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

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. 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 is recognized as a cornerstone issue for the development of databases and knowledge bases alike [11], [27], [18]. Integrity constraints express application dependent semantics that are not built into the data structures used to represent knowledge. 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 dynamic constraints.

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 [19], [9] must be generalized to that of temporal integrity. The problem of ensuring that the correctness criteria expressed by integrity constraints will not be violated, now has an additional dimension, namely that of monitoring time-dependent properties. Such properties arise naturally in dynamic domains. For instance, a common task of power plant monitoring systems is to enforce the requirement that values

* Current Address: Department of Computing and Information Sciences, Kansas State University, Manhattan, KS 66506-2302, USA
of certain parameters fluctuate in limited ways in certain time periods. Financial and trading applications need to preserve constraints on the time-varying characteristics of the objects involved, e.g., stock prices over time [6].

Temporal databases [26] and their extensions [2], [20] 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 integrity constraints may involve reasoning in multiple states. 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 [4], [5], [13]. On the other hand, research in temporal databases has almost exclusively adopted a relational model [15], [7], [12], [25]. Work on temporal deductive databases has mainly dealt with the problem of finitely representing infinite temporal properties [14] and with the evaluation of temporal logic programs [2]. This paper proposes an efficient method for enforcing temporal integrity constraints in a structurally object-oriented framework and in the presence of temporal deductive rules.

First-Order Temporal Logic (FOTL) [10] and several of its variants or subsets have been the most popular formalisms for expressing temporal constraints. Deontic variants of Dynamic Logic [28] have also been proposed for the expression of dynamic and deontic properties. We use a reified temporal logic based on time intervals for the expression of constraints and rules.

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 constraints and 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 constraints that have to be verified when updates occur. Simplification methods can be classified as runtime 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, however, restrict 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 temporal formulæ, 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 method for temporal constraints and rules. The basic approach was described in [20]. Here 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 constraints and rules. The simplified forms that suffice to be evaluated at run-time, when arbitrary transactions are issued, are organized in the form of a dependence graph, a structure that captures their logical and temporal interdependence. 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. Section 3 presents 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. Proofs of theorems can be found in [22].

2 Assertion Language

The representational framework of Telos [17] is 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. Telos has a formal semantics based on a possible-worlds model [21]. This section presents the features of Telos that are relevant to the processing of constraints and rules.

![Telos knowledge base](image)

**Fig. 1.** An example Telos knowledge base

### 2.1 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 are organized along three dimensions, referred to in the literature as *attribution, classification* and *generalization* [3].
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, metameta classes, and so on. Orthogonally to the classification dimension, classes can be organized in terms of generalization or isA hierarchies. The attribute mechanism is also used for attaching assertions to Telos objects. Figure 1 shows an example Telos knowledge base in the form of a labeled directed graph.

2.2 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.

Both history and belief time are represented by means of time intervals. The model of time adopted is a modification of Allen's framework [1]. The temporal relations equal, meet, before, after, during, start, end together with their inverses are used to characterize the possible positions of two intervals on a (non-branching) 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. The class Time denotes the class of time intervals. We assume that time is linear, discrete and unbounded. The model also includes temporal constants (dates and times), semi-infinite time intervals, the special interval Alltime and the special interval variable systime denoting the current clock time.

2.3 The Assertional Component

The assertion language is a many-sorted first-order language with equality. The language includes a sort for each built-in or user-defined class. We identify one sort for time intervals. 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 and their inverses. The predicate\(\text{instanceOf}(x, y, t)\) means that \(x\) is an instance of \(y\) for the time period \(t\). Similarly, \(\text{isA}(x, y, t)\)

---

2 The terms history and belief time correspond to the more popular terms of valid and transaction time respectively.

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 [20].
means that \( x \) is a specialization of \( y \) for time \( t \). Finally, \( att(x, y, t) \) denotes that \( y \) is a value of the attribute \( att \) of \( x \) for \( t \). For terms \( x \) and \( y \) and evaluable predicate \( \theta \), \( x \theta y \) is an atomic formula with the obvious meaning.

Well-formed formulae (wffs) of the assertion language are formed by using logical connectives and restricted quantification. Restricted quantification is of the form \( \forall x/C \) and \( \exists x/C \) for a Telos class \( C \). We conjecture that the assertion language we use is at least as expressive as FOTL [23].

