Development and Programming of Deductive Databases with PROTOS-L

Stefan Böttcher
IBM Deutschland GmbH
Scientific Center
Institute for Knowledge Based Systems
P.O.Box 80 08 80
D – 7000 Stuttgart 80
West Germany *

Abstract

This paper presents PROTOS-L, a logic programming language which embeds a module concept, provides read access to external databases, and combines order-sorted types with polymorphism. The presentation focusses on the development and programming of deductive databases within PROTOS-L. From this viewpoint, PROTOS-L is similar to DATALOG embedded in a typed logic programming language. In order to illustrate the development of deductive databases, we show how a small fragment of a travel information system can be developed within PROTOS-L.

1 Introduction

PROTOS-L is a logic programming language extended by the following features: access to relational databases, a module concept, a type concept including subtypes and polymorphism. Furthermore, PROTOS-L supports the programming of deductive databases.

The module concept [Beierle, 1989], [Zeller, 1990] hides the implementation of the exported predicates in the body of a module. The body of a module is either a program body or a database body depending on the kind of implementation, i.e. depending on whether the predicates of a module are implemented by a set of facts and rules in the embedding logic programming language or the predicates are implemented by a set of views on database relations. At the interface of a module it is transparent whether the body is a program body or a database body.

The PROTOS-L type system which is described e.g. in [Beierle and Böttcher, 1989] supports subtypes and polymorphism, i.e. type variables. It is derived from the type system of TEL [Smolka, 1988]. PROTOS-L types are not only used by the compiler for type checking, but also by the PROTOS Abstract Machine (PAM) for computations on types in order to reduce the search space.

The rest of the paper is organized as follows. Section 2 describes the module concept of PROTOS-L. Section 3 presents how PROTOS-L embeds access to external databases and supports programming of deductive databases. Section 4 outlines how the knowledge of multiple relational and deductive databases is integrated with private knowledge within PROTOS-L.

2 The PROTOS-L module concept

This section describes how the module concept of PROTOS-L supports knowledge structuring and rapid prototyping. The basic idea is to use the modules as knowledge packages. Furthermore, it describes how the module concept makes the access to relational databases and deductive databases transparent to the user of a module.

2.1 Knowledge structuring and knowledge encapsulation

For the purpose of knowledge structuring and knowledge encapsulation the programming language PROTOS-L provides a module concept. The module concept of PROTOS-L integrates ideas from the module concepts of Modula-2 [Wirth, 1983], TEL [Smolka, 1988] and DBPL [Eckhardt *et al.*, 1985], [Böttcher, 1989]. While the *interface* of a module specifies only the predicate names and the types of their arguments, the *body* of a module specifies their implementation. The user of a module only sees its interface, however, the implementor of a module ad-

*The research reported here has been carried out within the international EUREKA project PROTOS (EU56): Prolog Tools for Building Expert Systems. Project partners are BIM, IBM Stuttgart, Sandoz AG, Schweizerische Bankgesellschaft, University of Dortmund and University of Oldenburg.

```
interface flight_info .
  rel  direct_flights :  ?string x  ?string x  ?int x      ?int     ?string .
  %                      from        to         departure  arrival  airline
  % Which flights of which airline are scheduled for which time ?


  rel  prices :  ?string  ?string  ?string  ?int .
  %              from     to       airline  price
  % What are the ticket prices ?
...
endinterface.
```

Figure 1: Interface of a PROTOS-L module

ditionally writes its body.[1]

Hence, the concept of an interface supports knowledge encapsulation in PROTOS-L, whereas the distribution of rules into several separate modules supports knowledge structuring.

For example: Figure 1 shows a part of an interface of a travel information system. The predicates of the module compute which flights of which airline are scheduled for which time and which prices you have to pay for a flight of which airline. It is hidden how the predicates are implemented.

Furthermore, the module concept supports separate compilation and thereby it supports the maintenance of large program systems. This is done as in Modula-2: interface and body of a module are compiled separately. Since compilation units import only from interfaces but never from bodies, the body of any module can be recompiled independently of the rest of a module system.

2.2 Transparent access to relational and deductive databases

Like other implementation details, access to an external database shall be transparent for the user of a module. Therefore, in an interface of a module it is not visible whether or not the corresponding implementation of the body accesses an external database.

To be more precise, PROTOS-L offers two kinds of bodies of a module: *program bodies* and *database bodies*. Program bodies support logic programming with backtracking and database bodies support access to relational databases and the programming of deductive databases.

[1] In the body of a module every predicate exported by the corresponding interface has to be implemented. The implementation of a predicate in the body of a module has to be type compatible with its definition in the corresponding interface, i.e. the number and the types of the arguments of the predicates have to be unique.

