Expert Systems with Applications 37 (2010) 1374–1390

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A tool for addressing the ramification problem in spatial databases: A solution implemented in SQL

Nikos Papadakis *, Yannis Christodoulou

Department of Sciences, Techological Educational Institute of Crete, Greece

ARTICLE INFO

Keywords:
Ramification problem
Spatial databases
Common sense reasoning
Knowledge representation and reasoning
Software engineering

ABSTRACT

In this paper, we study the ramification problem in the setting of spatial databases. Standard solutions from the literature on reasoning about action are inadequate because they cannot capture integrity constraints in spatial data. In this paper, we provide a solution to the ramification problem based on situation calculus. We present a tool that connects the theoretical results to practical considerations, by producing the appropriate SQL commands in order to address the ramification problem in spatial databases.

1. Introduction

Spatial databases can be mainly implemented in Geographic and Information Systems for the representation-modeling of space and the representation of objects in that space (Chen & Gong, 1998; Oracle, 2003). They can also be implemented in multimedia (video editing), networks (representation of networks, hubs).

The various types of geometrical shapes, according to their complexity, do not allow for an immediate and flexible resolution of problems. The operators, in order to be able to provide some comparable properties in these shapes, introduce the concept of the bounding box. The bounding box encircles the geometrical shape in a box, whose area is the less possible one, thus providing an easier way to handle shapes. The role of the bounding box and the use of operators like window query, point query, intersection query, enclosure query, containment query and adjacent query are very important tools for the handling of shapes (Kiiveri, 1997).

The use of spatial operators in a spatial database can be a difficult process both in updating and finding data. This is due to the complexity a shape may have, but also because of the possible changes that may occur. Under these conditions, we can classify the operator methods in three broad categories: the one-dimensional access methods, area access methods and the point access methods. The multi-dimensional access methods mentioned need alternative ways to access data, so as to enable the comparison of geometrical representations in space between *k*-dimensional and one-dimensional representations.

The following sections will refer to cases of inconsistency that may occur during an update of the spatial data. In particular, we assume that spatial data is stored as a sequence of points (each one adjacent with its next) in the two-dimensional space and that a change in a shape occurs when the coordinates of a point change. In the case where such a change takes place, there is the possibility that an overlap may arise between two geometrical shapes. This is an inconsistency that is faced in this paper. More specifically, we propose an algorithm, which produces SQL transaction code. When a change in the Database takes place and an inconsistency arises, this code is responsible to change the adjacent geometrical shapes, so as the overlap between them is removed. The change of the first shape is the direct effect of the execution of a transaction, while the change of the rest shapes is the indirect effect of the execution of a transaction and takes place to maintain the Not-Overlap constraint. The description of the indirect effects of a transaction, when integrity constraints exist, is the ramification problem.

The guarantee of consistency of data that is stored in a database is very important and difficult problem. The consistency of data is determined by the satisfaction of the integrity constraints (Andrea Rodriguez, 2004; Kalum, Li, & Wijeratne, 2006; Kiiveri, 1997; Oracle, 2003; Scott, 1994) in the different database states (situations). A database state is considered valid (consistent) only when all integrity constraints are satisfied. When a transaction is executed, the context of the database is modified. In the new situation (which includes the direct effects of the transactions) the database may be inconsistent because some integrity constraints are not satisfied. Thus, it is necessary to produce some additional effects (indirect effects) to satisfy the integrity constraints. We can assume that an atomic transaction is an action.

In this context, the *ramification problem* (McCarthy & Hayes, 1969) is concerned with the indirect effects of actions in the presence of constraints. The ramification problem is of great importance to the database systems. Database users and designers may not know exactly all the indirect effects of their transactions. This

* Corresponding author.
E-mail addresses: npapadak@cs.teicrete.gr (N. Papadakis), jchrist@csd.uoc.gr (Y. Christodoulou).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.06.110

means that the users/designers can execute a transaction which has as a result to violate the integrity constraints. The most obvious solution is to determine manually all the indirect effects. The problem with this solution is that in a large database with a large number of constraints and transactions, the indirect effects produced by the evaluation of transactions may be too many to be accounted for manually. Also, users/designers may not know all the indirect effects of their transactions. Assume database systems with many hundreds of transactions, many hundreds of integrity constraints and more than one designer. In such databases, with a large number of constraints and transactions, indirect effects may be too many to be manually discovered. Notice that the same sequence of transactions may have different indirect effects, if the context of the database in the start of the execution is different.

In the following section, we will describe the ramification problem in conventional and in spatial databases, explaining the difference between them and the reason that the solutions that are proposed for conventional databases cannot address the ramification problem in spatial databases. Next, in Section 3, we will describe the representation of spatial data in relational databases, as has been proposed. Then, in Section 4, we will explain the solution we propose, describing the concept and the algorithms. In Section 5 we will give a complete example of the execution of the tool (Java program), describing what input should be given and what the results will be like. After that, in Section 6, we will explain the architecture of the system. In Section 7, we will present a proof of the correctness of the systems, the complexity of our solution and some evaluation results from the tests we made.

