Accommodating Integrity Constraints During Database Design

Dimitris Plexousakis and John Mylopoulos
Department of Computer Science
University of Toronto
Toronto, M5S 1A4, Canada
E-mail: {dp,jm}@ai.toronto.edu

June 28, 1995

Abstract

We address the problem of maintaining the integrity of large knowledge bases using a compile-time transaction modification technique. The novelty of the approach lies in the adaptation of ideas from Artificial Intelligence (AI) planning research. Specifically, starting with the observation that solutions to the frame and ramification problems can be used during database transaction design time, we propose an integrity maintenance technique that modifies transaction specifications by incorporating into them conditions necessary of the constraints' satisfaction. Additions to the transactions' postconditions whose effect is to maintain the integrity constraints, are generated from a set of (determinate) transaction specifications. Thus, the implications of constraints are realized by the transaction specifier and the effort of having to prove transaction safety is saved, since it is guaranteed by the correctness of the generation process. We envision the development of a tool that, given a set of determinate transaction specifications, automatically suggests additions to the transaction postconditions whose effect is to maintain the integrity constraints.

Keywords: Integrity constraints in large knowledge bases, transaction specification, database design, temporal constraints and rules.

1 Introduction

Integrity constraints specify the valid states of a data or knowledge base as well as its allowable state transitions [Flo74]. Structural integrity constraints express properties of the data model used for representing knowledge. Semantic integrity constraints on the other hand, are user-defined and express properties of the domain being modeled. The maintenance of semantic integrity constitutes a major performance bottleneck in database management systems. The majority of commercial products do not provide but for the enforcement of very limited types of integrity constraints. The maintenance of general semantic constraints becomes the responsibility of the user submitting transactions to the database. In large, semantically rich knowledge bases the problem becomes even harder due to the large numbers of constraints and due to implicit updates caused by the presence of deductive rules. Compile-time processing of constraints and transactions aims at reducing the run-time cost associated with the enforcement of constraints, and is, in our view, a promising avenue for achieving acceptable performance.

The problem of integrity maintenance has received considerable attention in the literature of the past decade. Most proposed techniques for integrity maintenance rely on monitoring by a generic application-independent program that verifies that updates do not violate semantic constraints [Nic82],[LST86], [BDM88], [JJ91], [Cho92], [Ple93a], [Ten95], [Ple95]. Another popular approach is based on maintenance by transactions [Sto75], [GM79], [Lip90], [CW90], [LTW93]. The former technique relieves the user from the burden of implementing database transactions in a way such that no constraint is violated. The latter requires that the specification of updating transactions is given. Transactions are then modified so that database integrity constraints are guaranteed to hold in any executable sequence of transactions.

In our earlier work [Ple93a], we presented a compile-time simplification method for temporal deductive knowledge bases, where integrity constraints, specified in a many-sorted temporal assertion language [MBJK90],[Ple93b], are specialized and simplified with respect to the anticipated updates. In this paper, we undertake a dual approach, namely compile-time transaction modification in the context of knowledge bases containing (temporal) integrity constraints and deductive rules. In particular, we examine how transactions specified by means of *precondition-postcondition* pairs, can be modified so that the constraints they - directly or indirectly - affect are guaranteed not to be violated in the state resulting from transaction execution. Specifying database transaction by means of pre- and post-conditions, permits the database designer to express *what* the effect of transactions should be and not *how* this effect should be accomplished. Database transaction specifications, should be given in a formal notation that possesses both notational suitability and the capacity of supporting formal treatment. The formal specification language should be accompanied by the appropriate machinery for proving properties of specifications. We choose first-order (temporal) logic as the language for expressing transaction pre/post-conditions, constraints and rules. We address the problem of proving integrity maintenance by relating it to that of reasoning about actions [GS87] and elaborate on the impact that the *frame problem* [McC69] and *ramification problem* [Fin88] have in transaction specifications.

In the majority of the existing methods for integrity constraint maintenance by transaction modification, the frame and ramification problems have either been ignored or bypassed by means of implicit assumptions that state that "nothing but what is explicitly declared to change in the update procedure does". Section 5 addresses the inadequacies of existing methods regarding the frame problem. Given a set of transaction specifications, the problem of succinctly stating that "nothing else changes" except the aspects of the state explicitly specified, has been called the *frame problem.* The *ramification* problem amounts to devising a way to avoid having to specify indirect effects of actions explicitly. Several attempts to solve these problems have appeared in the AI planning literature of the recent years. [LR94] presents a solution to the frame problem with application to database updates. [BMR93] and [BMR95] show why the frame problem becomes particularly acute in object-oriented specifications where transactions are inherited and specialized from superclasses to subclasses. The arguments presented therein apply directly to the specification of database transactions in deductive, active and object-oriented databases. The effects of rule evaluation or firing have to be accounted for and checked against integrity constraints. Moreover, transaction specifications can be specialized from superclasses to subclasses and conjoined to form complex transactions. The effects of specialized or synthesized transactions must be precisely characterized. To the best of our knowledge, there has not been any previous attempt to link the ramification problem with that of proving safety of transactions. In fact, a solution of the ramification problem can suggest a strategy for integrity maintenance: transform the specifications of transactions to embody implications of constraints (these may be simpler formulae than the constraints themselves); if the new specifications are not met then the constraints are violated. Embodying the implications of constraints in transaction specifications means that the constraints need not be checked at run-time.

The solution proposed in [LR94] consists of systematically deriving a set of *successor-state* axioms that completely describe how *fluents*[1] can change truth value as a result of some action taking place. These axioms formulate closed-world assumptions for actions. A syntactic generator of successor-state axioms in the presence of binary constraints and stratified definitions of non-primitive fluents is proposed in [Pin94]. We relate this solution to the problem of integrity maintenance in the context of temporal deductive KBs. The following example shows the rationale behind the use of ramifications for simplifying the task of proving transaction safety.

Example 1.1 Transaction *EnrollInCourse* records the enrollment of student *st* in course *crs*. *size* and *classlimit* are function symbols representing the class size and enrollment limit respectively, whereas *EnrolledIn* is a predicate symbol. We adopt the unprimed/primed notation[2] to refer to the values of variables, functions or predicates immediately before and after the execution of the transaction.

EnrollInCourse (st, crs) `Precondition:` $\neg EnrolledIn(st, crs)$ `Postcondition:` $size'(crs) = size(crs) + 1 \;\wedge\; EnrolledIn'(st, crs)$
`Invariant:` $\forall c/Course\ size(c) \leq classlimit(c)$

In this specification, the precondition requires that the student is not already enrolled in the course, whereas the postcondition specifies the effect of the transaction on the final state: the size of the class is incremented and the predicate *EnrolledIn* becomes true of *st* and *crs*. The invariant requires that the size of a course should not exceed its limit. We can easily verify that, if the condition $(size(crs) + 1 \leq classlimit(crs))$ is conjoined with the postcondition, then no implementation that meets the augmented postcondition can violate the invariant. Indeed, the invariant follows as a logical consequence of the augmented postcondition, the invariant in the previous state and the assumption that in the state before transaction execution the invariants are known to be satisfied. As will be shown in the sequel, the derived addition to the postcondition is a ramification of (a form of) the invariant and the initial postcondition. This is different than simply conjoining constraints to the transaction postconditions [Sto75]. Let us assume for a moment that the invariant had the form $\forall c/Course\ size(c) \leq classlimit(c) \wedge \psi$, where ψ does not mention predicate *EnrolledIn* or function *size*. Then, under the assumption that the invariant was satisfied before the transaction execution, the conjunct ψ can be eliminated since its satisfaction

[1] Fluents are predicates whose truth value may change from state to state [McC69].