Program bodies can import predicates from other modules, but database bodies can not, for the following reason. Program bodies use backtracking whereas database bodies use set-oriented evaluation strategies (c.f. section 3.2). PROTOS-L supports backtracking on top of set-oriented retrieval by allowing program bodies to import predicates from modules which are implemented by database bodies. However, set-oriented retrieval on top of backtracking is undesired. This undesired combination of evaluation strategies is prevented, because database bodies can not import predicates from program bodies. As a consequence, PROTOS-L does not allow any imports into database bodies, because at the interface of a module it is not visible which kind of body implements the module.

Example continued:

The predicates direct_flights and prices could be implemented either as in Prolog by a set of facts and rules outlined in a program body or as described in the third section by external databases. At the interface it is not visible how the predicates are implemented.

2.3 How PROTOS-L supports rapid prototyping

The module concept can be used to support rapid prototyping and top-down system implementation as well as bottom-up system implementation. Bottom-up system evaluation is even better supported than in many conventional programming languages, because in PROTOS-L the programmer can interactively query every predicate defined in any module, provided that the module and all its submodules are implemented. The basic idea how to support top-down system implementation is as follows. In order to check whether an upper module works well, the programmer can use dummy submodules which

```
database_body flight_info using flight_DB .
 rel    direct_flights :  ?string x  ?string x  ?int x     ?int     ?string .
 %                        from       to         departure  arrival  airline
 % Which flights of which airline are scheduled for which time ?

 dbrel  direct_flights    is Flight_Rel( FROM , TO , DEPARTURE , ARRIVAL , AIRLINE ) .

 rel    prices :  ?string  ?string  ?string  ?int .
 %                from     to       airline  price
 % What are the ticket prices ?

 dbrel  prices    is Price_Rel( FROM , TO , AIRLINE , PRICE ) .
 ...
endmodule.
```

Figure 2: Database body of a PROTOS-L module corresponding to the interface of figure 1

list only a few facts for rapid prototyping purposes. Later, the dummy submodules can be exchanged by modules which implement the desired predicates without changing the interfaces of the module.

Example continued:

In the rapid prototyping phase the predicates direct_flights and prices could be implemented in a program body, e.g. by listing some example facts. When the prototyping phase is completed successfully, the program body can be substituted by the database body described in the example of section 3.1.

3 Relational and deductive databases

This section describes how database access is embedded in the programming language PROTOS-L, i.e. how relational database access is expressed within database bodies and how deductive databases are programmed in PROTOS-L. The basic idea is to give the PROTOS-L programmer a *uniform* high-level database programming language. The mismatch of other language integrations, e.g. the integration of SQL into C, should be avoided.

3.1 Accessing relational databases

If the implementation of a module is a database body, then there is expected to be a corresponding database.[3] Furthermore, for every database relation declared in the database body there has to exist a corresponding database relation in the database schema and the argument types of the declared relation have to be the same as the types of the attributes of the underlying database relation [Böttcher and Beierle, 1989]. This is why the argument types of a declared relation are restricted to the attribute types supported by the underlying database system, i.e. to integer and string.

Example continued:

The interface given in figure 1 can be implemented by the database body outlined in figure 2. This database body requires that there are at least two relations in the database flight_DB: Flight_Rel and Price_Rel. Furthermore, Flight_Rel must have at least the five attributes: FROM, TO and AIRLINE of type string, and DEPARTURE and ARRIVAL of type integer. Similarly, Price_Rel must have at least the attributes PRICE of type integer and FROM, TO and AIRLINE of type string. This is checked by the PROTOS-L system at the time when the module flight_info is opened.

3.2 Programming a deductive database

The database module is also used in order to program a deductive database. A PROTOS-L database body may contain function free database rules in order to implement predicates specified in the module's interface. A function free database rule consists of a head and a number of goals each of which does not contain a function symbol. Rules in database bodies may contain the following kinds of goals (the syntax of rules for deductive databases is described in [Böttcher, 1990]):

database relation goals e.g.
direct_flight(From,Change,Departure,Arrive,Airline)

[3] In the database body the logical name of the database is specified. The corresponding database is expected to be in a directory the path of which is connected to this logical name.

```
database_body flight_info using flight_DB .

...

 rel  flight_connections :  int x   ?string x  ?string x  ?int x       ?int .
 %                          steps   from       to         departure    arrival
 % Which flight connections are possible for which time ?

      flight_connections( 1 , From , To , Departure , Arrival )
                  <-- direct_flight( From , To , Departure , Arrival , Airline ) .

      flight_connections( Steps , From , To , Departure , Arrival )
                  <-- Steps > 1
                    & direct_flight( From , Change , Departure , Arrive_at_Change , Airline )
                    & flight_connections( Steps - 1 , From , Change , Depart_from_Change , Arrival )
                    & Arrive_at_Change + 100 < Depart_from_Change .

endmodule.
```

