Compilation and Simplification of Temporal Integrity Constraints

Dimitris Plexousakis
Dept. of Computer Science, University of Toronto
Toronto, Ont M5S 1A4, Canada
E-mail: dp@ai.toronto.edu

April 12, 1995

Abstract

The paper presents a novel compilation scheme for temporal integrity constraints and deductive rules expressed in an interval-based first-order temporal logic. Compilation builds a dependence graph with simplified forms of the constraints and rules. This permits the compile-time simplification of the formulae that have to be verified at run-time, as well as the precomputation of potential implicit updates. Simplification takes place with respect to the anticipated types of updates. We show how simplified forms can be obtained with respect to transactions made up of arbitrary sequences of basic updates. Additional optimization steps exploit the organization of simplified forms in dependence graphs.

1 Introduction

The maintenance of semantic integrity has been recognized as a cornerstone issue for the development of data bases and knowledge bases alike [Flo74], [EC75], [Ull88], [MCP+94]. Integrity constraints express application dependent semantics that are not built into the semantics of the knowledge representation language or the data structures used. Additionally, they constitute a means for controlling the quality of information stored in knowledge repositories.

Despite the extensive research conducted during the last decade, semantic integrity enforcement has yet to become a practical technology. This is due to the lack of efficient methods for checking the satisfaction of general integrity constraints. Commercial database management systems provide automatic enforcement of limited types of integrity constraints, such as keys and referential integrity constraints, if at all. Moreover, the vast majority of research results on semantic integrity maintenance concern *static* integrity constraints, whereas only relatively few papers have dealt with the enforcement of *transition* or *dynamic* integrity constraints.

Furthermore, the need for modeling evolving domains has given rise to challenging research issues relating to the incorporation of time in knowledge bases. The by now well established notions of *static* and *transitional integrity* [Nic82], [ELG84] must be generalized to that of *temporal integrity.* The problem of ensuring that the correctness criteria expressed by integrity constraints will not be violated in a dynamic environment, now has an additional dimension, namely that of monitoring time-dependent properties. Such properties arise naturally in dynamic domains and the notion of time contained in such properties can be relative or absolute. For instance, a common task of power plant monitoring systems is to enforce the requirement that values of certain parameters fluctuate in limited ways in certain time periods. Financial and trading applications involve complex operations on time-series, e.g. stock prices over time, and need to preserve constraints on the time-varying characteristics of the objects involved [CS93].

Temporal databases [Tan93] and their extensions [BCW93], [Ple93a] need robust mechanisms for ensuring that time-dependent properties do not become violated due to the evolution of the database or the passage of time. The properties that need to be ensured may refer to arbitrarily many states, past or future, of the database. Thus, the verification of the properties expressed by integrity constraints may involve reasoning in multiple states which must be available at all times. The complexity of verifying temporal integrity constraints is substantially higher than that of verifying properties that refer to a single state only or to pairs of consecutive states.

So far, the research community has dealt with the problem of maintaining semantic integrity in contexts such as relational, deductive or object-oriented databases [BDM88], [CW90], [JJ91]. On the other hand, research in temporal databases has almost exclusively adopted a relational model [Kun85], [SHS94], [Cho92a], [Cho92b], [GL93], [SW95]. Work on temporal deductive databases has mainly dealt with the problem of finitely representing infinite temporal properties [KSW90] and with the evaluation of temporal logic programs [AM87], [BCW93]. This paper constitutes a proposal towards devising efficient methods for enforcing temporal integrity constraints in a structurally object-oriented framework and in the presence of temporal deductive rules.

Temporal integrity maintenance encompasses several challenging research issues. A first such issue is the specification of constraints. We will argue for a *declarative*, as opposed to *procedural*, specification of integrity constraints. *First-Order Temporal Logic* [Eme90] and several of its variants or subsets have been the most popular formalisms for expressing temporal constraints. Deontic variants of Dynamic Logic [WMW89] have also been proposed for the expression of dynamic and deontic properties. In this document, we use a reified temporal logic based on time intervals for the expression of integrity constraints and deductive rules. This language is a variation of the assertion language of Telos [MBJK90].

Integrity constraint verification consists of determining whether all integrity constraints are satisfied in the state resulting after an update. The expressive power of the assertion language, the anticipated large numbers of integrity constraints and deductive rules, and the inherent complexity of deduction in first-order (temporal) logic constitute major impediments to constraint verification. Constraint simplification methods attempt to derive simpler forms of the integrity constraints that have to be verified when an update occurs. Simplification methods can be classified as *run-time* or *compile-time* depending on whether simplification takes place at update time or at knowledge base definition time. Naive constraint verification methods check all integrity constraints after every update. *Incremental* methods, on the other hand, achieve restricting attention to only a subset of all constraints, namely those affected by a particular update type. They are based on the premise that the knowledge base is known not to violate any constraint at the state in which an update takes place. This knowledge can be exploited for the simplification of the formulae expressing the integrity constraints as well as for the minimization of the amount of historical information that needs to be examined for constraint verification.

This paper proposes such an incremental compile-time simplification methods for temporal integrity constraints and deductive rules. The basic approach was described in [Ple93a]. In this paper we focus on enhancements of the method with respect to the treatment of temporal constraints for arbitrary transactions. Specifically, we propose compile-time simplifications to the formulae expressing temporal integrity constraints and deductive rules. The simplified forms are organized in the form of a dependence graph, a structure the captures the logical and temporal interdependence of rules and constraints. The simplified forms generated at compile-time are used for deriving simpler formulae that suffice to be evaluated at run time when arbitrary transactions take place. Furthermore, additional optimization steps aiming at minimizing the validity intervals over which satisfaction of temporal constraints has to be verified, are proposed.

The remainder of this paper is organized as follows. Section 2 introduces the assertion language used for expressing temporal constraints and rules as well as the underlying data model. Section 3 presents the compile time simplification method with respect to individual updates as well as the generation of simplified forms with respect to arbitrary transactions. Section 4 discusses dependence graphs and additional temporal simplification steps that are applied to the compiled formulae. Section 5 summarizes the contributions of this paper and outlines directions for further research.

2 Assertion Language

This section introduces the language for the definition of temporal integrity constraints and deductive rules. The language definition is preceded by a brief introduction to the underlying data model. Finally, the notion of satisfaction of temporal formulae is defined.