2. The ramification problem in conventional and in spatial databases

2.1. The ramification problem in conventional databases

The ramification problem is a very hard problem that arises in robotics, software engineering and databases. We introduce this problem by means of an example. Suppose that we are interested in maintaining a database that describes a simple circuit (Fig. 1), which has two switches and one lamp. The circuit's behavior is described by the following integrity constraints:

$$up(s_1) \wedge up(s_2) \equiv light, \quad (1)$$
$$\neg up(s_1) \Rightarrow \neg light, \quad (2)$$
$$\neg up(s_2) \Rightarrow \neg light. \quad (3)$$

The first constraint implies that when the two switches are turned on, then the lamp is lit. The second and third constraints imply that if a switch is turned off, then the lamp must not be lit.

Action *toggle_switch* changes the situation of a switch as follows:

$$toggle\_switch(s) \rightarrow up(s) \quad if \quad \neg up(s),$$
$$toggle\_switch(s) \rightarrow \neg up(s) \quad if \quad up(s).$$

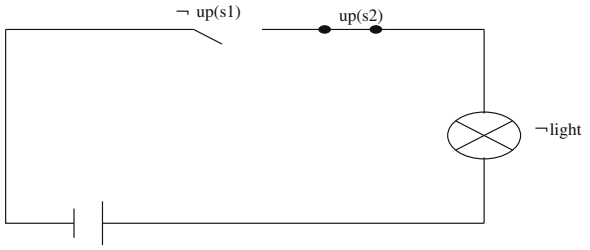

Fig. 1. Simple electric circuit.

The above propositions describe the direct effects of the action *toggle_switch*. A situation is consistent when it satisfies all the integrity constraints. Assume that the circuit is in the situation $S = \{\neg up(s_1), up(s_2), \neg light\}$. The situation S is consistent, because it satisfies all integrity constraints. Assume that we execute the action $toggle\_switch(s_1)$. This action has as a direct effect on the change of the state of switch s_1 from $\neg up(s_1)$ to $up(s_1)$. Now the situation of the circuit is $S_1 = \{up(s_1), up(s_2), \neg light\}$. This situation is inconsistent, because it violates the first integrity constraint. In order to reach a consistent situation we must change the situation S_1 in one of the following situations

$$S_2 = \{up(s_1), up(s_2), light\},$$
$$S_3 = \{up(s_1), \neg up(s_2), \neg light\}.$$

Both situations (S_2, S_3) are consistent because all the integrity constraints are satisfied. As we observe, the updates from $\neg light$ to $light$ in the situation S_2 or the change from $up(s_2$ to $\neg up(s_2)$ in the situation S_3 happen in order to produce a consistent situation and not as a direct effect of the action $toggle\_switch(s_1)$.[1] These are the indirect effects of the action $toggle\_switch(s_1)$.

The reasonable conclusion is that the lamp must be lit. In order to infer this conclusion we must determine which could be the indirect effects of the action. We discuss this in Section 4.

Notice that the indirect effects exist because of the presence of the integrity constraints. The *ramification problem* refers to the concise description of the indirect effects of an action in the presence of constraints.

Several ways of addressing the ramification problem have been suggested in literature. The majority of them are based on the situation (McCarthy & Hayes, 1969) and the event calculus.

2.2. Basic terminology in situation calculus

- All predicates and functions whose true value changes from one world state to another are called *fluents*.
- One possible evolution of the world is a sequence of actions and is represented by a first-order term, called *situation*.[2]
- An action can change the value of some fluents.
- A situation is consistent when all the integrity constraints are satisfied.
- An action occurs in a situation and yields another situation as a result.
- The binary function *do* is defined as follows, with $do(a, s)$ denoting the situation that will result from the execution of action a in the situation s.
- An action can be executed if it satisfies *some conditions*. We call these preconditions, as we have already mentioned. The binary predicate *Poss* declares whether a precondition holds. When the predicate $Poss(a, s)$ is true then the action a can be executed in the situation s.

Among the simplest solutions proposed are those which are based on the minimal change approach (Ginsberg & Smith, 1988; Winslett, 1988). These solutions suggest that when an action occurs in a situation S one needs to find the consistent situation S′, which has the fewer changes from the situation S. For instance, consider the modeling of a simple circuit as an example. Assume situation $S = \{\neg up(s1), up(s2), \neg light\}$. The action toggle – switch$(up(s1))$ changes the situation of the circuit to $S' = \{up(s1), up(s2), \neg light\}$, which is inconsistent. There are two consistent situations $S1 = \{up(s1), up(s2), light\}$ and $S2 = \{up(s1), \neg up(s2), \neg light\}$. It is sensible to light the lamp, whereas turning off the

[1] The direct effect of its action is $up(s_1)$.
[2] A more simple situation contains the value of fluents.