[2] We will abandon this notation when we present the formal solution to the frame and ramification problem. Instead, we will use a many-sorted logic and factor time into predicates and functions.

persists. The augmentation to the postcondition that suffices to guarantee the constraint is the same as above, i.e., $(size(crs) + 1 \leq classlimit(crs))$. A non-optimizing transaction modification technique, such as, e.g., in [Sto75], would include ψ as a condition that would need to be conjoined with the transaction postcondition. $\Box$

We review the concept of *ramifications* in section 2. We describe briefly the solution to the frame and ramification problems proposed in [LR92], as well as an extension to the procedure proposed in [BMR93] and [Pin94] for systematically solving the problem in the case of determinate transaction specifications. Section 3 presents the ramification method for transaction modification along with several examples of the application of the method on static and dynamic constraints. The results are extended in section 4 to deal with multiple transactions, conjunction and inheritance of specifications. Section 5 reviews related work and discusses the integration of the proposed method in the database design process. The final section summarizes the results and outlines directions for further research.

2 Ramifications

In this section we provide the necessary background for the ramification method for transaction modification. In the first part we review the concept and properties of ramifications and, in the second, we describe the solution to the ramification problem for the class of deterministic transaction specifications. The bulk of the background material is based on [Fin88] and [Pin94] but is recast here in database terminology. Moreover, the results presented therein are extended as propositions 2.1 - 2.3 describe.

2.1 Background

Let us assume that a knowledge base KB comprises a set of ground literals representing facts true in KB, a set R of deductive rules and a set I of integrity constraints. Constraints and rules are expressed as range-restricted formulae of a many-sorted first-order logic in which we distinguish one sort *Time* for time points. We will refer to this logic as MSTL for short. Time is interpreted as being relative, linear and discrete.

Intuitively, a *ramification* of a formula ϕ is a formula N such that, N is inevitably true if ϕ is true [Fin88]. This definition is amenable to different interpretations in different world models. If the world model in question - the knowledge base - is expressed as a first-order theory, then the concept of ramifications can be captured by first-order entailment. A formula ϕ is said to be in prenex normal form if it is of the form $\phi \equiv Q_1 x_1/S_1, \ldots, Q_m x_m/S_m\ \hat{\phi}$, where $Q_i \in \{\forall, \exists\}, i = 1, \ldots, m$, $S_1, \ldots, S_m$ are sorts and $\hat{\phi}$ is quantifier-free with free variables $x_1, \ldots, x_m$.

Definition 2.1 (*Ramification)* Given a knowledge base KB and a formula ϕ in prenex normal form, a formula N with free variables $y_1, \ldots, y_n$ among $x_1, \ldots, x_m$ is a ramification of ϕ in KB, if $KB \cup \{\phi\} \models \forall y_1, \ldots, \forall y_n N$.

Example 2.1 Consider the knowledge base[3] $KB = \{\forall x \forall y \forall z/Dom(A(x) \wedge C(y) \rightarrow D(x, y, z)\}$ and the formula $\phi \equiv \exists x \exists y(A(x) \wedge B(x, y) \wedge C(y))$. Then, the formula $N \equiv \forall z\ D(x^*, y^*, z)$ is a ramification of ϕ in KB since $KB \cup \{\phi\} \models N$. □

It is easy to verify the following properties of ramifications:

1. If a ramification of a formula is known to be unsatisfiable, then the formula itself is unsatisfiable
2. Ramifications of formulae can reduce the search space for the formulae satisfaction
3. Transformations may be applicable to formulae ramifications but not to the formulae themselves

Hence, if there exists a way to systematically generate ramifications from a set of formulae, then the derived ramifications can be used for optimizing the evaluation of the formulae themselves.

Generating ramifications of a formula ϕ essentially involves augmenting a partial description of a state where ϕ holds with additional descriptions that stem from the knowledge of the consistency of the state. A procedure for generating ramifications may require an arbitrary amount of inferencing. From the semi-decidability of first-order entailment, it follows that the problem of finding ramifications is, in its generality, intractable. Tractability can be achieved by restricting the class of formulae for which ramifications are sought. For instance, the task is tractable for ordered conjunctions of literals [Fin88]. Furthermore, not all derivable ramifications may be useful for simplifying the task of proving a formula. For that, the generator may be guided to derive only "useful" ramifications by providing appropriate input clauses. In fact, the solution to the

[3] For the sake of simplicity we assume that all variables range over the same sort Dom. Moreover, *-ed identifiers will denote Skolem constants.

ramification problem that we present in section 2.2 generates exactly those ramifications that are needed for proving the safety of transaction specifications.

The process of modifying a goal by conjoining it with additional constraints so that the reformulated formula is less expensive to check than the original one, has been termed *supersumption* in [Fin88]. The methods proposed therein apply only to conjunctive formulae. It is easy to establish the soundness of supersumption for formulae in disjunctive normal form (DNF). Proposition 2.1 is a direct consequence of the definition of ramifications and proposition 2.2 follows from the properties of first-order entailment.

Proposition 2.1 If N is a ramification of ϕ in the knowledge base KB, and N is falsified in KB, then so is ϕ.

Proposition 2.2 Let ϕ be a formula in DNF, i.e. $\phi \equiv \bigvee_{i=1}^{n} \phi_i$, where each ϕ_i is a conjunction of literals. Let also N_i be a ramification of ϕ_i for each $i = 1, \ldots, n$ and $KB \models (\phi_i \Rightarrow N_i)$. Then, if for each $i = 1, \ldots, n$ $KB \models \neg N_i$, then $KB \models \neg\phi$.

According to proposition 2.2, given a formula in DNF, one can derive ramifications of the conjuncts of the formula and then test whether all ramifications are falsified. If this is the case, then the initial formula is falsified. This strategy is useful if one is interested in monitoring when a formula becomes violated rather than proving that it is always satisfied. As will be seen in the next section, this strategy is sufficient to monitor integrity constraint violations in KBs. The next corollary establishes the soundness of supersumption.

Corollary 2.1 (*Soundness of Supersumption)* Let ϕ be the formula $Q_1 x_1, \ldots, Q_m x_m \hat{\phi}, Q_i \in \{\forall, \exists\}$, $i = 1, \ldots, m$, where $\hat{\phi}$ is quantifier-free in DNF with each of the variables $x_1, \ldots, x_m$, appearing in at least one of the disjuncts, and let N be a ramification of ϕ in the knowledge base KB. Then $KB \cup \{\phi\} \models \hat{\phi} \wedge N$.