Integrity constraints are expressed as rectified\(^4\) closed wffs of the assertion language. An integrity constraint can have one of the following two forms:

\[ I \equiv \forall x_1/C_1 \ldots \forall x_k/C_k F, \quad \text{or} \quad I \equiv \exists x_1/C_1 \ldots \exists x_k/C_k F \]

where \( F \) is any wff whose quantified subformulae are of the above forms and in which the variables \( x_1, \ldots, x_k \) occur free. Each \( C_i \) 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. Any constraint in this form is range-restricted.\(^5\) The quantifications \( \forall x/C \) and \( \exists x/C \) are short forms for:

\[ \forall x \forall t \ \text{instanceOf}(x, C, t) \land \text{instanceOf}(t, \text{Time}, \text{AllTime}) \Rightarrow F \quad \text{and} \quad \exists x \exists t \ \text{instanceOf}(x, C, t) \land \text{instanceOf}(t, \text{Time}, \text{AllTime}) \land F \]

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

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

\[
\forall p/\text{ConfPaper} \forall x/\text{Author} \forall r/\text{Referee} \forall t/\text{Time} \\
(\text{author}(p, x, t) \land \text{referee}(p, r, t) \Rightarrow (r \neq x)) \quad \text{(at 1993..*)}
\]

The above constraint is an example of a static constraint. The canonical example of a dynamic integrity constraint, expressing the property that "an employee’s salary should never decrease", can be expressed by the formula:

\[
\forall p/\text{Employee} \forall s, s'/\text{Integer} \forall t_1, t_2/\text{Time} \\
(\text{salary}(p, s, t_1) \land \text{salary}(p, s', t_2) \land \text{before}(t_1, t_2) \Rightarrow (s \leq s')) \quad \text{(at 02/01/93..*)}
\]

The general form of deductive rules in the assertion language is:

\[ DR \equiv \forall x_1/C_1 \ldots \forall x_n/C_n (F \Rightarrow A) \]

where \( F \) is subject to the same restrictions as above and \( A \) is an atomic formula. Rules are assumed to be stratified [16]. Constraints and 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..*).

**Example 2.** Consider the rule “A university affiliate works in the department that has the same address as she does”, expressed as the formula:

\[
\forall u/\text{UnivAffiliate} \forall d/\text{Department} \forall s, s'/\text{Address} \forall t/\text{Time} \\
(\text{address}(u, s, t) \land (D_{\text{addr}}(d, s', t) \land (s = s')) \Rightarrow \text{works\ in}(u, d, t)) \quad \text{(at 1993..*)}
\]

\(^4\) A formula is rectified if no two quantifiers introduce the same variable [4].

\(^5\) This class of constraints is equivalent to both the restricted quantification form of [4] and the range form of [13].
2.4 Satisfaction of Temporal Assertions

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.

**Definition 1.** An object variable substitution $\sigma$ is a function mapping a variable $x_i$ of sort $S_i$ to an instance of the corresponding Telos class $C_i$ so that $\text{instanceOf}(\sigma(x_i), C_i, t) \in KB_P$ for some interval $t$.

**Definition 2.** Let $I$ denote the set of time intervals with integer endpoints. A temporal variable substitution $\tau$ is a function mapping a temporal variable $t$ of sort $\text{Time}$ to an interval in $I$.

**Definition 3 (Satisfaction of Temporal Formulae).** For a base predicate$^6$ $P$, object substitution $\sigma$ and temporal substitution $\tau$

- 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
  
  \begin{align*}
  \text{((\tau(t) \text{ during } \tau(t')) and (KB, \sigma, \tau) \models P(\sigma(x), \tau(t'))).}
  \end{align*}

- $(KB, \sigma, \tau) \models \neg P(x, t)$ 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 $(KB, \sigma, \tau) \models P(x, t) \lor Q(x, t)$ 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 $\text{instanceOf}(d, C, T)$ for some interval $T$ contained in $\tau(t)$. $\sigma[x/d]$ is the substitution that differs from $\sigma$ only in the binding $d$ for $x$.

- $(KB, \sigma, \tau) \models \forall t / \text{Time} \ P(x, t)$ iff $\forall T \in I \ (KB, \sigma, \tau[t/T]) \models P(x, t)$.

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

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

Now we are in a position to define the satisfaction of temporal integrity constraints. We will use the notation $C \ [at \ T]$ to denote an integrity constraint $C$ associated with the history time interval $T$.

