Transitional Monitoring of Dynamic Integrity Constraints

Udo W. Lipeck, Michael Gertz
Institut für Informatik, Universität Hannover, Lange Laube 22, D-30159 Hannover, Germany
{ul|mg}@informatik.uni-hannover.de

Gunter Saake
Fakultät für Informatik, TU Magdeburg, Universitätsplatz 2, D-39106 Magdeburg, Germany
saake@iti.cs.tu-magdeburg.de

1 Introduction

Database specification should include not only the static structure of a database system, but also its dynamic behaviour. In order to determine *admissible* sequences of states *dynamic integrity constraints* may be utilized. They give conditions on state transitions as well as long-term relations between database states in a descriptive manner. Complementarily, predefined *transactions,* i.e. basic elements of application programs, or ad-hoc transactions completed by *triggers* induce *executable* state sequences in an operational manner.

We have investigated how to monitor dynamic constraints without looking at entire state sequences, i.e. database histories, but considering only single state transitions, as it is usual in monitoring. Such monitoring can be achieved either by a universal (application-independent) monitor algorithm, or by (application-specific and thus more efficient) transactions or triggers into which the constraints are transformed and specialized during database design.

In our approach, dynamic integrity constraints are expressed by formulae of temporal logic (sect. 2). We have developed procedures to construct from such formulae so-called *transition graphs,* whose paths correspond to admissible state sequences (section 3). On the one hand, these graphs can control execution of a monitor that reduces analysis of state sequences to tests on state transitions. On the other hand, these graphs can be transformed systematically into refined pre-/postconditions of transactions or into triggers, such that each executable sequence becomes admissible (section 4). Such transformations can partially be supported by an environment for integrity-centered database design.

2 Dynamic Integrity Constraints

Static integrity constraints describe properties of database *states;* they restrict the set of possible states to those states which are admissible with respect to the constraints. To express such constraints, it is usual to take formulae of first-order predicate logic or related calculi.

In order to specify admissibility of state *sequences* by *dynamic* constraints, we make use of temporal logic that extends predicate logic by special operators relating states within sequences [LEG85, LS87, Lip89, Lip90] (compare also references there to other work on database specification using temporal logic). *Temporal formulae* are built from nontemporal formulae by iteratively applying logical connectives and

- a **currently** operator referring to the current state,

- a **next** operator referring to the next state,
- temporal quantifiers **always** and **sometime** referring to every or some state in the future,
- and temporal bounds **from**, **after**, **before**, and **until** which limit the range of temporal quantifiers by start or end conditions

The nontemporal subformulae may contain quantifiers ($\forall, \exists$) over all/some objects existing at a given state. Temporal formulae are interpreted in complete state sequences over all possible objects or data. Unlike [Cho92], we restrict the usage of object quantifications to these cases. To allow nesting of temporal operators, nontemporal subformulae are evaluated in single states, and temporal subformulae in tail subsequences.

Ordinary constraints, however, affect only objects which exist in the database, and only during their time of existence. Since objects may be inserted and deleted during database runtime, their existence intervals form – often finite – subsequences of the complete database state sequence. To deal with such *existence-restricted* constraints, formulae are interpretable in finite sequences as well, including the eventually empty tail sequence. A corresponding modification of the logic was given in [LZ91]; an alternative approach utilizing two kinds of **next**-operators can be found in [Saa91].

Example 1: The following dynamic constraint refers to an automobile registration database. It specifies that a newly produced car is not registered initially, but then it must be registered sometime in the same or next year (with its manufacturer as first owner), whereupon it remains registered:

> **constraint** DYC_REG: **during-existence**(c: CAR):
> **currently** (**not** c.registered **and** c.year-of-production = current-year)
> **and sometime** c.registered **before** current-year>c.year-of-production+1
> **and from** c.registered (**currently** c.owner=c.manuf **and always** c.registered).