2.2 The Frame and Ramification Problems

In this section we sketch the solution to the frame and ramification problems for a class of constraints that encompasses static and transition constraints. The solution to the frame problem was initially proposed in [Rei91] in the framework of situation calculus [McC69]. It was extended in [LR94] and [Pin94] for dealing with the ramification problem as well. The solution relies on the automatic generation of complete characterizations of the conditions under which predicates or functions may change (truth) value as a result of transaction execution. Here we present an extension to the method of [Pin94] for dealing with transition constraints.

The method generates *successor-state axioms* from a given set of *effect axioms*, in the presence of a limited class of constraints and definitions of non-primitive predicates. Effect axioms specify the direct effects of transactions on predicates. For instance, a direct effect of transaction *EnrollInCourse* (see example 1.1), is that the size of the course in which a student enrolls is incremented by 1. Successor-state axioms characterize all conditions under which predicates and functions may change value as a result of transaction execution. Such axioms serve as a formalization of a closed-world assumption about the transactions themselves rather than the knowledge base.

Integrity constraints are of the form:

$\forall x_1/S_1, \ldots, x_k/S_k, \forall t_1, t_2/Time\ \phi(x_1, \ldots, x_k, t_1, t_2) \vee (\neg)p_1(x_1, \ldots, x_k, t_1) \vee (\neg)p_2(x_1, \ldots, x_k, t_2)$

where, p_1, p_2 are $(k+1)$-ary predicates, intensional or extensional, $S_1, \ldots, S_k$ are object sorts and $\phi(x_1, \ldots, x_k, t_1, t_2)$ is a formula in which variables $x_1, \ldots, x_k, t_1, t_2$ occur free, if at all, and does not mention any predicate other than evaluable predicates. This class of constraints is an extension of the class of *binary constraints* of [Pin94], since it allows the temporal variables to occur in evaluable predicates. It includes static constraints and transition constraints, i.e., constraints referring to two consecutive states of the knowledge base, but not general dynamic constraints. For example, the transition constraint specifying the property that an employee's salary can never decrease, can be specified by the formula

$\forall e/Employee\ \forall s_1, s_2/Salary\ \forall t_1, t_2/Time\ (s_1 < s_2) \vee \neg(t_1 < t_2) \vee \neg salary(e, s_1, t_1) \vee \neg salary(e, s_2, t_2)$.

A transaction T with parameters $\overline{x}$ is specified by a pair $(pre_T(\overline{x}), post_T(\overline{x}))$, where $pre_T(\overline{x})$, and $post_T(\overline{x}))$ denote the transaction pre- and post-condition respectively, both specified as well-formed formulae of MSTL.

The solution assumes that deductive rules are not recursive. Moreover, they are assumed to be stratified. It also presupposes the existence of causal rules describing the direct effects of transactions. These causal rules are expressed by *direct effect axioms* which, for a transaction $T(\overline{x}) = (pre_T(\overline{x}), post_T(\overline{x}))$ have the form:

$$\forall \overline{x}/\overline{S}\ \forall t/Time\ (Occur(T(\overline{x}), t) \Rightarrow pre_T(\overline{x}, t) \wedge post_T(\overline{x}, next(t)))$$

where the term $next(t)$ denotes the state resulting from the execution of the transaction at time t. Given any transaction specification, the effect axioms are derived independently of any other transaction specification. Hence, we can avoid having to specify the axioms in a reified logic that interprets the predicate *Occur* outside a standard first-order interpretation. The above axiom can now be written as:

$$\forall \overline{x}/\overline{S}\ \forall t/Time\ T(\overline{x},t) \Rightarrow pre_T(\overline{x},t) \wedge post_T(\overline{x},next(t)).$$

From the direct effect axioms we can systematically generate *positive* and *negative* effect axioms [BMR95] for every predicate P that occurs in $post_T$, as described in the following steps.[4] The rationale is to describe concisely all conditions that are necessarily true when a predicate changes value from *False* to *True* (*True* to *False* respectively).

1. Construct the following positive and negative axioms:
$$\forall \overline{x}/\overline{S}\ \forall t/Time\ (\neg P(\overline{x},t) \wedge P(\overline{x},next(t))) \wedge T(\overline{x},t) \Rightarrow False)$$
$$\forall \overline{x}/\overline{S}\ \forall t/Time\ (P(\overline{x},t) \wedge \neg P(\overline{x},next(t))) \wedge T(\overline{x},t) \Rightarrow False)$$

2. If $post_T$ is $P(\overline{x},next(t))$ $(\neg P(\overline{x},next(t)))$, add $True$ as a disjunct to the positive (negative) effect axiom for P.

3. If $post_T$ is of the form $\gamma(\overline{x},t) \Rightarrow (\neg)P(\overline{x},t)$, where γ does not contain terms referring to any time point except t, add a disjunct $\gamma(\overline{x},t)$ to the positive (negative) effect axiom for P.

4. If $post_T(\overline{x})$ is of the form $\exists \overline{z}\ (\gamma(\overline{x},\overline{z},t) \Rightarrow (\neg)P(\overline{w},next(t))$, where $\overline{w}$ consists of constants and variables from $\overline{x},\overline{z}$, then augment the positive (negative) axiom for P with a disjunct $\exists \overline{z}\ (\gamma(\overline{x},\overline{z},t) \wedge (\overline{x} = \overline{w}))$.

This process results in a set T_{ef} of effect axioms of the form:

$$\forall \overline{x}/\overline{S}\ \forall t/Time\ \neg P(\overline{x},t) \wedge \neg \Phi_{1P}(\overline{x},t) \Rightarrow \neg P(\overline{x},next(t)) \quad (1)$$
$$\forall \overline{x}/\overline{S}\ \forall t/Time\quad P(\overline{x},t) \wedge \neg \Phi_{2P}(\overline{x},t) \Rightarrow P(\overline{x},next(t))) \quad (2)$$

These axioms concisely describe how transactions directly affect the truth values of predicates. It remains to describe the indirect effects that are due to the presence of integrity constraints and deductive rules.

In addition to the effect axioms T_{ef}, the knowledge base is augmented with an axiomatization of time (T_{time}) formalizing the properties of discreteness and unboundedness, as well as unique name axioms (T_{una}) for predicates and functions. A new set T'_{ef} is obtained from T_{ef} by replacing each derived predicate occurring in an effect axiom by the disjunction of the bodies of the rules that define it. Since the set of deductive rules is assumed to be stratified, the process of replacing derived

[4] We only show the derivation of effect axioms for predicates. Effect axioms for functions can be derived quite similarly.

predicates by their definitions will terminate with all effect axioms mentioning only primitive predicates.

