G. Gupta and C. Keen, editors, *Proc. of the 15th Australian Computer Science Conference*, pages 517–528, Hobart, Australia, Jan. 1992.

[8] J. W. Lloyd, E. A. Sonenberg, and R. W. Topor. Integrity constraint checking in stratified databases. *Journal of Logic Programming*, 4(4):331–343, Dec. 1987.

[9] W. W. McCune and L. J. Henschen. Maintaining state constraints in relational databases: A proof theoretic basis. *Journal of the ACM*, 36(1):46–68, 1989.

[10] J.-M. Nicolas. Logic for improving integrity checking in relational database. *Acta Informatica*, 18:227–253, 1982.

[11] T. Sheard and D. Stemple. Automatic verification of database transaction safety. *ACM Transactions on Database Systems*, 14(3):322–368, Sept. 1989.

[12] M. Wallace. Compiling integrity checking into update procedures. In *Proc. Twelfth International Joint Conference on Artificial Intelligence*, pages 903–908, Aug. 1991.