3 Transition Graphs

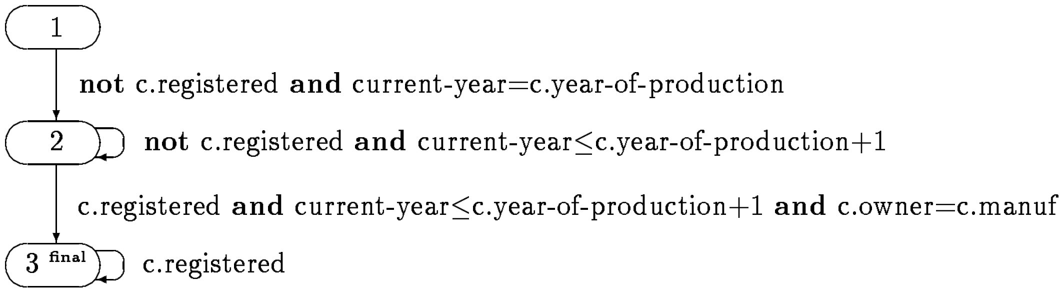
The essential means of monitoring dynamic integrity constraints on state transitions are *transition graphs.* Such a transition graph $T = \langle V, E, F, \nu, \eta, v_0\rangle$ for a temporal formula φ consists of

- a directed graph $\langle V, E\rangle$ having a finite set V of nodes and a set $E \subseteq V \times V$ of edges,
- a set of *final nodes* $F \subseteq V$,
- a *node labelling* ν with temporal formulae,
- an *edge labelling* η with nontemporal formulae,
- and an *initial node* $v_0 \in V$ labelled with φ.

A transition graph can be used to analyze a state sequence by searching for a corresponding path whose edge labels successively hold in the states of the sequence. For that, the nodes which have been reached at each moment must be *marked.* Then a state sequence is *accepted* up to a current state iff there is a node marked after passing that state; a finite state sequence is *completely accepted* iff a final node is marked after the last state.

To guarantee that accepted state sequences correspond to admissible sequences with respect to the constraint, the nodes must be labelled by temporal formulae indicating what remains to be monitored in the future (as it is the case for the initial node). The edges, however, are labelled by nontemporal formulae which have to be checked in the next state such that the right target node gets marked. Additionally, the labels of final nodes must hold in the empty sequence. To monitor existence-restricted constraints, such a “correct” transition graph is needed for the constraint body, and it is applied to check admissibility of the existence interval for every database object.

Example 2: The following transition graph is correct for the constraint of example 1; the node labels are 1: the constraint body itself, 2: the tail formula **sometime** ..., and 3: **always** c.registered.



Obviously, the graph reflects the admissible object life-cycles with respect to the dynamic constraint. The nodes correspond to *situations* in such life-cycles, the edges give the (changing) static conditions under which transitions are allowed. In situation 2, the car c is waiting for timely registration; there is no transition for the case that the deadline is exceeded, i.e. the constraint would be violated. Only the final situation 3 allows a deletion of the object from the database.

Since in general an object may reach different situations throughout such a life-cycle, a monitor has to keep track of the marked nodes for each object (or object combination) involved in the constraint. In a deterministic graph, at most one node is marked per object. Thus the database structures must be extended to store object situations. Such markings seem to be the minimal historical information needed for transitional monitoring. Whereas the approach in [Cho92] uses object sets for all temporal subformulae of the given constraint, our approach stores object sets only for the formulae corresponding to the nodes. To avoid infinitely many object/data combinations in the presence of data variables, there are techniques for finite representation as described in [HS91].

Influenced by work on deciding satisfiability of propositional-temporal formulae like in [MW84], we have developed procedures to construct strongly correct transition graphs from temporal formulae. The algorithms in [LS87, SL87, Saa91] derive graphs by iteratively transforming formulae into a disjunctive normal form corresponding to the correctness condition above. Another algorithm [LF88, Lip89] constructs deterministic graphs by composing graphs for subformulae in a bottom-up way corresponding to the formula structure; [LZ91] gives the additions to distinguish final nodes.