Let $C \equiv \forall \overline{x}/\overline{S}, \forall t_1, t_2/Time\ \phi(\overline{x}, t_1, t_2) \vee P(\overline{x}, t_1) \vee Q(\overline{x}, t_2)$ be a constraint that has to be satisfied at all times. Then, for each effect axiom of type (1) for P, the following axiom for Q is generated:

$$\forall \overline{x}/\overline{S}\ \forall t/Time\ \neg P(\overline{x}, t) \wedge \neg \Phi_{1P}(\overline{x}, t) \wedge \neg \phi(\overline{x}, t, next(t)) \Rightarrow Q(\overline{x}, next(t))$$

This axiom expresses the property that, if predicate P is known not to be true in the state prior to the execution of a transaction and constraint C is known to be satisfied in the same state, then, if the conditions that cause P to change truth value from $False$ to $True$ are not satisfied, $Q(\overline{x}, next(t)) \vee \phi(\overline{x}, t, next(t))$ has to be true in order for the constraint to remain satisfied in the state after the transaction execution.

Symmetrically, for each effect axiom of type (1) for Q, generate the following axiom for P:

$$\forall \overline{x}/\overline{S}\ \forall t/Time\ \neg Q(\overline{x}, t) \wedge \neg \Phi_{1Q}(\overline{x}, t) \wedge \neg \phi(\overline{x}, t, next(t)) \Rightarrow P(\overline{x}, next(t))$$

The respective process takes place if the constraint contains negated predicates. Then, under the assumption that the given specifications characterize all transactions, we can generate the set T_{ss} of *successor-state axioms* as follows: Let $\Psi_P(\overline{x}, t) = \neg \Phi_{1Q}(\overline{x}, t) \wedge \neg \phi(\overline{x}, t, next(t))$ and $\Psi_{\neg P}(\overline{x}, t) = \neg \Phi_{2Q}(\overline{x}, t) \wedge \neg \phi(\overline{x}, t, next(t))$. $\Psi_Q(\overline{x})$ and $\Psi_{\neg Q}(\overline{x})$ are defined analogously. Then, the successor-state axiom for P is:

$$\forall \overline{x}/\overline{S}\ \forall t/Time\ \Psi_P(\overline{x}, t) \vee (\neg \Psi_{\neg P}(\overline{x}, t) \wedge P(\overline{x}, t))$$

Example 2.2 Example 1.1 showed the definition of transaction *EnrollInCourse* which affects the predicate *EnrolledIn* and the function *size*. Let us assume that another transaction, *DropCourse* is defined as follows:

DropCourse(st, crs) `Precondition:` $EnrolledIn(st, crs)$ `Postcondition:` $size'(crs) = size(crs) - 1\ \wedge\ \neg EnrolledIn'(st, crs)$
`Invariant:` $\forall c/Course\ size(c) \leq classlimit(c)$

The generation process described in this section will generate the following successor-state axioms for *EnrolledIn* and *size*:

$\forall st/Student\ \forall crs/Course\ \forall t/Time$

$\neg EnrolledIn(st, crs, t)\ \vee\ EnrolledIn(st, crs, next(t))\ \vee\ DropCourse(st, crs, t)$

$\forall crs/Course\ \forall t/Time\ (size(crs, t) = size(crs, next(t)))\ \vee\ \exists st\ EnrollInCourse(st, crs, t)\ \vee$

$\exists st \; DropCourse(st, crs, t)$

Under the assumption that the transactions *EnrollInCourse* and *DropCourse* are the only ones affecting *EnrolledIn* and *size*, the above axioms characterize all conditions under which the predicate or function can change (truth) value as a result of transaction execution. Specifically, the first axiom expresses the property that if a student is enrolled in a course in the state prior to a transaction's execution but is not enrolled in the course after the transaction's execution, then it has to be the case that transaction *DropCourse* occurred and it cannot be the case that any other transaction may have occurred. The second axiom says that, if function *size* changes value because of transaction execution, then it is the case that either transaction *EnrollInCourse* occurred or transaction *DropCourse* occurred. □

It has been shown in [LR94] that a set, T_{ss}, of successor-state axioms constitutes a solution to the frame and ramification problem if, for every predicate p, the condition $T_{una} \models \neg(\Psi_p \wedge \Psi_{\neg p})$ is satisfied. Moreover, the correctness of the syntactic generation has been proven in [LR94] and [Pin94]. We are now in a position to state the relationship of the syntactic generation of successor-state axioms with ramifications of constraints. A similar result holds for the case in which predicates occur negated in the expression of a constraint. The proposition follows from the syntactic generation process.

Proposition 2.3 For a knowledge base KB and a constraint

$$I \equiv \forall \overline{x}/\overline{S}, \forall t_1, t_2/Time \; \phi(\overline{x}, t_1, t_2) \vee P(\overline{x}, t_1) \vee Q(\overline{x}, t_1, t_2)$$

the following entailment relation holds:

$$KB \cup \{I\} \models [(\neg\Phi_{1P}(\overline{x}, t) \wedge \neg\phi(\overline{x}, t, next(t))) \;\vee\; (\neg\Phi_{1Q}(\overline{x}, t) \wedge \neg\phi(\overline{x}, t, next(t)))]$$

The above result is significant to the problem of proving transaction safety, since it provides a way to produce systematically necessary conditions (ramifications) for the satisfaction of constraints in the state resulting from the transaction execution.

3 Integrity Maintenance by Transaction Modification

In this section we establish the relationship between the ramification problem and the maintenance of integrity constraints in the context of knowledge bases containing deductive rules and temporal knowledge.

3.1 Ramifications and Integrity Maintenance

The problem of integrity maintenance is defined as follows: Given a knowledge base KB with constraint set I and a set of transaction specifications $T = \{T_1, \ldots, T_k\}$ with $T_i = (pre_i, post_i)$, can the set I be systematically partitioned into sets I_t, I_f, I_c so that: (a) constraints in I_t are *provably* maintained by T, (b) constraints in I_f are *provably* violated by some $T' \subseteq T$, and (c) constraints in I_c have to be checked after execution of some transactions in T but possibly in some simplified form? Since we are following a transaction modification approach, the problem is equivalent to transforming the set, T, of transactions into a set, T', of transactions with the property that, for each transaction T'_i in T', either $T'_i = T_i$ and T_i has been shown not to violate any of the integrity constraints, or T_i has been modified to T'_i and every implementation meeting its new specification cannot possibly violate any of the constraints in I.

Integrity constraints serve the role of invariants of transactions. Each transaction must maintain its invariants, i.e. not violate the relevant integrity constraints in order to be accepted. The problem of proving that a transaction maintains its invariants is formalized in the following definition.

Definition 3.1 (*Invariant Maintenance)* Let T be a transaction with precondition P, postcondition Q and invariant I. T is said to maintain invariant I, if $I \wedge P \Rightarrow (Q \Rightarrow I')$. where I' denotes the invariant in the state resulting after the transaction takes place.

Proving that transactions maintain invariants is a difficult task since it requires theorem proving. Furthermore, the cost of undoing the transaction in case it is discovered to violate the invariants is high. A way to avoid checking whether transactions maintain their invariants is to augment their postconditions in a way such that the invariant is maintained as a result of meeting the postcondition.