2.1 Overview of Telos

The representational framework of Telos [MBJK90] constitutes a generalization of graph-theoretic data structures used in semantic networks, semantic data models and object-oriented representations. Telos treats attributes as first-class citizens, supports a powerful classification mechanism which enhances extensibility and offers special representational and inferential mechanisms for temporal knowledge. In addition, there have been formal accounts of the semantics of the language based on an axiomatic approach [Sta86] or a possible-worlds model [Ple93b]. This section introduces the core features of Telos which are divided into *structural, temporal* and *assertional* features. We only present features that are relevant to the processing of constraints and rules. A more comprehensive description of the language can be found elsewhere [MBJK90].

2.2 Structural Component

A Telos knowledge base consists of structured objects built out of two kinds of primitive units, *individuals* and *attributes*. Individuals are intended to represent entities (concrete ones such as `John`, or abstract ones such as `Person`), while attributes represent binary relationships between entities or other relationships. Individuals and attributes are referred to by a common term – *proposition*. As in object models, Telos propositions have their own internal identifiers. Propositions (individuals and attributes) are organized along three dimensions, referred to in the literature as *attribution*, *classification* and *generalization* [BMS84].

Structured objects consist of collections of (possibly multi-valued) attributes that have a common proposition as a source, thus adding a simple form of *aggregation*. Each proposition is an instance of one or more generic propositions called *classes* – thus giving rise to a classification hierarchy. Propositions are classified into *tokens* – propositions having no instances and intended to represent concrete entities in the domain of discourse, *simple classes* – propositions having only tokens as instances, *meta classes* – having only simple classes as instances, *metametaclasses*, and so on. Orthogonal to the classification dimension, classes can be organized in terms of *generalization* or `isA` hierarchies. A class may have incomparable generalizations leading to hierarchies that are directed acyclic graphs rather than trees. The attribute mechanism is also used for attaching assertions (deductive rules and integrity constraints) to Telos objects. Inheritance of attributes

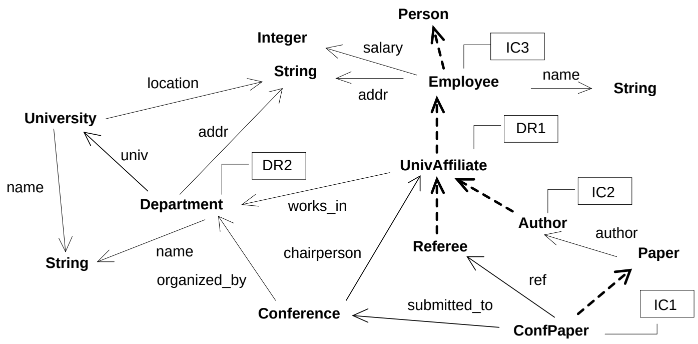

Figure 1: An example Telos knowledge base

and assertions with respect to generalization is assumed to be strict in the sense that a class definition cannot override inherited attributes and assertions.

Figure 1 shows an example Telos knowledge base in the form of a labeled directed graph. Instantiation, generalization and attribution relationships are represented as edges in the graph.[1]

2.2.1 Temporal Knowledge

Every Telos proposition has an associated *history* time and a *belief* time. The history time of a proposition represents the lifetime of a proposition in the application domain (i.e., the lifetime of an entity or a relationship). A proposition's belief time, on the other hand, refers to the time when the proposition is believed by the knowledge base, i.e., the interval between the moment the proposition is added to and that when it is removed from the knowledge base. [2] Both history and belief time are represented by means of time *intervals*. The model of time adopted is a modification of Allen's framework [All83]. Seven exclusive temporal relations (e.g, `equal, meet, before, after, during, start, end`) together with their inverses are used to characterize the possible positions of two intervals on a linear time line. Temporal relationships participate in the expression of deductive rules and integrity constraints in the assertion language. Disjunction of temporal relationships is disallowed in order to facilitate temporal reasoning.

2.3 The Assertional Component

Telos provides an assertion language for the expression of deductive rules and integrity constraints. The assertion language is a first-order language with equality. The terms of the language include variables and constants, including conventional dates.

The atomic formulae of the assertion language include the predicates `instanceOf(), isA()`, *att*() for an attribute *att*, the temporal predicates `before, during, overlaps, meets, starts, finishes, equal` and their inverses `after, contains, overlapped-by, met-by, started-by` and `finished-by`. The predicate[3] `instanceOf(x,y,t)` means that `x` is an instance of `y` for the time period `t`. Similarly, the predicate `isA(x y,t)` means that `x` is a specialization of `y` for time

[1] Attribution is represented using solid edges whereas generalizations is represented using dashed edges.

[2] Belief time is also known as *transaction* time.

[3] For the purposes of this paper we will assume that only historical time appears in assertions. Belief time can be treated in a similar manner, as shown in [Ple93a].

t Finally, att(x y,t) denotes that y is a value of the attribute att of x for t. Also, for any terms x and y and any evaluable predicate θ, $x\ \theta\ y$ is an atomic formula with the obvious meaning.

Well-formed formulae of the assertion language are formed by using logical connectives and restricted quantification. Restricted quantification can be of the form Forall x/C and Exists x/C for a Telos class C. The assertion language is used to specify the integrity constraints and deductive rules in the knowledge base.

Integrity constraints are expressed declaratively as rectified[4] closed wffs of the assertion language. An integrity constraint can have one of the following two forms:

$$\mathtt{I} \equiv \forall x_1/\mathtt{C}_1 \ldots \forall x_k/\mathtt{C}_k F, \text{ or } \mathtt{I} \equiv \exists x_1/\mathtt{C}_1 \ldots \exists x_k/\mathtt{C}_k\ F$$

where F is any well-formed formula (wff) of the assertion language whose quantified subformulae are of the above forms and in which the variables $x_1, \ldots, x_k$ are free variables. Each $\mathtt{C}_i$ is a Telos class and the meaning of each restricted quantification is that the variable bound by the quantifier ranges over the extension of the class instead of the entire domain. Any constraint in this form is *range-restricted*.[5] The typed quantifications $\forall x/\mathtt{C}\ F$ and $\exists x/\mathtt{C}\ F$ are short forms for the formulae:

$$\forall x\ \forall t\ \mathtt{instanceOf}(x, \mathtt{C}, t) \wedge \mathtt{instanceOf}(t, \mathtt{TimeInterval}, \mathtt{Alltime}) \Rightarrow F \qquad \text{and,}$$
$$\exists x\ \exists t\ \mathtt{instanceOf}(x, \mathtt{C}, t) \wedge \mathtt{instanceOf}(t, \mathtt{TimeInterval}, \mathtt{Alltime}) \wedge F$$

The introduction of temporal variables and their restricting literals is necessary since all atomic formulae of the assertion language have a temporal component.

