On Transactions in Logic Programming Languages

Stefan Böttcher
Daimler-Benz Forschung und Technik
Eberhard-Finckh-Straße 11
D - 7900 Ulm-Böfingen , Germany[1]

Introduction

The extension of logic programming languages by database access has up to now concentrated on read access of logic programs to external relational databases. Additionally, proposals for the integration of database updates into a logic programming language have been made, e.g. [Manchanda88]. However, whenever many users access a database, the accesses of these users have to be synchronized too. Therefore, a transaction concept is needed. We present a proposal how to embed a suitable transaction concept in a logic programming language. The basic idea is to introduce transactions as a special kind of deterministic relations in a logic programming language.

Requirements

In order to support the synchronization of database accesses, conventional database programming languages provide a procedural transaction concept. However, in the case of an integrated logic programming and database access language (as e.g. PROTOS-L [Beierle90], [Böttcher90]) the transaction concept has to be integrated into the logic programming language, i.e. persistence and atomicity of transactions have to be integrated with backtracking. In order to provide transaction *persistence*, backtracking should be prevented from returning into commited transactions. *Atomic* transaction execution requires to undo write operations in the case that a transaction can not be commited.

Additionally, database access should be allowed only in the scope of transactions for safety reasons and because transaction synchronization provided by the database system can synchronize the database accesses of several users correctly. In order to keep a logic-based database programming language powerful enough, the programmer should have the possibility to program several transactions in a single logic program.

The suggested solution

We suggest to regard transactions as a special kind of relations, i.e. the logic programming language should provide several types of relations, one being a transaction.

Then transaction persistence is acchieved, if the transaction is a deterministic relation. Although transaction persistence could also be acchieved by the use of the cut operation, we suggest that the logic programming language contains a language construct which explicitly declares a relation to be a transaction. For example, a transaction declaration may consist of a set of rules preceeded by the keyword *trans*, whereas a relation declaration may consist of a set of rules preceeded by the keyword *rel*.

Transaction atomicity is acchieved in a natural way, if write operations to databases are backtrackable. In this case, all modification operations done inside a transaction are undone, if the transaction fails, i.e. transactions that fail to commit leave the database without changing the database state.

The basic idea for the implementation of a transaction *trans* t(...) is to access the database system transaction management services for transaction begin, commit and abort by built-ins tBegin, tCommit and tAbort and to use these built-ins in the following way. Each transaction *trans* t(...) can be implemented by an ordinary relation *rel* tBody(...) containing exactly the same rules as given in the declaration of the transaction t(...) by:

```
t(...) :- tBegin, tBody(...), tCommit, ! .
t(...) :- tAbort, fail.
```

The cut after tCommit prohibits backtracking to undo parts of a committed transaction, i.e. the committed transaction is persistent. The goal tAbort guarantees transaction atomicity, if the execution of tBody fails.

Since the implementation of t(...) by tBody(...) and the built-ins is independent of the code of the transaction t, the sketched implementation is generic and can be used e.g. as a first compilation step in a compiler for logic programming languages.

References

[Beierle90] C. Beierle. Types, modules and databases in the logic programming language PROTOS-L. In K.-H. Bläsius, U. Hedtstück, and C.R. Rollinger, editors: Sorts and Types for Artificial Intelligence, Springer, Berlin [et.al.], 1990.

[Böttcher90] S. Böttcher. A tool kit for knowledge based production planning systems. In A.M. Tjoa, W. Wagner, editors: Proc. Intern. Conference on Database and Expert System Applications, Vienna, Austria, 1990.

[Manchanda88] S. Manchanda and S. Warren. A logic-based language for database updates. In J. Minker, editor: Foundations of deductive databases and logic programming, Morgan Kaufmann, Los Altos, 1988.

[1]The work reported here has been done at IBM Deutschland GmbH, Scientific Center, Stuttgart as part of the EUREKA Project PROTOS (EU56).