We now show that such an augmented postcondition can be found by computing ramifications. The following discussion is based on a number of completeness assumptions. These assumptions specify that, firstly, all transaction specifications are known at the time that integrity constraints are specified and, secondly, that transaction invariants are known to be maintained in the state before the transaction execution. We discuss in the sequel how we can incrementally accommodate newly defined transactions.

We consider the single-transaction case first. Assume a transaction T specified by a pair (P, Q) of a precondition and a postcondition expressed in MSTL. Let I be an integrity constraint relevant to the transaction[5]. We need to find a formula N such that $KB \models (Q \wedge N \Rightarrow I')$, or equivalently that, $KB \models (Q \wedge \neg I' \Rightarrow \neg N)$. If $\neg N$ is a ramification of $Q \wedge \neg I'$ as computed by the syntactic generator, then the desired entailment relationship holds. This leads to the following theorem, whose proof follows from definitions 2.1, 3.1 and corollary 2.1:

Theorem 3.1 Let T be a transaction specified in terms of a precondition/postcondition pair (P, Q) and I be an invariant of T. If N is a ramification of $Q \wedge \neg I'$ computed by the syntactic generator of section 2.2, then the invariant is maintained in the state resulting after transaction execution if the postcondition $Q \wedge \neg N$ is met.

This result has significant impact to both the areas of procedure specification and constraint enforcement: if the process of suggesting additions to the postconditions of transactions can be automated, the transaction specifier actually realizes the implications of transaction invariants and the procedure implementor is saved the burden of finding ways to meet the postcondition in a way such that no invariant is violated. In fact, the implementor may not be familiar with all the invariants that a certain transaction may affect. The theorem also suggests a way of enforcing integrity constraints by requiring that updating transactions meet postconditions that embody implications of constraints: first, ramifications of the conjunction of the postcondition and the negation of the invariant instantiated in the state after transaction execution are computed; the negation of the computed ramifications is conjoined with the transaction postcondition to form a postcondition which should be met by the implementation in order not to violate the invariants. The fact that postconditions describe all the direct effects of transactions can be exploited to simplify the formula of which ramifications are sought. Specifically, (truth) values of predicates or functions changed by the transaction can be assumed to be known in the state resulting from transaction execution. Hence, the (truth) values can be substituted for the predicates or functions and logical simplifications may be applicable. The invariants themselves need not be verified in the state resulting from the transaction execution since their satisfaction is guaranteed by the transformation process. Moreover, the ramifications generated may be simpler formulae than the invariants and hence, the transformation of postconditions can incur considerable savings in testing for the satisfaction of invariants. An exact assessment of the run-time complexity of testing the satisfaction of augmented postcondition for the class of transaction specifications considered here is a topic of current research.

[5] The notion of "relevance" is defined formally in the sequel.

The following examples show the merits and problems associated with the technique of incorporating ramifications into transaction specifications.

Example 3.1 Transaction *EnrollInCourse* was defined in example 1.1. In this example, we show the derivation of ramifications for postcondition augmentation, by rewriting the postcondition and invariant in MSTL. We first construct the conjunction of the postcondition Q and the negation of the invariant instantiated as follows:

$$\neg I' \equiv \neg(size(crs, t+1) \leq classlimit(crs))$$

$$\neg I' \wedge Q \equiv \neg(size(crs, t+1) \leq classlimit(crs)) \ \wedge \ (size(crs, t+1) = size(crs, t) + 1) \ \wedge$$
$$EnrolledIn(st, crs, t+1)$$

By substituting $True$ for $EnrolledIn(st, crs, t+1)$ and $size(crs, t) + 1$ for $size(crs, t+1)$, the conjunction becomes $\neg(size(crs, t) + 1 \leq classlimit(crs))$. This formula is in fact a ramification of $\neg I' \wedge Q$ and can be computed as shown in section 2.2. According to theorem 3.1, it suffices to conjoin the negation of the ramification to the potscondition. The invariant is no longer needed for verifying the safety of transaction *EnrollInCourse*, since the invariant is embodied in the new postcondition. The augmented transaction specification now becomes :

EnrollInCourse (st, crs) `Precondition:` $\neg EnrolledIn(st, crs)$ `Postcondition:` $size'(crs) = size(crs) + 1 \ \wedge \ EnrolledIn'(st, crs) \ \wedge$ $(size(crs, t) + 1 \leq classlimit(crs))$

□

The next example shows that certain ramifications can suggest that invariants do not have to be checked and postconditions need not be modified in order to guarantee the invariants.

Example 3.2 (*Special cases*) The specification of transaction *DropCourse* was given in example 2.2. Intuitively, the invariant cannot be violated as a result of executing *DropCourse*, and hence the specification of the transaction need not be modified. In fact, the ramification generation process produces the Boolean constant $False$ as a ramification of the negation of the conjunction formed by instantiating and negating the invariant in the state after the transaction execution and by conjoining it with the transaction postcondition. According to theorem 3.1, it suffices to augment the postcondition with the negation of the derived ramification, i.e., the Boolean constant $True$. This means that it suffices for the implementation meet the transaction postcondition as was initially specified, in order to maintain the invariant. □

The case in which the propositional constant $False$ is derived as a ramification is of particular interest since, as the following corollary specifies, no change in the postcondition is needed in order to meet the invariant. The derivation of $True$ as a ramification is only possible when the transaction specification is inconsistent.

Corollary 3.1 If $False$ is a ramification of $Q \wedge \neg I'$, then I' is maintained by a transaction meeting Q. If $True$ is a ramification of $Q \wedge \neg I'$, then the transaction specification is inconsistent.

Hence, the process of generating ramifications can also discover inconsistent specifications of transactions that may have escaped the specifier's attention.

A valid question that arises is whether a similar approach where preconditions rather than postconditions of transaction specifications can be augmented so that the maintenance of the invariants is a result of the satisfaction of the transaction precondition. In general, the two approaches are not equivalent. They are equivalent only in the case where derived ramifications refer only to the state before the transaction execution. The following section shows an example of a transaction specification where the derived ramification refers to the state after the transaction's execution. It is, thus, unnatural to augment the postcondition with a condition that refers to the state resulting from the transaction execution.

3.2 Dynamic Integrity Constraints

We would like to be able to propose similar augmentations to postconditions when the invariant refers to any number of knowledge base states, both before and after the transaction. In other words, we need to extend the method for the enforcement of dynamic integrity constraints. The solution to the ramification problem presented in section 2.2 does not deal with constraints more general than transition constraints. It is applicable in the cases where the checking of conditions over multiple consecutive states can be reduced to checking conditions over pairs of consecutive states. The extension of the method to general dynamic constraints is a topic of current research. Some initial results are given through examples of the use of ramifications for transactions that involve transition and general dynamic constraints. These examples also motivate the use of a temporal calculus for expressing transaction specifications.

Example 3.3 (*Transition constraints as transaction invariants*) Transaction *RaiseSalary* assigns an employee an increase to her salary.