Example 2.1 Referring to the knowledge base of figure 1, the following formula expresses the constraint that "*no author of a conference paper can be a referee for it*".

$$\mathtt{Forall}\ p/\mathtt{ConfPaper}\ \mathtt{Forall}\ x/\mathtt{Author}\ \mathtt{Forall}\ r/\mathtt{Referee}\ \mathtt{Forall}\ t/\mathtt{TimeInterval}$$
$$(author(p, x, t) \wedge referee(p, r, t) \Rightarrow (r \neq x))\ (\mathtt{at\ 1993..*})$$

The above constraint is an example of a *static* constraint, i.e. a property applicable to all states of the domain of discourse. The canonical example of a *dynamic* integrity constraint, expressing the property that "*an employee's salary should never decrease*" can be expressed by the following assertion language formula:

$$\mathtt{Forall}\ p/\mathtt{Employee}\ \mathtt{Forall}\ s, s'/\mathtt{Integer}\ \mathtt{Forall}\ t_1, t_2/\mathtt{TimeInterval}$$
$$(salary(p, s, t_1) \wedge salary(p, s', t_2) \wedge \mathtt{before}(t_1, t_2) \Rightarrow (s \leq s'))\ (\mathtt{at\ 02/01/1993..*}) \qquad \Box$$

Deductive rules are considered to be special cases of integrity constraints. Their general form is:

$$\mathtt{DR} \equiv \forall x_1/\mathtt{C}_1 \ldots \forall x_n/\mathtt{C}_n\ (F \Rightarrow A)$$

where F is subject to the same restrictions as above and A is an atomic formula of the assertion language. In addition, deductive rules are assumed to be *stratified* [LT85].

Example 2.2 Consider the rule "*A university affiliate works in the department that has the same address as she does*", expressed as the formula:

$$\mathtt{Forall}\ u/\mathtt{UnivAffiliate}\ \mathtt{Forall}\ d/\mathtt{Department}\ \mathtt{Forall}\ s, s'/\mathtt{Address}\ \mathtt{Forall}\ t/\mathtt{TimeInterval}$$
$$(address(u, s, t) \wedge (D\_addr(d, s', t) \wedge (s = s') \Rightarrow\ works\_in(u, d, t))\ (\mathtt{at\ 1993..*}) \qquad \Box$$

[4] A formula is rectified if no two quantifiers introduce the same variable [BDM88].

[5] This class of constraints is equivalent to both the *restricted quantification form* of [BDM88] and the *range form* of [JJ91].

Constraints, as well as deductive rules, are associated with history and belief time intervals. If no such association appears explicitly with their definition, both intervals are assumed to be equal to `(systime..*)`, where `systime` denotes the current system time.

We conjecture that the assertion language we use is at least as expressive as First-Order Temporal Logic ($FOTL$) [Eme90]. In [Ple95] we show a translation of the future and past fragments of $FOTL$ into the assertion language of Telos.

2.4 Satisfaction of Temporal Assertions

In the sequel, we define the notion of satisfaction of temporal formulae and that of temporal integrity constraints in particular.

A Telos knowledge base KB comprises a set of propositions, KB_P, defining the validity of predicates over (history) time intervals, as well as a set, KB_R, of deductive rules and a set, KB_I, of integrity constraints.

The notions of *object* and *temporal* variable substitutions need to be defined first.

Definition 2.1 (*Object variable substitution*) An *object variable substitution* σ is a function mapping a variable x_i of sort S_i to an instance of the corresponding Telos class C_i so that $\mathtt{instanceOf}(\sigma(x_i), C_i, t) \in KB_P$ for some interval t.

Let I denote the set of time intervals with integer endpoints. Then, a temporal variable substitution is defined as follows.

Definition 2.2 (*Temporal variable substitution*) A *temporal variable substitution* τ is a function mapping a temporal variable t of sort `TimeInterval` to a time interval in I.

We also need to distinguish between base and derived predicates for defining the satisfaction of temporal formulae with respect to object and temporal substitutions σ, τ.

Definition 2.3 (*Satisfaction of Temporal Formulae*) For a base predicate[6] P

- If P is ground, then $(KB, \sigma, \tau) \models P$ if and only if (iff) $P \in KB_P$.
- $(KB, \sigma, \tau) \models P(x, t)$ iff there exists an interval t' such that $((\tau(t)\ \mathtt{during}\ \tau(t'))$ and $(KB, \sigma, \tau) \models P(\sigma(x), \tau(t'))$.
- $(KB, \sigma, \tau) \models \neg P(x, t)$ iff there does not exist an interval t' containing t such that $(KB, \sigma, \tau) \models P(\sigma(x), \tau(t'))$.
- If Q is also a base predicate, then $(KB, \sigma, \tau) \models P(x, t_1) \vee Q(x, t_2)$ iff $(KB, \sigma, \tau) \models P(x, t_1)$ or $(KB, \sigma, \tau) \models Q(x, t_2)$.
- $(KB, \sigma, \tau) \models \forall x/C\ P(x, t)$ iff $(KB, \sigma[x/d], \tau) \models P(x, t)$ for all d such that $\mathtt{instanceOf}(d, C, T)$ for some interval T contained in $\tau(t)$. $\sigma[x/d]$ is the substitution that differs from σ only in the binding d for x.
- $(KB, \sigma, \tau) \models \forall t/\mathtt{TimeInterval}\ P(x, t)$ iff $(KB, \sigma, \tau[t/T]) \models P(x, t)$ for all intervals T.

If P is a derivable predicate defined by a set of deductive rules with bodies $R_1, \ldots, R_k$ and respective time intervals $T_1, \ldots, T_k$, then

[6] The satisfaction of the basic temporal predicates is defined as in [All83].