Figure 3: A deductive database programmed in a PROTOS-L database body

virtual relation goals e.g.
flight_connection(St,From,Ch,Depart_from_Ch,Arrive)

built-in goals e.g.
Steps > 1 ,
Arrive_at_Change + 100 < Depart_from_Change

Note that the PROTOS-L programmer may program recursive and non-recursive predicates in database bodies. For example, the database body given in figure 2 may additionally contain a non-recursive predicate flight_connections the implementation of which is shown in figure 3. The first rule states that a direct flight is a 1-step flight connection, whereas the second rule computes those n-step flight connections from direct flights and (n−1)-step connections that leave enough time to change the airplane (c.f. figure 3).

PROTOS-L supports the built-in goals $=$, $\neq$, $\leq$, $\geq$, $<$ and $>$ in order to compare arithmetic expressions and additionally the built-in goal like in order to compare strings. Arithmetic expressions may contain integer constants and variables, brackets and the operators + , — , * and // .

Since rules in program bodies and view definitions in database bodies are expressed in the same way, the PROTOS-L programmer has to learn only one single language for deductive databases and application programs. This avoids the mismatch of other integrations of database query languages into host programming languages, e.g. of the integration of SQL into C.

In contrast to the rules in program bodies which are evaluated by backtracking, the rules in database bodies are evaluated by set-oriented query evaluation strategies. The implementation of these strategies is described e.g. in [Meyer, 1989]. Set-oriented query evaluation strategies are especially advantageous, if the accessed data sets are large.

4 An integration of logic programming and deductive database programming

This section describes how PROTOS-L integrates logic programming and deductive database programming. It outlines how the module concept is used in order to choose the evaluation strategy of rules, how the knowledge of databases is integrated with private knowledge and how the knowledge of multiple databases is integrated into PROTOS-L application programs.

4.1 The evaluation strategy for rules

Whenever a rule uses facts that are stored in a database, the programmer has the choice to select an adequate evaluation strategy for this rule. Whenever the programmer assumes that there are many results of a rule R needed to solve a goal, he may prefer set-oriented evaluation of the rule R. In this case, he codes the rule R in a database body and the PROTOS-L system evaluates the rule set-oriented. On the other hand, if he assumes that there are only a few results of a rule R needed to solve a goal, he may prefer an evaluation by backtracking and use a cut at that place of a program, where no more answers to the rule R are needed. In this case, he codes the

```
rel  my_prices :  ?string  ?string  ?string  ?int .
%                 from     to       airline  price
% What are the ticket prices including special offers ?

     my_prices( 'New York' , 'Frankfurt' , 'Cheap_Air' , 300 ) .

     my_prices( From , To , Airline , Price )
          <--  prices( From , To , Airline , Price ) .
```

Figure 4: Integration of private knowledge with the knowledge of relational and deductive databases

same rule R in a program body and the PROTOS-L system evaluates the rule by backtracking.

Example continued:

If it is assumed that many calls of flight_connections are needed in order to solve a query (or rule) containing the goal flight_connections(...), then the predicate flight_connections is preferably implemented in a database body as shown in the last example, because the database body performs a set-oriented evaluation of the rule. However, if it is assumed that only a few solutions of flight_connections are needed in order to solve a traveller's query and therefore backtracking is preferred, then the rule may be implemented in another program body instead of the database body. Hence, whether a rule accessing database relations should be implemented in a database body or in a program body depends on the desired evaluation strategy for this rule.

4.2 Integrating private knowledge

Sometimes private knowledge stored in facts and rules in a PROTOS-L program body has to be integrated with knowledge stored in a relational database. One typical case is that private facts shall be added to the facts retrieved from a relational database. A different case which can be solved in the same way is that a private program contains some exceptions, i.e. that some data although derived from the database shall not be included in further inferencing.

Example continued:

Assume that the following private knowledge shall be added to the knowledge which can be retrieved from the database: "Cheap_Air offers 300 Dollar tickets for flights from New York to Frankfurt." This private knowledge can easily be integrated with the knowledge of the deductive database by implementing a predicate my_prices in a program body as outlined in figure 4.

