A Tool Kit for Knowledge Based Production Planning Systems

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 describes the logic programming language PROTOS-L and how to use it as a tool kit for the programming of knowledge based production planning system prototypes. PROTOS-L embeds transparent access to relational databases, supports the programming of deductive databases, provides a module concept similar to Modula-2 and contains a type concept with subtypes and polymorphism. The programming language PROTOS-L and the PROTOS-L system have been developed and implemented at IBM Stuttgart. PROTOS-L is currently used to reimplement parts of production planning systems which have been developed at Sandoz AG in Basel and at Hoechst AG in Frankfurt. The investigated production planning applications need an integration of both concepts: read access to relational databases (which contain the relevant planning data) and a logic programming language for the implemention of heuristic rules (which control the planning strategy). In order to support both facilities, PROTOS-L fully embeds database access into a logic programming language.

1 Introduction

The Eureka project PROTOS investigates logic programming tools for expert system applications, and focusses on tools for production planning systems. Production planning system prototypes have been developed at Sandoz AG in Basel and at Hoechst AG in Frankfurt. Currently, it is investigated how parts of a planning algorithm for the Sandoz production planning problem, namely finding a production schedule for several products under given production constraints, can be reimplemented in the logic programming language PROTOS-L.

The paper is organized as follows. The second section describes some requirements of the PROTOS production planning applications, which are relevant to the use of databases and knowledge based systems. Next, we describe which facilities PROTOS-L provides in order to meet these requirements, i.e. we describe the module concept of PROTOS-L and how PROTOS-L supports database access and the development of deductive databases.

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

2 Requirements of the PROTOS production planning applications

The requirements of the PROTOS production planning applications result from many discussions with our industrial project partners.

2.1 A sketch of the production planning problems

Within the PROTOS project, there is already a production planning system prototype implemented at Sandoz in Basel [Sauer *et al.*, 1989], [Sauer, 1990], [Slahor *et al.*, 1990]. Another production planning system prototype has been implemented at Hoechst in Frankfurt. We summarize both applications and outline their requirements to databases and knowledge based systems.

A production planning system has to plan orders for several kinds of products and has to fulfill the production constraints. The most important production constraints for the production planning problem at Sandoz are the following:

- Each machine can only be used for one production step at a given time. If there are two production steps which are planned to use the same machine in overlapping time intervals, we call this a *planning conflict.*
- Each production step can be performed only on some of the machines, i.e. there are *different types* of machines.
- The production of a product must not be interrupted, i.e. intermediate products have to be processed immediately (*continuity constraint*).

Each order contains information about: which product is required, the order priority, the earliest time to begin the production and the time at which the order shall be carried out.

For each product there are several production alternatives. A production alternative consists of a sequence of production steps. Each step can be performed on several machines. Therefore, the order for a certain product can be carried out, if there exists a production time interval and a production alternative such that for all of steps of this production alternative there exists an available machine.

An important criterion for the quality of a plan is to carry out the production orders in time (or at least with a short delay). However, the search space is too large to find an optimal solution, e.g. to find the production plan with the minimal sum of delay days. Therefore, the planning algorithm combines heuristic rules and interaction with a human planner[1] in order to construct the

[1]The time needed for human interaction is acceptable, because

production plan. Human interaction is also required, because in the case of planning conflicts it is the decision of humans which product will be delayed.

The production planning system prototype developed at Hoechst AG in Frankfurt differs from that of Sandoz in several aspects. For example, the constraints and the criterion for the quality of a plan are different from the Sandoz production planning problem. In the case of Hoechst, the production planning system prototype is used for a couple of production lines, each of which is capable to fulfill any order for the given plant. It is desirable to produce many orders of the same kind sequentially on a single production line in order to reduce cleaning costs. The number of product changes on a production line shall be minimized. Moreover, if it is not possible to use a production line for a single kind of orders, then it is preferable to put similar production orders on a production line, which at least need less cleaning of machines than other production orders.

Both production planning scenarios have the following problems:

- The storage of data relevant to production in relational databases: for example, an `order` database contains the orders of customers and different `production` database contains information about used machines, product descriptions and information about which machines can can produce which products or can carry out which production steps.
- The search space is too large to find an optimal solution. Heuristic rules have to be used in order to find a good solution.
- Human planners must have the possibility to modify the planning strategy i.e. the decisions for conflict resolution.

In the next section, we summarize the requirements to a tool kit for the implementation of such production planning systems.

2.2 Requirements to a tool kit for knowledge based production planning systems

The need for database access

Embedding database access into a programming language for production planning systems is enforced by the following reasons:

- Most of the characteristics of products, production alternatives, production steps and machines are (or can be) stored in relational databases and have to be accessed efficiently.
- The production orders are stored in a relational database.
- Databases are used for information sharing, e.g. if the actual accessible set of machines changes because of repair or maintenance activities, then this information will be provided in a database relation.

The need for a logic programming language

A programming language for production planning systems should support the programming of rules for the following reason. As mentioned before, the search space for production planning problems is too large for an exhaustive search of an optimal solution. "Good" solutions found by heuristic rules and human interaction are sufficient for the applications. Finally, in our applications the heuristic knowledge for finding a good solution is typically given in the form of rules as in logic programs. Alltogether, a logic programming language seems to be the most adequate approach for the implementation of heuristic planning knowledge.