- $(KB, \sigma, \tau) \models P(x, t)$ iff $(KB, \sigma, \tau) \models \bigvee_{i=1}^{k} R_i \wedge (t \texttt{ during } T)$, where T is the intersection of the intervals T_i.

Now we're in a position to define the satisfaction of temporal integrity constraints. Recall that every integrity constraint is associated with a validity interval. We will use the notation C [*at* T] to denote an integrity constraint C and its association with the validity interval T.

Definition 2.4 (*Satisfaction of Temporal Integrity Constraints)* If the temporal variables $t_1, \ldots, t_k$ appear in the constraint C, then $(KB, \sigma, \tau) \models C$ [*at* T] iff $(KB, \sigma, \tau) \models C'$, where $C' \equiv C \wedge \bigwedge_{i=1}^{k}(t_i \texttt{ during } T)$.

The above definition shows that the semantics of integrity constraints as first-class citizens of the knowledge base is taken into account into the definition of the notion of satisfaction. It would not make sense to evaluate an integrity constraint outside its validity interval. We will not define the respective notions for the bitemporal case, where both history and belief time are present, since their definitions follow similar lines.

3 Compilation and Simplification of Temporal Constraints

In this section the compilation of constraints and rules into simplified forms is presented. Our method builds on the *compilation* method that was initially proposed in [BDM88] and was later adapted to an object-oriented setting in [JJ91]. Our method extends the latter by the treatment of temporal constraints and by the introduction of an efficient compilation scheme that allows us optimize the computation of implicit updates and to perform additional simplifications. The efficiency of the method stems from the separation of the task of constraint enforcement in two separate phases: a *compilation* phase, performed at schema definition time and an *evaluation* phase performed at knowledge base update time. During compilation, constraints and relevant rules are compiled into simplified forms whose evaluation can be triggered by the occurrence of affecting updates. Compilation and simplification apply uniformly to integrity constraints and the bodies of their relevant deductive rules. It exploits the assumptions that, first, the knowledge base is known to satisfy its constraints prior to an update and, second, that the type of updates can be anticipated. This knowledge is used for generating simplified forms of constraints and for calculating implicit changes to the knowledge base.

In [Ple93a], we presented a compilation method that produced simplified forms of constraints and rules with respect to a single affecting update. In this section, we extend the method to treat dyanmic constraints in a special manner and to take into account multiple updates. Before we proceed in presenting the compilation and simplification steps applied to temporal formulae, we define the notions of *affecting updates*, *transactions* and *literal dependence*.

Definition 3.1 (*Updates* and *Transactions*) An *update* is an instantiated literal of the assertion language whose sign determines whether it is an insertion or a deletion. A *transaction* is an arbitrary sequence of updates.

To reason about which types of updates affect integrity constraints, we will henceforth assume that constraints are written in disjunctive normal form (DNF). It can be seen that a constraint in DNF is affected by an update only when a "tuple" is inserted into the extension of a literal occurring negatively in the constraint, or when a "tuple" is deleted from the extension of a literal occurring positively in the constraint. The definition of *relevance* found in [JJ91] is not sufficient in the presence of time. The following definition provides sufficient conditions for "relevance" of a constraint to an update, by considering the relationship of the history time intervals participating in the literals of the constraint and the update.

Definition 3.2 (*Affecting Updates* and *Transactions*) An update $U(\_, \_, t)$ is an *affecting update* for a constraint C [*at* T] if and only if there exists a literal $L(\_, \_, \_)$ in C such that L unifies with the complement of U and the intersection, $t * T$, of intervals t and T is non-empty. A transaction $X = [U_1, \ldots, U_m]$ is called an *affecting transaction* for a constraint C [*at* T] if and only if at least one of $U_1, \ldots, U_m$ is an affecting update for the constraint.

In the compilation phase, along with each integrity constraint, deductive rules that may contribute to the constraint's evaluation will be compiled. These are the rules whose conclusion literal unifies with literals of the constraint. In this case, it is said that the constraint *directly depends* on the deductive rules. A constraint cannot directly depend on a rule whose conclusion literal does not match any of the constraint's literals. It can however depend transitively on a rule whose conclusion literal matches a condition literal of a rule on which the constraint depends either directly or transitively. Formally, we can define the notions of *dependence* and *direct dependence* along the lines of [JJ91].

Definition 3.3 (*Direct* and *Transitive 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 F unifying with K with most general unifier θ and $A\theta = L$. A literal L *transitively depends* (or, simply, depends) on literal K if and only if it directly depends on K, or depends on a literal M that directly depends on K.

Based on the above definitions, a constraint C depends on a rule R if a literal in the constraint depends on the rule's conclusion literal. Similarly, a rule R_1 depends on a rule R_2 if R_1's body contains a literal that depends on R_2's conclusion literal. These 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. We will elaborate on the structure and properties of a dependence graph in section 4.

We now turn to the definition of compiled forms of constraints and rules. The key issue in compilation is to associate every constraint or rule body with the updates that may affect its evaluation. Moreover, for the purpose of restricting the search space for constraints or rules affected by a transaction the concept of a *concerned class* [JJ91], [Ple93a] is used.

Definition 3.4 (*Concerned Class)* The *Concerned Class* for a literal L is the most specialized class CC such that, inserting or deleting an instance of CC can affect the truth of L and the time intervals of L and CC are unifiable.

[Ple93a] presents a set of rules for the compile-time derivation of concerned classes of literals that occur in integrity constraints or bodies of deductive rules. Compilation produces a *parameterized simplified structure* (PSS) for each such literal.[7] This form contains a simplified form of the constraint and rule that suffices to be evaluated when an affecting update on the literal occurs at run time. It has to be noted that, for every occurrence of a literal in a constraint or rule, there is only one update on the literal that may affect the constraint or rule. Since the only updates possible are insertions and deletions in the extensions of literals, a literal occurrence can only be affected by one of the two operations. The simplified form of a constraint with respect to a transaction is then computed using the simplified forms generated for the particular updates

[7] PSSs are readily representable in Telos via its metaclassing mechanism.

and the sequence of updates in a transaction. The simplification method is based on the premise that integrity constraints are known not to be violated in the state prior to an update and that the types of updates can be anticipated.