RaiseSalary (emp, new_sal) **Precondition:** $\exists old\ salary(emp, old, T_b)$ **Postcondition:** $salary(emp, new\_sal, T_a)$
Invariant: $\forall e, s, s'/D\ \forall t, t'/T\ [salary(e, s, t) \wedge salary(e, s', t') \wedge (t \leq t') \Rightarrow (s \leq s')$

T_b and T_a are used to denote time points before and after the transaction respectively. They are parameters whose exact values are not known at transaction specification time[6]. For the purpose of deriving the successor state axioms, one needs to instantiate the time component of predicates referring to the state prior to the transaction with T_b. The time component of predicates referring to the state after the transaction are instantiated with T_a. The ramification derived is: $\neg(old^* > new\_sal)$. Its addition to the postcondition suffices to ensure the maintenance of the invariant. □

For the sake of demonstrating the applicability of using ramifications with constraints strictly more general than the ones dealt with so far, we now switch to using first-order temporal logic (FOTL) [MP91] as the specification language.[7] FOTL allows one to express constraints referring to an arbitrary number of states. We need to assume, without loss of generality, that exactly one action can occur between two successive states of the knowledge base. The following example also demonstrates why it is unnatural to consider augmentations of the precondition of a transaction specification in order to achieve the maintenance of invariants.

Example 3.4 (*General dynamic integrity constraints*) The formula expressing the property "*If P(x), then sometime in the future Q(x)*" is an invariant for transaction *InsertP* that inserts a tuple (x, t) in the extension of base predicate P.

InsertP (x,t) **Precondition:** $True$ **Postcondition:** $P(x, t)$
Invariant: $\forall x/D\ \forall t/Time\ [P(x, t) \Rightarrow \exists t'/Time\ (t' > t\ \wedge\ Q(x, t'))]$

In addition, assume that the knowledge base includes the following deductive rules:

R_1: $\forall x/D\ \forall t/Time\ P(x, t)\ \rightarrow R(x, t+1)$

R_2: $\forall x/D\ \forall t/Time\ R(x, t)\ \rightarrow Q(x, t)$

[6] These time points are not unique. It suffices that T_a is a time point at which the constraints have to be verified (before the transaction commits) and that T_b a time point before the transaction begins execution and at which it is known that the KB is in a consistent state.

[7] Similar results can be obtained when transactions are specified in a FOTL with modal operators $\bigcirc$, $\Box$ and $\Diamond$.

Intuitively, after the transaction *InsertP* finishes execution, the knowledge base is in a consistent state, since the constraint is satisfied due to the implicit updates. Because of the presence of the rules, no precondition exists that will guarantee the invariant. The invariant however contributes to effects (ramifications) that can be used to eliminate the need for proving the invariant. We generate the ramifications by first instantiating and negating the invariant and then computing its logical consequences given the knowledge base. The negated invariant is $\neg I' \equiv \forall t'[P(x,t) \wedge (Q(x,t') \Rightarrow (t' \leq t))]$. From the postcondition and rule R_1 we derive $R(x,t+1)$. Using rule R_2, we can now derive $Q(x,t+1)$. Using the negated invariant and the postcondition we derive the ramification $N \equiv Q(x,t+1) \Rightarrow False \ \equiv \neg Q(x,t+1) \equiv False$. Hence, the invariant is maintained if the postcondition is met. Notice that it is unreasonable to include $Q(x,t+1)$ as a precondition to a transaction of which it is an implicit consequence. The constant $False$ is generated as a ramification by the syntactic generator by replacing the derived predicates that occur in $\neg I' \wedge Q$ by their definitions and then applying the steps described in section 2.2. □

Albeit artificial, example 3.4 shows that dynamic constraints can have useful effects for transactions. Even in cases where the satisfaction of a constraint cannot be determined because the constraint refers to the yet undetermined future, the method yields a ramification that can be used in place of the original constraint. This is the case in example 3.4 if the rules are omitted. In this case, we cannot determine whether the constraint is satisfied. or violated, since it refers to the possibly infinite set of subsequent states. In this case, the method can propose a simpler condition, that is actually a ramification of the original constraint and the postcondition. The formula $N \equiv \neg\forall t' \ (t' \leq t \vee \neg Q(x,t'))$ is derived and the new constraint that suffices to be verified is $\neg N \equiv \exists t' \ (t' > t \ \wedge \ Q(x,t'))$. It contains the property that has to be verified by the future states and can be treated as the original constraint would.

4 Extensions

In this section we present extensions of the ramification method for dealing with multiple constraints, multiple transactions, conjoining transaction specifications, and inheritance of specifications.

4.1 Multiple Transactions

The approach followed in the previous section is certainly applicable to the problem of reasoning about the set of constraints of a knowledge base in the presence of multiple transaction specifications. In this case all transaction specifications have to be taken into account since a constraint may be relevant to, or affected by, more than one transactions. The notion of relevance is formally defined here. It is based on the notion of *dependence*, taken from [Ple93a]. For that we will assume that the specifications are given in first-order temporal logic. Moreover, we will assume that no interleaving of transactions is allowed. A transaction is regarded as the only means of knowledge base state change.

Definition 4.1 (*Relevance)* A constraint I is relevant to a transaction $T = (pre, post)$ if *post* contains a literal on which some literal of I depends.

Definition 4.2 (*Direct Dependence)* A literal L directly depends on a literal K if and only if there exists a rule of the form $\forall x_1/\mathtt{C}_1 \ldots \forall x_n/\mathtt{C}_n\ (F \Rightarrow A)$ such that, there exists a literal in the body F of the rule unifying with K with mgu θ and $A\theta = L$. *(Dependence)* A literal L depends on literal K if and only if it directly depends on K, or depends on a literal M that directly depends on K. A constraint/rule depends on a rule if its literal depends on the rule's conclusion literal.

The above relationships define a *dependence graph* for a set of rules and constraints. The dependence graph is a directed graph representing how implicitly derived facts from deductive rules can affect the integrity of the knowledge base. The graph can be constructed at transaction specification time and, as shown in [Ple93a], is incrementally modifiable to accommodate changes in the sets of rules or constraints.

An integrity constraint relevant to a set of transactions $\{T_i = (pre_i, post_i) | i = 1, \ldots, k\}$ has to be considered for the modification of each $post_i$, so that the execution of any T_i provably maintains the constraint. Hence, it suffices to repeat the process presented in section 3 for every T_i. The process may be optimized by reusing the derivation of ramifications for transactions that involve common predicates. The symmetric case, where a transaction specification is associated with more than one invariants, is dealt with by simply taking the conjunction of the invariants as the new invariant. Then the derived ramification depends on all invariants, provided that the union of the invariants is a satisfiable set.

4.2 Conjoining Transaction Specifications

The conjunction of specifications - denoted by the operator $||$ - is formed by conjoining the respective pre/post-conditions. Then, as theorem 4.1 suggests, it suffices to conjoin the ramifications of the two invariant-postcondition pairs, to guarantee that the invariant will be maintained if the new postcondition is met.