In [SS92] transition graphs have been utilized for monitoring past-oriented temporal formulae, basically by reversing edges of graphs constructed for analogous future-oriented formulae. Currently we are working on transition graphs for formulae with mixed usage of past and future operators.

4 Transformation into Transactions or Triggers

In order to prepare efficient monitoring already at database design time, the processing of transition graphs (checking edge conditions, storing marked nodes) can be transformed systematically into additional pre-/postconditions of transactions such that each state sequence induced by transactions becomes admissible, too.

Example 3: For the database above, a transaction 'register' might intuitively be specified as:

> **transaction** register (c: CAR):
> **pre** **not** c.registered **post** c.registered.

The transition graph above, however, requires 'register' to behave as follows, since it can fit only the transition from situation 2 to 3:

> **transaction** register (c: CAR):
> **pre** c.situation=2 **post** c.registered **and** c.owner=c.manuf **and** c.situation=3.

To specialize transformation results for the given transactions (as already done in this example), we have developed appropriate constraint simplification techniques: Classic simplification relies on assuming that a (static) constraint is valid before executing a transaction and on checking the same constraint after the transaction. Such techniques utilize that many constraint parts and instances remain invariant and thus need not be checked explicitly after the transaction. For dynamic constraints and transition graphs, the disjunction of ingoing edge labels can be assumed as a precondition in each situation instead. For a large class of constraints, namely those having so-called iteration-invariant transition graphs, invariance of these preconditions again leads to considerable simplifications. But even partial invariants may be erased, if the condition to be checked afterwards is different from the condition before. More application-specific simplifications are, of course, left to the designer. Rules for these transformations and simplifications have been introduced in [Lip86, Lip89, Lip90].

In the case that ad-hoc transactions (arbitrary unforeseen sequences of basic database operations like insertions and deletions) may occur, integrity maintaining triggers are needed instead.

Example 4: Situation 2 in the transition graph above gives rise to the following trigger for checking outgoing edges (written in an SQL-like style):

trigger on CAR **when updated**(registered):
  if exists (**select** * **from** CAR.updated_old c_old, CAR.updated_new c_new
    where c_old.situation=2 **and** c_old.serialno=c_new.serialno
    and (current-year>c_new.year-of-production+1
        or (c_new.registered **and not** c_new.owner=c_new.manuf)))
  then rollback.

('CAR.updated_old/new' denote tables with the updated CAR tuples before/after updating.) Another trigger must care for updating the 'situation' attribute, if no rollback has occurred before:

trigger on CAR **when updated**(registered):
  update CAR c **set** c.situation=3
  where exists (**select** * **from** CAR.updated_old c_old
    where c_old.situation=2 **and** c_old.serialno=c.serialno **and** c.registered)

In [GL93], we have given rules for how triggers for checking transitional conditions (like in the example) and triggers for actively updating situations can be derived at design time for every critical database operation and every situation.

5 Outlook

We consider the specification and transformation of integrity constraints into integrity maintaining mechanisms to be a central task in the database design process. Since such transformations require a lot of administration for combining, refining, and revising elements of a database schema like static and dynamic integrity constraints, transactions and triggers, a computer-aided design environment is strongly needed. We are developing a tool box called ICE ("Integrity-Centered Environment") that shall support dynamic integrity constraints in particular [GL94]. In addition to the construction of transition graphs from temporal formulae, transformations of these graphs into transactions and triggers are the main lines of the project. Tests for invariants and automatic detection of simplifications are subjects of future extensions.

Currently, we are investigating how to specify active *repairing* of constraint violations in order to extend the derivation of triggers [Ger94]. These efforts shall be generalized to dynamic constraints within the transitional monitoring approach as presented in this paper.