3.1 Compile-time Simplification

We first describe the rules for generating a parameterized simplified form for each update and then show how to obtain the simplified form for a transaction given the simplified forms for individual updates. It has to be noted that we cannot possibly derive the simplified forms with respect to all possible transactions that may affect a given constraint due to the exponential number of such transactions.

Definition 3.5 (*Parameterized Simplified Structure*) Given a temporal constraint C $[at\ T]$ expressed in DNF and a literal L occurring positively (negatively) in C, the *parameterized simplified structure* of C with respect to L is a quintuple $(L, Params, CC, T, SF)$, where, $Params$ is the list of instantiation variables[8] occurring as components to L, CC is the concerned class of L, T is the history time interval associated with the constraint, and SF is the simplified form of the constraint that suffices to be evaluated when a deletion from (insertion to respectively) L takes place. SF is derived by applying the following steps:

1. The quantifiers binding instantiation variables are dropped. Variables become parameters.
2. The temporal variables are constrained with respect to the history time of the constraint: a temporal predicate of the form $(t_i\ \texttt{during}\ T)$ is conjoined with C for every temporal variable t_i that occurs in C.
3. The literal into (from) whose extension a tuple is inserted (deleted) is substituted by the Boolean constant `True` (`False`) and absorption rules of first-order logic are applied. If a literal occurs more than once with different history time intervals, then select for replacement the literal with the greatest right endpoint.
4. Temporal simplification rules are applied if applicable (figure 2).

A comment on the treatment of dynamic constraints in step 3 is in order. Dynamic constraints are distinguished from static constraints by the presence of explicit temporal constraints on the history time variables. In addition, since dynamic constraints express properties depending on two or more knowledge base states, some literals will occur more than once in the expression of constraints. Step 4 replaces the literal with the time interval extending to the future of the rest of the intervals in the other literal occurrences. This is done because the constraint must be verifiable using the history of the knowledge base up to the present state. Determining which literal to replace can be done by comparison of the intervals if their endpoints are known, or by reasoning by cases based on the temporal relationship of the time intervals of the literals. The following lemma expresses the above property.

Lemma 3.1 (*Simplification of Dynamic Constraints*) Let C be the integrity constraint $(\neg L(\overline{x}, t_1) \vee \neg L(\overline{y}, t_2) \vee \neg(t_1\ r\ t_2) \vee R)$ $[at\ T]$, where R is a temporal formula in DNF that does not mention any L literal and r is a temporal relation on t_1, t_2. Given the premise that C is known not to be violated in the state prior to the occurrence of an affecting update on L, C's satisfaction can

[8] Instantiation variables are universally quantified variables that do not occur in the scope of an existential quantifier.

r1(t,i2) / r2(i1,i2)	b	d	o	m	s	f	e	a	c	ob	mb	sb	fb
before	**during i1**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
during	**F**	**during i1**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
overlaps	**during i1-i1*i2**	**during i1*i2**	**overlaps i1*i2**	**finishes i1-i1*i2**	**starts i1*i2**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
meets	**during i1**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
starts	**F**	**during i1**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
finishes	**F**	**during i1**	**F**	**F**	**F**	**finishes i1**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
equal	**F**	**during i1**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
after	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**during i1**	**F**	**F**	**F**	**F**	**F**
contains	**during (i1-,i2-)**	**during i2**	**no simp.**	**finishes (i1-,i2-)**	**starts i2**	**finishes i2**	**equal i2**	**during (i2+,i1+)**	**no simp.**	**no simp.**	**starts (i2+,i1+)**	**starts (i2-,i1+)**	**finishes (i1-,i2+)**
overlapped by	**F**	**during (i1-,i2+)**	**F**	**F**	**F**	**finishes (i1-,i2+)**	**F**	**during (i2+,i1+)**	**F**	**no simp.**	**starts (i2+,i1+)**	**F**	**finishes (i1-,i2+)**
met by	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**during i1**	**F**	**F**	**F**	**F**	**F**
started by	**F**	**during i2**	**F**	**F**	**F**	**finishes i2**	**F**	**during i1-i2**	**F**	**no simp.**	**starts i1-i2**	**F**	**F**
finished by	**during (i1-,i2-)**	**during i2**	**no simp.**	**finishes (i1-,i2-)**	**starts i2**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**finishes i1**

Legend

F: false	**no simp.** : no simplification possible
* : intersection operator	**t-** : left endpoint
- : difference operator	**t+** : right endpoint

Figure 2: Temporal Simplification Table

be determined in the state after the update if and only if the satisfaction of $\overline{C}$ can be determined, where $\overline{C}$ is the formula obtained from C as follows: if r is any of the relations `before`, `during`, `overlaps`, `meets`, `starts` or `finishes`, then the literal $\neg L(\overline{y}, t_2)$ is substituted with a Boolean constant and the variables y, t_2 become instantiated. If r is any of the relations `after`, `contains`, `overlapped-by`, `met-by`, `started-by` or `finished-by`, then the literal $\neg L(\overline{x}, t_1)$ is substituted with a Boolean constant and the variables x, t_1 become instantiated. If r is the relation `equal`, any of the literals can be substituted.

Example 3.1 Applying the simplification steps to the dynamic constraint of example 2.1 will generate the following simplified form (capitalized variables denote parameters):

$$\forall s/\mathtt{Integer}\ \forall t_1/\mathtt{TimeInterval}$$
$$(\mathit{salary}(P, s, t_1)\ \wedge (t_1\ \mathtt{during}\ \mathtt{02/01/1993..}{*}\ \wedge\ (t_1\ \mathtt{before}\ T_2) \Rightarrow (s \leq S'))$$

In this example, the literal $\mathit{salary}(p, s', t_2)$ of the original constraint was replaced because of the relationship (t_1 `before` t_2) between t_1, t_2. □

The last step in the generation of simplified forms is the application of temporal simplification. The objective of temporal simplification is to simplify a conjunction of temporal relationships into a single temporal relationship. Hence, the number of subformulae to be evaluated at run-time is reduced. Formally, temporal simplification replaces a conjunction $(t\ \mathtt{during}\ i_1) \wedge (t\ \mathtt{r}_1\ i_2)$, where i_1 and i_2 are known time intervals, with a temporal relationship r, such that $(t\ \mathtt{r}\ i)$ is satisfied if and only if the original conjunction is satisfied. The interval i is a function of the intervals i_1 and i_2. The fact that the intervals i_1 and i_2 are known permits us to derive a relationship $({}_1\ \mathtt{r}_2\ i_2)$.