Theorem 4.1 Let $T_1 = (pre_1, post_1)$ and $T_2 = (pre_2, post_2)$ be two transaction specifications sharing invariant I. If there exist ramifications N_1 and N_2 which, if conjoined with the postconditions $post_1$ and $post_2$ guarantee the maintenance of I in T_1 and T_2 respectively, then $N_1 \wedge N_2$ is a ramification which, if conjoined with $post_1 \wedge post_2$ guarantees the maintenance of I in $T = T_1||T_2$.

An important consequence of theorem 4.1 is the ability to accommodate new invariants without having to redo the entire process. Specifically, if a new invariant is to be added and is relevant to a transaction specification whose postcondition has already been augmented by computed ramifications, it suffices to verify that the new invariant does not introduce any contradiction, and, if this is the case, to generate ramifications of the new invariant and the postcondition. The new ramification can be conjoined with the previously derived ones, so that the new postcondition guarantees the invariants.

4.3 Inheritance of Transaction Specifications

In object-oriented specification languages, inheritance of transaction (*method*) specifications is traditionally accomplished by conjoining the superclass' method specification to that of its subclasses [SP87]. We examine whether ramifications derived for the superclass can be inherited by the subclasses.

Assume a transaction $T_2 = (pre_2, post_2)$ is a specialization of $T_1 = (pre_1, post_1)$ and that there exists a formula N_1 with the property $KB \models (post_1 \wedge N_1) \Rightarrow I$. The specification of T_1 is inherited by T_2. It is the responsibility of the specifier to ensure that neither of the conjunctions $(pre_1 \wedge pre_2)$ and $(post_1 \wedge post_2)$ is a contradiction. Then, according to theorem 4.1, if a ramification N_2 can be found, such that $KB \models (post_2 \wedge N_2) \Rightarrow I$, then augmenting $post_2$ with $N_1 \wedge N_2$ suffices to guarantee that the invariant will be maintained if the augmented refined postcondition is met.

5 Discussion

In the majority of the existing methods for integrity constraint maintenance by transaction modification, the frame and ramification problems have either been ignored or bypassed by means of implicit assumptions that state that "nothing but what is explicitly declared to change in the update procedure does". Within AI however, the problems have long been recognized as common sense reasoning problems and several attempts towards their solution have appeared (e.g., [Lif91], [Sch90]). The solution presented by Reiter [Rei91], combined and extended previous results, leading to the systematic solution on which this paper is based.

In [Sto75], integrity constraints expressed as QUEL queries on relational databases are added as qualifications to transactions. The execution of the resulting transaction guarantees integrity preservation. However, no simplification of constraints takes place and the knowledge of their satisfaction prior to the update is not exploited. A set-oriented language for transaction specification is used in [LTW93]. For each update and each constraint, a *weakest precondition* (wp) is derived so that, if wp is true in the state prior to the update, then the constraint is guaranteed to be true in the state resulting from the update. Although the derived weakest preconditions are frequently amenable to simplification, there is no systematic treatment of precondition optimization. The method is applicable to a limited class of static constraints only and there is no mention of derivations of weakest preconditions when multiple constraints are relevant to an update. Finally, although the authors claim that it is trivial to incorporate implicit updates, the machinery provided does not account for them. constraints In [SS89], a general-purpose theorem prover employing heuristic rewrite rules is used for proving safety of transactions with respect to a set of static constraints. Implicit updates and dynamic constraints are not considered. Although safety of transactions is proved at compile-time, the theorem prover could take advantage of the knowledge of the updates taking part in the transaction in order to simplify the proof procedure and possibly suggest changes to the transaction specification. Although not discussed, the frame problem is implicitly dealt with by restricting attention to the predicates changed by transactions and by eliminating inertial terms from the theorems that have to be proven in order to verify integrity. The concept of *constraint protectors* discussed in [SMS87] resembles that of constraint ramifications. Their generation however assumes the existence of a fairly general theory of lemmas that is independent of the transactions. Several modes of feedback to the transaction designer are suggested but are not put into a formal framework.

Dynamic integrity constraints specified in temporal logic are translated into transaction specifications in [Lip90]. Constraints are translated into *transition graphs* and transformation steps that simulate the evaluation of the constraints on the graphs are applied. Transactions are specified in terms of pre/post-condition pairs and explicit frame assumptions. The transaction specifier has to supply all frame conditions explicitly. The specifications are the modified by incorporating conditions that represent the parts of the constraints that remain to be verified. Although, the transformation technique is sound, i.e., violations of constraints are detected, the transformations to transaction specifications introduce conditions that refer to the transition graphs instead of predicates occurring in constraints or transactions. It It becomes unclear what the specifications of the transaction mean with respect to the database state. Moreover, the problem of implicit updates is not addressed. In [CW90], integrity constraints are semi-automatically translated into set-oriented production rules activated at the commit point of transactions. The focus of this work is on automatic integrity repair rather than optimization of constraint checking. In [CT94], active rules are used to implement temporal integrity constraints, but the problem of implicit updates is not addressed.

The previous sections showed how the task of proving integrity maintenance can be assisted by the adaptation of a systematic solution to the frame and ramification problems. The method presented is applicable to a fairly large class of transaction specifications, namely that of determinate specifications, and a class of constraints that encompasses the types of constraints that are usually supported by commercial database systems and most research prototypes. Furthermore, the results extend to the case of object-oriented specifications and for the synthesis of specifications by conjoining existing specifications.

The process of designing a knowledge base can be assisted by a tool that, when given a set of transaction specifications, suggests modifications to the postconditions so that the integrity constraints are provably maintained by any implementation meeting the modified specification. Moreover, inconsistencies in the specifications can be discovered. We argue that this form of feedback is crucial for the knowledge base design process, since it provides with a systematic way of testing whether certain desirable aspects of a 'good' design can be achieved.

It has to be noted that the results presented here are not tied to a particular specification language. A similar generation process can be devised for specifications given in language like SQL or its extensions.

6 Conclusions

The contribution of the research presented in this paper is two-fold: it shows that the use of a solution to the frame and ramification problem can provide valuable feedback during the transaction design phase and suggests a new technique for integrity maintenance in large knowledge bases.

In particular, we have presented an adaptation of ideas from AI into the problem of maintaining the integrity of a knowledge base. We adopted and extended a systematic solution to the frame and ramification problem for simplifying - at compile time - the task of proving that transaction execution does not violate the integrity constraints. This becomes possible by the syntactic generation of successor state axioms which, for the case of determinate transaction specifications, completely characterize under which circumstances predicates or functions change value after a state transition. The complexity of the process of generating these axioms is polynomial in the number of predicates and functions in the constraints [LR94]. We extended the method to apply to a class of constraints which includes static and transition constraints. We showed that the introduction of new transaction specifications or new constraints can be accommodated incrementally. This technique can lead to the development of a tool which, as part of a knowledge base management system [MCP+95], will suggest additions to transaction postconditions, whose effect will be to maintain the invariants. This tool aims at assisting the database design process by providing feedback to the designer of transactions and by automating the task of verifying the safety of transactions.