**Definition 4 (Satisfaction of Temporal Integrity Constraints).** If the temporal variables $t_1, \ldots, t_k$ occur in the constraint $C$, then $(KB, \sigma, \tau) \models C \ [at \ T]$ iff $(KB, \sigma, \tau) \models C'$, where $C' \equiv C \land \bigwedge_{i=1}^k (t_i \text{ during } T)$.

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

$^6$ The satisfaction of the basic temporal predicates is defined as in [1].
3 Compilation and Simplification

Our approach builds on the method proposed in [4], which was adapted to an object-oriented setting in [13]. It extends the latter by the treatment of temporal constraints and by the introduction of an efficient compilation scheme that allows us to 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 maintenance 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 types of updates can be anticipated.

In [20], we presented a compilation method that produced simplified forms of constraints and rules with respect to a single affecting update. We extend the method to treat dynamic constraints in a special manner and to take into account multiple updates. The notions of affecting updates, transactions and literal dependence are defined first.

Definition 5. An update is an instantiated literal whose sign determines whether it is an insertion or a deletion. A transaction is an arbitrary sequence of updates.

We will henceforth assume that constraints are written in disjunctive normal form (DNF). 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 [13] is not sufficient in the presence of time. Definition 6 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 6. An update \( U(\omega t) \) is an affecting update for a constraint \( C \) [at \( T \)] iff there exists a literal \( L(\omega t) \) in \( C \) such that \( L \) unifies with the complement of \( U \) and the intersection, \( t \ast T \), 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 \( T \)] iff at least one of \( U_1, \ldots, U_m \) is an affecting update for the constraint.

During compilation, along with each integrity constraint, deductive rules that may contribute to the constraint’s evaluation will be compiled. These are 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 on a rule whose conclusion literal matches a condition literal of a rule on which the constraint depends. We define the notions of dependence and direct dependence along the lines of [13].
Definition 7. A literal \( L \) directly depends on a literal \( K \) iff there exists a rule of the form \( \forall x_1 / \mathcal{C}_1 \ldots \forall x_n / \mathcal{C}_n \) (\( F \Rightarrow A \)) such that, there exists a literal in \( F \) unifying with \( K \) with most general unifier \( \theta \) and \( A \theta \Rightarrow L \). A literal \( L \) transitively depends (or, simply, depends) on a literal \( K \) if it directly depends on \( K \), or depends on a literal \( M \) that directly depends on \( K \).

Hence, 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. A dependence graph represents how implicitly derived facts can affect the integrity of the knowledge base. Dependence graphs are discussed in section 4.

The key issue in compilation is to associate every constraint or rule body with the updates that may affect its evaluation. Compilation produces a parameterized simplified structure (PSS) for each such literal. This form contains a simplified form of the constraint or rule that suffices to be evaluated when an affecting update on the literal occurs at run time. Note that, for every occurrence of a literal in a constraint or rule, there is only one update 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.

3.1 Compile-time Simplification

We first list the rules for generating a PSS simplified form for each update and then show how to obtain the simplified form for a transaction given the simplified forms for individual updates. The analysis of the soundness and completeness of the rules can be found in [22]. Note 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 8. 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\(^7\) of \( L \), \( CC \) is the concerned class\(^8\) of \( L \), \( T \) is the history time interval of the constraint, and \( SF \) is the simplified form of the constraint that suffices to be evaluated when a deletion from (insertion to) \( L \) takes place. \( SF \) is derived as follows:

1. The quantifiers binding instantiation variables are dropped. Variables become parameters.

\(^7\) Instantiation variables are \( \forall \)-quantified variables that are not in the scope of a \( \exists \).

\(^8\) The Concerned Class [13], [20] 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. The compile-time derivation of concerned classes of literals is discussed in [20].
2. The temporal variables are constrained with respect to the history time of the constraint: a temporal predicate of the form \((t_i \text{ during } T)\) 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 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.

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. Since they express properties depending on two or more knowledge base states, some literals will occur more than once in their expression. Step 3 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. Lemma 9 expresses the above property.