This relationship is exploited for restricting the possibly multiple alternatives for $\mathtt{r}$. In fact, the expression that is simplified is the conjunction $(t\ \mathtt{during}\ i_1) \wedge (t\ \mathtt{r}_1\ i_2) \wedge (i_1\ \mathtt{r}_2\ i_2)$. It is not always possible to derive a single definite relation $\mathtt{r}$ that has the above property. For some combinations of temporal relationships $\mathtt{r}$ is a disjunction of temporal relationships. In those cases, and for the sake of completeness, we do not replace the original expression by the equivalent disjunction.

The table in figure 2 contains all simplifications that can be carried out at compile time. The rows are labeled with the 13 possible relationships between the time intervals i_1 and i_2, whereas the columns are labeled with those between intervals t and i_2. The column labels are abbreviations of the corresponding row labels. All relationships are treated as being mutually exclusive. The content of each table entry is the relationship $\mathtt{r}$ between t and i that fulfills the properties stated above, if such a relationship can be found without introducing indefiniteness. In the cases where this is not possible, the entries in the table indicate that no simplification is performed. Inconsistencies arising in some of the 169 possible combinations are also discovered. **F** indicates that a combination of relationships is unsatisfiable. In the case that the negation of a temporal relationship appears in r_1 one can only suggest a weaker condition r, which if satisfied guarantees that the original conjunction is satisfied. A table similar to that of figure 2 can be defined for the cases where negation appears in r_1. The operations of intersection and difference can be performed efficiently for intervals with known endpoints. We assume that the cost of performing these operations is negligible compared to that of evaluating an atomic formula of the assertion language. The task of temporal simplification does not introduce any prohibitive complexity. Its requirements include determining the relationship between the known time intervals, a constant time operation, a table lookup for finding the simplifying expression and simple operations between interval endpoints.

The proofs of soundness of simplification and of the completeness of temporal simplification are omitted due to lack of space. They can be found in [Ple94] and [Ple95]. The following theorem expresses the soundness of the simplification.

Theorem 3.1 (*Soundness of Constraint Simplification for Single Updates*) Given a temporal constraint C known not to be violated in the state prior to the occurrence of an affecting update, C is violated in the state resulting from the update if the formula produced after applying the simplification steps 1-4 is.

3.2 Generating Simplified Forms for Transactions

We now turn to the derivation of simplified forms of constraints with respect to arbitrary transactions. This derivation takes place at run-time when the actual transaction is specified, but uses the simplified forms that were generated with respect to the individual updates at compile-time.

As far as the integrity of a knowledge base is concerned, transactions are considered to be atomic. Hence, the effect of a transaction made up of any subset of the affecting updates for a particular integrity constraint is the same independently of the order in which the updates occur in the transaction. [9] This is easy to see, if we consider that it is known the the constraint is not violated in the state prior to the transaction's execution and we apply the simplification steps for all updates occurring in the transaction simultaneously. The above observation allows us to use the simplified forms generated for individual updates in order to derive one simplified form that suffices to be evaluated in order to verify the satisfaction of a constraint with respect to a transaction containing an arbitrary sequence of the individual updates.

[9] Of course, the order of updates is important for producing the result intended by the transaction specifier.

Definition 3.6 (*Simplification of Constraints with Respect to Transactions)* Let $X = [U_1, \ldots, U_m]$ be a transaction affecting constraint C $[at\ T]$, and let i be the index of the first update U_i in X that affects C. Let also X' be the sequence of updates in X that follow U_i and affect C. Then, the simplified form of C with respect to X is obtained by applying all updates in X' simultaneously to the PSS $(L, Params, CC, T, SF)$ of C that is such that L is unifiable with U_i, and simplifying SF as described in steps 1-4 of steps of definition 3.5.

The resulting formulae can be considerably simpler than the ones produced if only individual updates are considered. The following is a simple example showing the simplification of a constraint with respect to a transaction. We're using first-order predicate calculus notation for ease of exposition.

Example 3.2 Consider the constraint $\forall\overline{x},\overline{y}(P(\overline{x})\wedge\neg Q(\overline{x},\overline{y}))\vee(R(\overline{x},\overline{y})\wedge\neg P(\overline{x}))\vee S(\overline{x},\overline{y})$ and the transaction $X = [\neg P(\overline{X}), \neg R(\overline{X},\overline{Y})]$. If the updates were treated independently, two formulae would need to be verified, namely $\forall\overline{y}(R(\overline{X},\overline{y}) \vee S(\overline{X}))$ and $(P(\overline{X}) \wedge \neg Q(\overline{X},\overline{Y})) \vee S(\overline{X})$. This should be contrasted with the formula $S(\overline{X})$ that is the simplified form of C that suffices to be verified when C is simplified with respect to X. $\square$

The following theorem expressed the second main result of this paper. It establishes the soundness of the simplification method with respect to transactions. Its proof uses the result of theorem 3.1.

Theorem 3.2 (*Soundness of Constraint Simplification for Transactions)* Given a temporal constraint C known not to be violated in the state prior to the execution of an affecting transaction, C is violated in the state resulting from the transaction's execution if the formula produced after applying the simplification steps of definition 3.6 is.

4 Dependence Graphs

In this section we examine the organization of the simplified forms of rules and constraints into, so called, *dependence graphs* and the optimization potentials that this scheme provides. The scheme presented in [Ple93a] is extended to take into account the specific temporal interrelationships between constraints and rules.

4.1 Graph Construction and Properties

The definitions of direct and transitive dependence of constraints/rules on other rules define a directed graph representing how literals, implicitly derived from deductive rules, can affect the integrity of the knowledge base. The graph nodes are the PSSs of rules and constraints. Edges denote dependence of constraints on rules and dependence of rules on other rules. There exists an edge form the node of a rule R to that of a constraint C (a rule R'), if C (R') directly depends on R.