Furthermore, a new technique for integrity constraint checking is suggested. Checking for the satisfaction of constraints after each transaction execution is saved for the constraints for which, it can be decided at compile-time that they remain provably true. Similarly, for those that are provably violated, the cost of undoing transactions is saved. Otherwise, a simpler condition sufficient for ensuring that constraints will not be violated is derivable. The proposed method is a promising avenue for achieving acceptable run-time performance for the problem of maintaining the integrity of large knowledge bases as it is updated through transactions.

We are currently investigating the possibility of devising similar syntactic generators for general dynamic constraints, as well as the application of additional optimization steps for generating simpler conditions that suffice to guarantee invariant maintenance. For the case of temporal constraints in particular, the minimization of the temporal information required to verify the constraints appears to be possible by using knowledge about the satisfaction of the constraints in the history of states up to the current state.

References

[BDM88] F. Bry, H. Decker, and R. Manthey. A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases. In *Proceedings of the Int. Conference on Extedning Data Base Technology*, pages 488–505, 1988.

[BMR93] A. Borgida, J. Mylopoulos, and R. Reiter. And nothing else changes: The Frame Problem in Procedure Specifications. In *Proceedings of the 15th Int. Conference on Software Engineering*, 1993.

[BMR95] A. Borgida, J. Mylopoulos, and R. Reiter. The Frame Problem in Procedure Specifications. *IEEE Transactions on Software Engineering*, 1995. To appear.

[Cho92] J. Chomicki. History-less Checking of Dynamic Integrity Constraints. In *8th Int. Conference on Data Engineering*, pages 557–564, Phoenix,AZ, 1992.

[CT94] Jan Chomicki and David Toman. Implementing Integrity Constraints Using an Active DBMS. In *Proceedings, RIDE-94 Active Database Systems*, pages 87–95, 1994.

[CW90] S. Ceri and J. Widom. Deriving Production Rules for Constraint Maintenance. In *Proceedings of the 16th Int. Conference in Very Large Databases*, pages 566–577, 1990.

[Fin88] J. Finger. Exploiting Constraints in Design Synthesis. Technical Report STAN-CS-88-1204, Stanford University, 1988.

[Flo74] J. Florentin. Consistency Auditing of Data Bases. *Computer Journal*, 17(1):52–58, 1974.

[GM79] G. Gardarin and M. Melkanoff. Proving Consistency of Database Transactions. In *Proceedings VLDB '79*, pages 291–298, 1979.

[GS87] M. Ginsberg and D. Smith. Reasoning about Action I: A Possible Worlds Approach. In Matthew Ginsberg, editor, *Readings in Non-Monotonic Reasoning*, pages 433–463. Morgan Kauffmann, 1987.

[JJ91] M. Jeusfeld and M. Jarke. From Relational to Object-Oriented Integrity Simplification. In *2nd Int. Conference on Deductive and Object-Oriented Databases*, pages 460–477, 1991.

[Lif91] V. Lifschitz. Towards a Metatheory of Action. In *Proceedings of the 2nd International Conference on Knowledge Representation and Reasoning*, pages 376–386, 1991.

[Lip90] U. Lipeck. Transformation of Dynamic Integrity Constraints into Transaction Specifications. *Theoretical Computer Science*, 76:115–142, 1990.

[LR92] F. Lin and R. Reiter. State Constraints Revisited. In *Proceedings of the 2nd Symposium on Logical Formalizations of Commonsense Reasoning*, pages 114–121, 1992.

[LR94] F. Lin and R. Reiter. State Constraints Revisited. *Journal of Logic and Computation - Special Issue on Actions and Processes*, 1994.

[LST86] J. Lloyd, E. Sonenberg, and R. Topor. Integrity Constraint Checking in Stratified Databases. Technical Report 86/5, Department of Computer Science, University of Melbourne, 1986.

[LTW93] M. Lawley, R. Topor, and M. Wallace. Using Weakest Preconditions to Simplify Integrity Constraint Checking. In *Proceedings of the Australian Database Conference*, pages 161–170, 1993.

[MBJK90] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: A Language for Representing Knowledge in Information Systems. *ACM Transactions On Information Systems*, 8(4):325–362, 1990.

[McC69] J. McCarthy. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence 4*, pages 463–502. Edinburg University Press, 1969.

[MCP+95] J. Mylopoulos, V. Chaudhri, D. Plexousakis, A. Shrufi, and T. Topaloglou. Building Knowledge Base Management Systems. *The VLDB Journal*, 1995. To appear.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1991. Vol. 1: Specification.

[Nic82] J.-M. Nicolas. Logic for Improving Integrity Checking in Relational Databases. *Acta Informatica*, 18:227–253, 1982.

[Pin94] J. Pinto. *Temporal Reasoning in the Situation Calculus*. PhD thesis, Department of Computer Science, University of Toronto, 1994.

[Ple93a] D. Plexousakis. Integrity Constraint and Rule Maintenance in Temporal Deductive Knowledge Bases. In *Proceedings of the 19th Int. Conference on Very Large Data Bases*, pages 146–157, Dublin, IR, August 1993.

[Ple93b] D. Plexousakis. Semantical and Ontological Considerations in Telos: a Lanugage for Knowledge Representation. *Computational Intelligence*, 9(1):41–72, 1993.

[Ple95] D. Plexousakis. Compilation and Simplification of Temporal Integrity Constraints. In *Proceedings of the 2nd Int. Workshop on Rules in Database Systems*, Athens, GR, September 1995. To appear.

[Rei91] R. Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Model for Goal Regression. In V. Lifschitz, editor, *Artificial Intelligence and the Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Sch90] L. Schubert. Monotonic Solution to the Frame Problem in the Situation Calculus: An Efficient Method for Worlds with Fully Specified Actions. In H. Kyberg, R. Loui, and G. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. 1990.

[SMS87] D. Stemple, S. Mazumdar, and T. Sheard. On the Modes and Meaning of Feedback to Transaction Designers. In *Proceedings of ACM-SIGMOD Int. Conference on the Management of Data*, pages 374–386, San Francisco, CA, 1987.

[SP87] S. Schuman and D. Pitt. Object-Oriented Subsystem Specification. In L. Meertens, editor, *Program Specification and Transformation*, pages 313–341. Elsevier Science, 1987.

[SS89] Tim Sheard and David Stemple. Automatic Verification of Database Transaction Safety. *ACM Transactions on Database Systems*, 14(3):322–368, 1989.

[Sto75] M. Stonebraker. Implementation of Integrity Constraints and Views by Query Modification. In *Proceedings of ACM-SIGMOD Int. Conference on the Management of Data*, pages 65–78, 1975.

[Ten95] Teniente, Ernest and Olivé, Antoni. Updating Knowledge Bases While Maintaining Their Consistency. *The VLDB Journal*, 4(2):193–241, 1995.