Automatic Verification of Transactions on an Object-Oriented Database∗

David Spelt1, Herman Balsters2

University of Twente, Enschede, The Netherlands

Abstract. In the context of the object-oriented data model, a compile-time approach is given that provides for a significant reduction of the amount of run-time transaction overhead due to integrity constraint checking. The higher-order logic Isabelle theorem prover is used to automatically prove which constraints might, or might not be violated by a given transaction in a manner analogous to the one used by Sheard and Stemple (1989) for the relational data model. A prototype transaction verification tool has been implemented, which automates the semantic mappings and generates proof goals for Isabelle. Test results are discussed to illustrate the effectiveness of our approach.

Keywords: object-oriented databases, transaction semantics, transaction verification

1 Introduction

Static integrity constraints are essential in mission-critical application domains, where one wants to offer integrity preserving update operations to clients. One way to enforce database integrity is by testing at run-time those constraints that are possibly violated by a transaction before allowing the transaction to commit. Various techniques, surveyed in [1], have been proposed to optimize such a test for a limited class of simple constraints, such as key and referential integrity. But committing complex transactions on large amounts of data becomes increasingly difficult if the constraint language is extended to include full first-order logic formula with bounded quantifications over arbitrary collection types.

A second approach towards integrity maintenance aims at a compile-time reduction of the amount of run-time transaction overhead due to integrity constraint checking. This approach was first introduced by Sheard & Stemple ([2]) for the relational data model. It uses a theorem prover to verify that a transaction will never raise an integrity conflict, provided that the database was in a consistent state before the transaction was executed. Transactions are complex updates involving multiple relations, whereas the constraint language includes quantifications and aggregate constructs. These are related to expressions in higher-order logic for automatic proof assistance.

Another related compile-time approach is proposed in [3,4]. It exploits several techniques of abstract interpretation for the task of compile-time transaction verification in an O2 database system extended with a notion of declarative integrity.

∗ Our e-mail addresses are: [spelt,balsters]@cs.utwente.nl
constraints. Their analysis starts with a simple compilation technique. It identifies those constraints that will certainly not be affected by a transaction because different attributes or even different class extents are accessed. For instance, a transaction that only changes the age of a person can never violate a constraint that does not access this field. Recently, in [5], this approach was augmented with a second, more powerful analysis. For each combination of transaction and constraint that could not be proved safe in the first step, it applies Dijkstra’s concept of predicate transformer. This yields a simple first-order logic formula which is automatically verified by a theorem prover. A major difference between this approach and the one presented in [2] is that the latter fully exploits the (denotational) semantics of a database specification, whereas the former only takes some global abstract properties of the semantics into account.

The above compile-time strategies supplement the existing run-time techniques: they can filter out the set of relevant constraint checks, thus allowing the run-time optimizer to focus on a restricted set of constraint predicates that could not be proved safe at compile-time.

Our work extends the work of [2]. Rather than a relational model, it uses a powerful specification and verification environment for object-oriented databases with transactions and integrity constraints. The specification framework uses TM [6,7], a typed formal specification language based on the well-known ideas of Cardelli [8], extended with logic formalism and sets [9]. The declarative flavour of this language permits compile-time transaction verification using a theorem prover, while retaining ODMG compliancy [10].

The verification framework uses higher-order logic. Specifications in TM are automatically mapped to expressions in higher-order logic (HOL), such that the consistency requirements can be given as input to the higher-order logic Isabelle theorem prover [11] for automatic proof assistance.

The rest of this paper is organized as follows. Section 2 introduces the TM data model and gives an example of a job agency service. This example was also studied by [2]. In Section 3, the Isabelle theorem prover is introduced, along with a motivation for why the HOL theory is used. We then proceed with a sketch of how TM database schemas are represented in HOL. Section 5, discusses transaction verification using the Isabelle theorem prover and shows how standard Isabelle tactics can be used to implement a transaction verifier in HOL. In Section 6, we then compare our work to several related compile-time approaches. We finish by stating our conclusions and give directions for future work.

2 Database Definition in TM

This section introduces the TM data model, using the job agency service example [2], which is re-engineered to the object-oriented data model. Objects in the database include people who apply for, and are placed with certain jobs. A job can be shared by multiple employees. Skills are required to execute jobs, and people should have abilities that satisfy these requirements. TM extends the ODMG interface definitions with formal specifications of methods, transactions