Definition 4.1 (*Dependence Graph)* The dependence graph of a KB is defined as $G(KB) = (V, E)$, where V comprises one node for each PSS of an integrity constraint or deductive rule of KB. The set V of nodes is equal to the union of the set of nodes corresponding to integrity constraints (KB_I) with the set of nodes corresponding to deductive rules (KB_R). Hence, $V = V_I \cup V_R$, where V_I and V_R are the aforementioned sets. The set E of edges is defined as: $E = \{(v_i, v_j) | v_i \in V_R, v_j \in V_I\ \mathit{and}\ v_j\ \mathit{directly\ depends\ on}\ v_i\} \cup \{(v_i, v_j) | v_i, v_j \in V_R\ \mathit{and}\ v_i\ \mathit{directly\ depends\ on}\ v_j\}$.

The set E of edges is made up of edges between rule nodes (E_{RR}) and edges from rule to constraint nodes (E_{RC}).

A dependence graph may contain cycles among deductive rule nodes. This happens in the case the knowledge base contains mutually recursive rules. As shown in [Ple94] the graph is free of trivial cycles and may contain cycles of length at most equal to the number of deductive rules participating in the same recursive scheme. The number of nodes in the dependence graph is in the order of the number of literals occurring in rule and constraint bodies, since one node is created for each compiled form. The number of edges in the graph can be at most equal to twice the number of rules in KB_R since there exists an edge between compiled forms of a rule and a constraint only if the rule's head unifies with the constraint's literal or a literal in another rule's body. For an average number of literals per constraint or rule greater than 2 the dependence graph of a knowledge base is sparse [Ple93a]. The dependence graph is constructed once when the knowledge base is compiled and is updated incrementally when new rules or constraints are inserted or deleted. Although sparse, the dependence graph for a large knowledge base will be quite large, even too large to fit in main memory. The problem of storage of the dependence graph in secondary storage remains as a future research problem. Below we list some of the properties that the above compilation scheme enjoys and which contribute to the efficiency of integrity constraint checking. These properties are elaborated on in [Ple93a] and [Ple94], where performance results with randomly generated dependence graphs are presented. These results attest to the efficiency of the compilation scheme.

A dependence graph reflects both the logical and temporal interdependence of rules and constraints. Following paths from rules to constraints in the graph permits us to derive implicit updates caused by explicit ones. The set of implicit updates can be precomputed at the time of graph construction by computing the graph's transitive closure. At evaluation time, node reachability information does not have to be recomputed. The implicit updates computed in this manner are only *potential* updates. The actual updates can be obtained by instantiating the potential updates and evaluating the rule bodies in which they occur, starting with the ones matching the update's literal and following the order in which the implicit updates were computed. Moreover, the dependence graph structure is incrementally modifiable to accommodate insertion/deletions of rules and constraints without having to recompile the entire graph structure. [Ple94] presents algorithms for efficiently maintaining the graph's precomputed transitive closure.

4.2 Optimizations

We now present additional optimizations that take into account the temporal properties of constraints and rules. These optimizations take place after the graph's construction and aim at producing more efficiently evaluable temporal formulae.

The first optimization step replaces the temporal intervals of the PSSs that comprise the graph nodes by smaller intervals. Algorithm *Minimize Intervals* is applied after the construction of the dependence graph is complete and whenever the dependence graph structure is modified by the insertion of a new constraint or rule. The algorithm, shown in figure 3 starts with an initial set of nodes and, for every node v in this initial set, it replaces the history time intervals of each of its incident nodes with the time interval that results from computing the intersection of the time interval of v with that of its incident node. The algorithm proceeds in a breadth-first fashion until no more replacements can take place. In this manner, the validity time intervals of the formulae whose satisfaction has to be determined at run-time are minimized. Since the history time intervals of the constraints and rules are used to constrain the temporal variables of their

Algorithm: *Minimize Intervals* (G, S)
begin

For every node $v \in S$ {
  $Temp \ := \ \emptyset$
  For every node $u \in V_R$ such that $(u, v) \in E$ {
    $u.T \ := \ (v.T * u.T)$
    $newSF \ := \ u.SF \ \wedge \ \bigwedge_{t_i \in qtv(u.SF)}(t_i \ during \ u.T)$
    $u.SF \ := \ temp\_simp(newSF)$
    $Temp \ := \ Temp \cup \{u\}$
  }
}
$S \ := \ Temp$
Minimize Intervals (G, S)

end

Figure 3: Interval Minimization

respective expressions, the minimized intervals must be used in their place. As was shown in section 3.1, the time intervals originally associated with a rule or constraint are factored-in the rule or constraint expression using a `during` relationship that captures the semantics of temporal validity. When the minimized interval of a PSS is computed, the new interval must also be factored into the simplified form. The `during` relationship that introduced the original interval in the temporal formula expressing the constraint or rule may not appear explicitly due to the simplification step 4 that replaces conjunctions of temporal relationships by a new relationship. Hence, a new `during` relationship introducing the minimized interval must be conjoined with the simplified form. Additional simplifications may be carried out according to the temporal simplification table of figure 2 if applicable. In the presentation of the algorithm in figure 3 we assume that each node is represented as a PSS and we use the "."(dot) operator to refer to its components. The call to *temp_simp* symbolizes the application of step 4 of definition 3.5. Also, the function *qtv* returns the quantified temporal variables of a simplified form.

Initially, the algorithm is applied to a dependence graph G with the set of nodes V_I corresponding to simplified forms of integrity constraints as the set of source nodes S. When G is modified by the insertion of a new integrity constraint, the set of source nodes is the set comprising the PSSs of the new constraint. If a new rule is inserted, then the set of source nodes is the set of successor nodes of all the PSSs of the newly inserted rule. Note that we do not need to apply the algorithm for the case of deletions of constraints or rules since deletion can only disconnect nodes from the dependence graph.

Since a dependence graph may contain cycles, special attention must be paid to the graph's strongly connected connected components which can be identified at compile time. The algorithm of figure 3 can be easily modified to take the graph's strongly connected components into account.

We argue that the optimization steps presented above can yield considerable savings in evaluating simplified forms at run-time by minimizing the intervals over which the evaluation of formulae must take place and by, possibly, carrying out additional temporal simplifications. It is easily seen that the transformations carried out preserve the satisfaction of constraints since only the simplified forms of their relevant rules are modified. Moreover, the derivation of the actual implicit updates becomes simpler since the formulae that have to be verified have been simplified. The following lemma is needed for establishing the correctness of the algorithm.

Lemma 4.1 (*Soundness of Rule Transformations*) Let SF be the simplified form of a rule whose associated time interval is restricted by algorithm *Minimize Intervals* and let SF' be the new simplified form. Then, $KB \models (SF \Rightarrow SF')$.