**Lemma 9.** Let \(C\) be the constraint \((\neg L(x, t_1) \lor \neg L(y, t_2) \lor \neg (t_1 r t_2) \lor R)\) [at \(T\)], where \(R\) is a temporal formula in DNF that does not mention any \(L\) literal and \(r\) is a temporal relation. Given 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 be determined in the state after the update iff the satisfaction of \(\overline{C}\) can be determined, where \(\overline{C}\) is the formula obtained from \(C\) as follows: if \(r\) is before, during, overlaps, meets, starts or finishes, then the literal \(\neg L(y, t_2)\) is substituted with a Boolean constant and the variables \(y, t_2\) become instantiated. If \(r\) is after, contains, overlapped-by, met-by, started-by or finished-by, then the literal \(\neg L(x, t_1)\) is substituted with a Boolean constant and the variables \(x, t_1\) become instantiated. If \(r\) is equal, any of the literals can be substituted.

**Example 3.** Applying steps 1-4 to the dynamic constraint of example 1 will generate the following simplified form (capitalized variables denote parameters):

\[
\forall s/\text{Integer} \forall t_1/\text{TimeInterval} (\text{salary}(P, s, t_1) \land (t_1 \text{ during } 02/01/1993..) \land (t_1 \text{ before } T_2) \Rightarrow (s \leq S'))
\]

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

The last step in the generation of simplified forms is temporal simplification. The objective is to simplify a conjunction of temporal relationships into a single temporal relationship. Hence, the number of subformulas to be evaluated at runtime is reduced. Temporal simplification replaces a conjunction \((t \text{ during } i_1) \land (t \text{ over } i_2)\), where \(i_1\) and \(i_2\) are known time intervals, with a temporal relationship \(r\), such that \((t \text{ for } i)\) is satisfied iff 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 \( r_3 \) that is true of \((i_1 \text{ and } i_2)\). The expression that is simplified is the conjunction \((t \text{ during } i_1) \land (t \text{ during } i_2) \land (i_1 \text{ and } i_2)\). It is not always possible to derive a single definite relation \( r \) that has the above property. For some combinations of temporal relationships, \( r \) is a disjunction of temporal relationships. In these cases, and for the sake of completeness, we do not replace the original expression by the equivalent disjunction. Temporal simplification can be performed efficiently as shown in [20]; only a table lookup is needed for any of the 169 possible combinations of relations \( r_1 \) and \( r_2 \). 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.

The following theorem expresses the soundness of the simplification.

**Theorem 10.** A constraint \( C \), known not to be violated in the state prior to the occurrence of an affecting update, 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 that 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.

**Definition 11.** Let \( X = [U_1, \ldots, U_m] \) be a transaction affecting constraint \( C \), and let \( i \) be the index of the first \( U_i \) in \( X \) that affects \( C \). Let \( X' \) 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 \((L, \text{Params}, CC, T, SF)\) of \( C \) that is such that \( L \) is unifiable with \( U_i \).

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 are using first-order predicate calculus notation for ease of exposition.

\(^9\) Of course, the order of updates is important for producing the result intended by the transaction specifier.
Example 4. Consider the constraint $\forall \mathbf{y} \left( P(\mathbf{x}) \land \neg Q(\mathbf{x}, \mathbf{y}) \right) \lor \left( R(\mathbf{x}, \mathbf{y}) \land \neg P(\mathbf{x}) \right) \lor S(\mathbf{x}, \mathbf{y})$ and the transaction $X = \neg P(\mathbf{x}), \neg R(\mathbf{x}, \mathbf{y})$. If the updates were treated independently, two formulae would need to be verified, namely $\forall \mathbf{y} \left( R(\mathbf{x}, \mathbf{y}) \lor S(\mathbf{x}, \mathbf{y}) \right)$ and $\left( P(\mathbf{x}) \land \neg Q(\mathbf{x}, \mathbf{y}) \right) \lor S(\mathbf{x}, \mathbf{y})$. This should be contrasted with the formula $S(\mathbf{x}, \mathbf{y})$ that is the simplified form of $C$ that suffices to be verified when $C$ is simplified with respect to $X$.

The following theorem establishes the soundness of the simplification method with respect to transactions. Its proof uses the result of theorem 10.

Theorem 12. A temporal constraint $C$, known not to be violated in the state prior to the execution of an affecting transaction, is violated in the state resulting from the transaction's execution if the formula produced after applying the simplification steps of definition 11 is.

4 Dependence Graphs

In this section we examine the organization of the simplified forms of rules and constraints into dependence graphs and the optimization potentials that this scheme provides.

4.1 Graph Construction and Properties

The definitions of direct and transitive dependence 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/rules on rules.