4.3 Integrating the knowledge of multiple databases

PROTOS-L can integrate the knowledge of many databases within a single application program. According to the claim that knowledge structuring is supported by the module concept, every database (like every other knowledge package) is enclosed in its own module. Hence, every database needs its own database body. The information of several databases can be integrated within program bodies which import all the predicates they need from the database modules.

Example continued:

In the same way as the knowledge stored in the deductive database flight_DB is integrated with private data (this was shown in the example above), this knowledge can be integrated with the knowledge stored in other (deductive or relational) databases.

For example, a similar deductive database train_info could contain information about train connections. Then information contained in both deductive databases flight_info and train_info can be combined in a module travel_info which is outlined in figure 5. The predicate travel computes flight connections, train connections and such combined flight and train connections where the flights are taken first.

5 Summary and conclusion

PROTOS-L provides a module concept which supports data and knowledge structuring and integration. The module concept supports transparent database access by hiding the implementation of predicates from the user of a module. Additionally, the programming of deductive databases is supported by database bodies. Further, PROTOS-L supports the integration of multiple databases and private knowledge.

PROTOS-L offers set-oriented evaluation strategies for rules contained in database bodies and backtrack-

```
module travel_info .

imports flight_info , train_info .

 drel  travel :  int x    int x    ?string x  ?string x  ?int x      ?int .
 %               flights  trains   from       to         departure   arrival
 % Which connections are possible for which time ?

 travel( Flights , 0 , From , To , Departure , Arrival )
         <-- flight_connection( Flights , From , To , Departure , Arrival ) .

 travel( 0 , Trains , From , To , Departure , Arrival )
         <-- train_connection( Trains , From , To , Departure , Arrival ) .

 travel( Flights, Trains , From , To , Departure , Arrival )
         <-- flight_connection( Flights , From , Change , Departure , Arrive_at_Change )
            & train_connection( Trains , Change , To , Depart_from_Change , Arrival )
            & Arrive_at_Change + 100 < Depart_from_Change .
endmodule.
```

Figure 5: A module integrating the knowledge of two deductive databases

ing for rules contained in program bodies. Nevertheless, the PROTOS-L programmer has to learn only one single language because program body rules and database views are expressed in the same way. This avoids the mismatch of other integrations of programming languages and database languages.

In order to support rapid prototyping, PROTOS-L allows to submit queries interactively for every predicate in every module body and to exchange module bodies easily. Finally, the example showed how to implement fragments of a travel information system within PROTOS-L.

Acknowledgement

The programming language PROTOS-L was designed together with Christoph Beierle and Gert Smolka. I would like to thank them for their contributions.

References

[Beierle, 1989] 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-Verlag, Berlin, Heidelberg, New York, 1989. (to appear).

[Beierle and Böttcher, 1989] C. Beierle and S. Böttcher. PROTOS-L: Towards a knowledge base programming language. In *Proceedings 3. GI-Kongreß Wissensbasierte Systeme, Informatik Fachberichte*, Springer-Verlag, 1989.

[Böttcher, 1989] S. Böttcher. *Prädikative Selektion als Grundlage für Transaktionssynchronisation und Datenintegrität.* PhD thesis, FB Informatik, Univ. Frankfurt, 1989.

[Böttcher, 1990] S. Böttcher. How to use PROTOS-L as a logic-based database programming language. In H.-J. Appelrath, A.B. Cremers, and O. Herzog, editors, *The EUREKA Project PROTOS: From PROTOS to PROTOS II: Logic Programming Tools for Expert System Applications*, Springer-Verlag, 1990. (to appear).

[Böttcher and Beierle, 1989] S. Böttcher and C. Beierle. Data base support for the PROTOS-L system. *Microprocessing and Microcomputing*, 27(1–5):25–30, August 1989.

[Eckhardt *et al.*, 1985] H. Eckhardt, J. Edelmann, J. Koch, M. Mall, and J. W. Schmidt. *Draft Report on the Database Programming Language DBPL.* DBPL-Memo 091-85, Univ. Frankfurt, 1985.

[Meyer, 1989] G. Meyer. *Regelauswertung auf Datenbanken im Rahmen des PROTOS-L-Systems.* Diplomarbeit Nr. 630, Universität Stuttgart, December 1989.

[Smolka, 1988] G. Smolka. *TEL (Version 0.9), Report and User Manual.* SEKI-Report SR 87-17, FB Informatik, Univ. Kaiserslautern, 1988.

[Wirth, 1983] N. Wirth. *Programming in Modula-2.* Springer, Berlin, Heidelberg, New York, 1983.

[Zeller, 1990] M. Zeller. *Erweiterung eines Compilers für die Sprache PROTOS-L um ein Modulkonzept.* Diplomarbeit, Universität Stuttgart, January 1990.