Theorem 4.1 Algorithm *Minimize Intervals* is correct: if the violation of an integrity constraint C can be determined using the dependence graph G, then it can also be determined using the graph resulting from applying *Minimize Intervals* to G.

5 Conclusions and Outlook

This paper presented a simplification method applying uniformly to temporal integrity constraints and deductive rules in large Telos knowledge bases. The proposed techniques focus on simplifying formulae as much as possible at schema definition time and aim at yielding acceptable performance at knowledge base update time. The method operates in two phases, a compilation phase during which integrity constraints and deductive rules are simplified with respect to the anticipated types of updates and organized in a graph structure that permits the efficient derivation of implicit updates. This structure is incrementally modifiable for accommodating dynamic changes to the rule and constraint set of the knowledge base. Moreover, additional optimizations on the formulae expressing the simplified forms generated at compile-time were proposed. These optimization steps minimize the temporal intervals over which the validity of formulae has to be checked in order for their satisfaction to be verified at update-time. These optimizations exploit the semantics of constraint satisfaction and the organization of simplified forms in the dependence graph structure. Similar simplification steps can be performed when constraints contain belief time, in addition to history time [Ple95].

Another contribution is the simplification of temporal formulae with respect to arbitrary transactions. The simplified formulae produced by the compile-time simplification method are used in order to derive a single formula that suffices to be verified at update-time when a transaction takes place. Hence, a single formula incorporating the affecting updates of an arbitrary transaction needs only to be verified as opposed to a number of more complex formulae incorporating only individual updates.

Current research focuses on devising historical knowledge minimization techniques for integrity constraints, in the lines of [Cho92a], [Cho92b], and [SL88], [SHS94] but in the richer context of temporal deductive knowledge bases introduced in this paper. The reuse and adaptation of proofs of temporal formulae satisfaction over dependence graphs is investigated. Last, but not least, the derivation of incrementally evaluable formulae as simplified forms of integrity constraints expressed in the interval based assertion language is examined. Incrementally evaluable constraints lend themselves to an implementation using an active rule model, as, e.g., in [GL93], [CT94] and [SW95].

References

[All83] J. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[AM87] M. Abadi and Z. Manna. Temporal Logic Programming. In *IEEE Symposium on Logic Programming*, pages 4–16, 1987.

[BCW93] Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Temporal Deductive Databases. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases*. Benjamin Cummings, 1993.

[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.

[BMS84] Michael Brodie, John Mylopoulos, and Joachim Schmidt, editors. *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. Springer Verlag, 1984.

[Cho92a] J. Chomicki. History-less Checking of Dynamic Integrity Constraints. In *8th Int. Conference on Data Engineering*, pages 557–564, Phoenix,AZ, 1992.

[Cho92b] Jan Chomicki. Real-Time Integrity Constraints. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* , pages 274–282, San Diego, CA, 1992.

[CS93] R. Chandra and A. Segev. Managing Temporal Financial Data in an Extensible Database. In *Proceedings of the 19th Int. Conference on Very Large Data Bases*, pages 302–313, Dublin, IR, August 1993.

[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.

[EC75] K. Eswaran and D. Chamberlin. Functional Specifications of a Subsystem for Database Integrity. In *Proceedings of the 1st International Conference in Very Large Databases*, pages 48–68, 1975.

[ELG84] H. Ehrich, U. Lipeck, and M. Gogolla. Specification, Semantics and Enforcement of Dynamic Database Constraints. In *Proceedings of 10th VLDB Conference*, pages 301–308, 1984.

[Eme90] E. Allen Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. MIT Press, 1990.

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

[GL93] Michel Gertz and Udo Lipeck. Deriving Integrity Maintaining Triggers from Transition Graphs. In *International Conference on Data Engineering*, pages 22–29, 1993.

[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.

[KSW90] F. Kabanza, J-M. Stevenne, and P. Wolper. Handling Infinite Temporal Data. In *PODS-90*, pages 392–403, 1990.

[Kun85] C.H. Kung. On Verification of Database Temporal Constraints. In *Proceedings ACM-SIGMOD International Conference on Management of Data*, pages 169–179, 1985.

[LT85] J. Lloyd and R. Topor. A Basis for Deductive Database Systems. *Journal of Logic Programming*, 2:93–109, 1985.

[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.

[MCP+94] J. Mylopoulos, V. Chaudhri, D. Plexousakis, A. Shrufi, and T. Topaloglou. Building Knowledge Base Management Systems: A Progress Report. Technical Report DKBS-TR-94-4, Department of Computer Science, University of Toronto, 1994.

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

[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.

[Ple94] D. Plexousakis. Integrity Maintenance in a Telos based KBMS, 1994. Submitted to IEEE-TKDE.

[Ple95] D. Plexousakis. *On the Efficient Enforcement of Integrity Constraints in Temporal Deductive Knowledge Bases*. PhD thesis, Department of Computer Science, University of Toronto, 1995. Forthcoming.

[SHS94] S. Schwiderski, T. Hartmann, and G. Saake. Monitoring Temporal Preconditions in a Behavior Oriented Object Model. *Data & Knowledge Engineering*, 14(1):143–186, 1994.

[SL88] G. Saake and U. Lipeck. Foundations of Temporal Integrity Monitoring. In C. Roland, editor, *Temporal Aspects in Information Systems*, pages 235–249. North Holland, 1988.

[Sta86] Martin Stanley. CML: A Knowledge Representation Language with Application to Requirements Modelling. Master's thesis, Department of Computer Science, University of Toronto, 1986.

[SW95] A.P. Sitsla and O. Wolfson. Temporal Conditions and Integrity Constraints in Active Database Systens. In *Proceedings of ACM SIGMOD-95*, 1995. To appear.

[Tan93] A. et al. Tansel. *Temporal Databases, Theory Design and Implementation*. Benjamin/Cummings, 1993.

[Ull88] J. Ullman. *Fundamentals of Database and Knowledge-base Systems*, volume 1. Computer Science Press, 1988.

[WMW89] R. Wieringa, J-J. Meyer, and H. Weigand. Specifying Dynamic and Deontic Integrity Constraints. *Data and Knowledge Engineering*, 4:157–189, 1989.