Definition 13. The dependence graph of a knowledge base $KB$ is defined as $G(KB) = (V, E)$, where $V$ comprises one node for each PSS of an integrity constraint or deductive rule of $KB$. $V = V_I \cup V_R$, where $V_I$ and $V_R$ are the sets of nodes corresponding to integrity constraints ($KB_I$) and deductive rules ($KB_R$) respectively. $E = \{(v_i, v_j) | v_i \in V_R, v_j \in V_I \text{ and } v_j \text{ directly depends on } v_i\} \cup \{(v_i, v_j) | v_i, v_j \in V_R \text{ and } v_i \text{ directly depends on } v_j\}$.

A dependence graph may contain cycles among deductive rule nodes. This happens in the case that the knowledge base contains mutually recursive rules. As shown in [22], 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. 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. 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 [20] and
[22], 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 potential 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 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. The dependence graph
structure is incrementally modifiable to accommodate insertions/deletions of
rules and constraints without having to recompile the entire graph. [22] presents
algorithms for efficiently maintaining the graph’s transitive closure.

4.2 Optimizations

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

\begin{algorithm}
\textbf{Algorithm: Minimize Intervals} $(G, S)$
\begin{algorithmic}
\STATE \textbf{begin}
\STATE For every node $v \in S$ \{\ 
\STATE \hspace{1em} $Temp := \emptyset$
\STATE \hspace{1em} For every node $u \in V_R$ such that $(u, v) \in E$ \{
\STATE \hspace{2em} $u.T := (v.T + u.T)$
\STATE \hspace{2em} newSF := $u.SF \land \bigwedge_{t_i \in EV(u, SF)} (t_i \text{ during } u.T)$
\STATE \hspace{2em} $u.SF := \text{temp} \cup \text{imp(newSF)}$
\STATE \hspace{2em} $Temp := Temp \cup \{u\}$ \}
\STATE $S := Temp$
\STATE \textbf{Minimize Intervals} $(G, S)$
\STATE \textbf{end}
\end{algorithmic}
\end{algorithm}

\textbf{Fig. 2.} Interval Minimization

The first optimization step replaces the temporal intervals of the PSSs that
comprise the graph nodes by smaller intervals. Algorithm \textit{Minimize Intervals} (see
figure 2) 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 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 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 into the rule or constraint expression using a \texttt{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 \texttt{during} relationship that introduced the original interval in the temporal formula expressing the constraint or rule may not appear explicitly due to simplification step 4 that replaces conjunctions of temporal relationships by a new relationship. Hence, a new \texttt{during} relationship introducing the minimized interval must be conjoined with the simplified form. Additional simplifications may be carried out if applicable. In the presentation of the algorithm we assume that each node is represented as a PSS and we use the “.” (dot) operator to refer to its components. The call to \texttt{temp.simp} symbolizes the application of step 4 of definition 8. The function \texttt{qte} 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 graph nodes. Since a dependence graph may be cyclic, special attention must be paid to the graph’s strongly connected components which can be identified at compile time. The algorithm 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. 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. Lemma 14 and theorem 15 establish the correctness of the algorithm.

**Lemma 14.** 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 15.** Algorithm Minimize Intervals is correct: if the violation of a constraint \( C \) can be determined using the dependence graph \( G \), then it can also be
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 update time. During compilation 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. Moreover, additional optimizations on the formulae expressing the simplified forms generated at compile-time were proposed. These optimizations 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. They 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 [23].

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 [7], and [24] but in the richer context of 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 constraints expressed in the interval based assertion language is examined. Incrementally evaluable formulae lend themselves to an implementation using an active rule model, as, e.g., in [12], [8] and [25].

References

8. J. Chomicki and D. Toman. Implementing Integrity Constraints Using an Active 
1974.
12. M. Gertz and U. Lepeck. Deriving Integrity Maintaining Triggers from Transition 
13. M. Jentfeld and M. Jarke. From Relational to Object-Oriented Integrity Simplification. 
15. C.H. Kung. On Verification of Database Temporal Constraints. In SIGMOD-85, 
17. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: A Language for 
to IEEE-TKDE.
25. A.P. Sielja and O. Woldson. Temporal Conditions and Integrity Constraints in 
jamin/Cummings, 1993.
28. R. Wieringa, J-J. Meyer, and H. Weigand. Specifying Dynamic and Deontic In- 
This article was processed using the \LaTeX\ macro package with LLNCS style