References

[Cho92] J. Chomicki. History-less checking of dynamic integrity constraints. In *Proc. 8th Int. Conf. on Data Engineering (ICDE'92)*, IEEE Computer Society Press, 1992, 557–564.

[Ger94] M. Gertz. Specifying reactive integrity control for active databases. In *Proc. 4th Int. Workshop on Research Issues in Data Engineering: Active Database Systems (RIDE-ADS'94)*, IEEE Computer Society Press, 1994, 62–70. (∗)

[GL93] M. Gertz and U.W. Lipeck. Deriving integrity maintaining triggers from transition graphs. In *Proc. 9th Int. Conf. on Data Engin. (ICDE'93)*, IEEE Comp. Soc. Press, 1993, 22-29. (∗)

[GL94] M. Gertz and U.W. Lipeck. ICE: An environment for integrity-centered database design, Informatik-Bericht 02/94, Universität Hannover 1994. (∗)

[HS91] K. Hülsmann and G. Saake: Theoretical foundations of handling large substitution sets in temporal integrity monitoring. *Acta Informatica*, 28:365–407, 1991.

[LEG85] U.W. Lipeck, H.-D. Ehrich, and M. Gogolla. Specifying admissibility of dynamic database behaviour using temporal logic. In A. Sernadas, J. Bubenko, and A. Olive (eds.), *Theoretical and Formal Aspects of Information Systems (Proc. IFIP Work. Conf. TFAIS '85)*, North-Holland Publ. Co., Amsterdam 1985, 145–157.

[LF88] U.W. Lipeck and D.S. Feng. Construction of deterministic transition graphs from dynamic integrity constraints. In J. van Leeuwen (ed.), *Graph-Theoretic Concepts in Computer Science (Proc. Int. Workshop WG'88)*, LNCS 344, Springer-Verlag, Berlin 1988, 166–179.

[Lip86] U.W. Lipeck. Stepwise specification of dynamic database behaviour. In C. Zaniolo (ed.), *Proc. SIGMOD'86 Int. Conf.*, ACM, 1986, 387–397.

[Lip89] U.W. Lipeck. *Dynamische Integrität von Datenbanken: Grundlagen der Spezifikation und Überwachung* (Dynamic Database Integrity: Foundations of Specification and Monitoring, in German). Informatik-Fachberichte 209. Springer-Verlag, Berlin 1989, 140pp.

[Lip90] U.W. Lipeck. Transformation of dynamic integrity constraints into transaction specifications. *Theoretical Computer Science*, 76:115–142, 1990.

[LS87] U.W. Lipeck and G. Saake. Monitoring dynamic integrity constraints based on temporal logic. *Information Systems*, 12(3):255–269, 1987.

[LZ91] U.W. Lipeck and H. Zhou. Monitoring dynamic integrity constraints on finite state sequences and existence intervals. In J. Göers, A. Heuer, and G. Saake (eds.), *Proc. 3rd Workshop on Foundations of Models and Languages for Data and Objects*, Informatik-Bericht 91/3, TU Clausthal 1991, 115–130. (∗)

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Progr. Lang. Systems*, 6(1):78–85, 1984.

[Saa91] G. Saake. Descriptive specification of database object behaviour. *Data & Knowledge Engineering*, 6(1):47–73, 1991.

[SL87] G. Saake and U.W. Lipeck. Foundations of temporal integrity monitoring. In C. Rolland, M. Leonard, and F. Bodart (eds.), *Temporal Aspects in Information Systems (Proc. IFIP Work. Conf. TAIS'87)*, North-Holland Publ. Co., Amsterdam 1988, 235–249.

[SS92] S. Schwiderski and G. Saake. Monitoring temporal permissions using partially evaluated transition graphs. In U.W. Lipeck and B. Thalheim, editors, *Modelling Database Dynamics (Proc. Int. Workshop, Volkse 1992)*, Springer-Verlag, London 1993, 196–220.

The papers marked by (∗) are available on the ftp-server "wega.informatik.uni-hannover.de".