Further useful language features

A knowledge structuring concept, a type concept and the support of rapid prototyping are further useful properties of a tool kit for the development of production planning systems for the following reasons.

- In our applications, production planning systems will only be accepted, if the production planner can control the planning algorithm, i.e. he must be supported by a simple facility to exchange parts of the planning algorithm. Therefore, some kind of knowledge structuring concept or module concept would be very useful.
- A type concept would be useful for data structuring and for avoiding type errors.
- Since production plans and the underlying algorithms are often developed interactively, the support of rapid prototyping is also necessary.

3 PROTOS-L

This section outlines the design goals and summarizes the main language features of the programming language PROTOS-L, which are described e.g. in [Beierle, 1989] and [Böttcher, 1990b]. It further describes how the PROTOS-L module concept supports the development of knowledge based production planning systems.

3.1 Design goals

The design goals of PROTOS-L are to meet the production planning system requirements, i.e.

- to embed database access in a logic programming language,
- to support both backtracking and set-oriented query evaluation,
- to support data structuring and a powerful type concept and
- to support knowledge structuring, separate compilation and rapid prototyping.

In order to meet these goals, the logic programming language PROTOS-L incorporates the following basic language features, c.f. [Beierle, 1989] and [Böttcher, 1990b].

- The module concept supports knowledge structuring, separate compilation and rapid prototyping. Similar to Modula-2 [Wirth, 1983], PROTOS-L distinguishes between interface and body of a module and hides the implementation of the exported predicates in the module body. The body of a PROTOS-L module is either a *program body* or a *database body*. Program bodies implement the predicates of a module by a set of facts and rules in the embedding logic programming language. However, database bodies implement the predicates by a set of views on database relations. At the interface of a module it is transparent how the predicates are implemented, i.e. whether the module body is a program body or a database body.

the time needed for interactive planning is only a small fraction of the time during which the production plan is carried out.

- The logic programming language PROTOS-L integrates access to external relational databases. Furthermore, PROTOS-L supports the programming of deductive databases. A database body is used to access an external relational database or a deductive database.
- Set-oriented evaluation is used for rules in database bodies (i.e. for deductive databases) whereas backtracking is used for program bodies.
- Program bodies for small amounts of data used by prototypes can be substituted by database bodies for the access to external databases, when the first prototyping phase is completed.
- As in TEL [Smolka, 1988], the PROTOS-L type system [Beierle and Böttcher, 1989] supports data structuring and writing more readable programs, and it helps to avoid typing errors. The type system requires to assign types to all arguments of a predicate, but it provides subtypes and polymorphism, i.e. type variables. However, PROTOS-L types are not only used by the PROTOS-L compiler for type checking purposes, but also by the PROTOS Abstract Machine (PAM) [Beierle, 1989], [Böttcher and Beierle, 1989] for type inferencing in order to reduce the search space. If a predicate is implemented in a database body, then the types of its arguments are restricted to integer and string.

The basic design goals are met by the language features of PROTOS-L as follows.

- The module concept supports knowledge structuring, separate compilation and rapid prototyping.
- Database bodies support both access to relational databases and the programming of deductive databases.
- The type concept supports data structuring and more readable programs, it leads to a reduction of the search space during query evaluation, and it helps to avoid typing errors. A description of the PROTOS-L type concept can be found e.g. in [Beierle and Böttcher, 1989] and [Beierle, 1989].

3.2 The module concept of PROTOS-L

The PROTOS-L module concept integrates ideas from the module concepts of Modula-2, DBPL [Eckhardt *et al.*, 1985], [Böttcher, 1989] and TEL [Smolka, 1988]. As in TEL, the *interface* specifies only the predicate names and the types of their arguments. Different from DBPL, PROTOS-L supports database access by a special kind of module bodies and not by a special kind of interfaces. The advantage of this approach over the DBPL approach is that PROTOS-L allows to keep the database access transparent to the user of a module.

The example of figure 1 shows a part of an interface of that module in our production planning system which contains and computes the production data. The module computes

- which machines are used in which time intervals for which order and
- which production steps can be done by which machines.

It is hidden how the predicates are implemented, i.e. the concept of an interface supports knowledge encapsulation. Similarly, the distribution of rules into several separate modules supports knowledge structuring.

Separate compilation and thereby the maintenance of large program systems is supported 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 modified and recompiled independently of the rest of a module system.

The PROTOS-L module concept also supports two kinds of rapid prototyping: namely top-down and bottom-up system development. Top-down system implementation is facilitated as follows. The programmer can use dummy submodules which list only a few facts for rapid prototyping purposes in order to check whether an upper module works well. Later, the dummy submodules can be exchanged by modules which implement the desired predicates, without changing the interfaces of the module. 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.

Let us continue our example: During a rapid prototyping phase the predicates **used_machines** and **step_machines** could be implemeted in a program body, e.g. by listing some example facts. When the prototyping phase is completed successfully, this program body can be substituted by a database body selecting the facts from a relational or deductive database.

4 Access to relational and deductive databases

This section summarizes how access to relational and deductive databases is embedded in the programming language PROTOS-L, i.e. how database access is expressed within database bodies [Böttcher, 1990a]. The basic idea is to give the PROTOS-L programmer a uniform high-level database programming language. The impedance mismatch of other language integrations, e.g. the integration of SQL into C, should be avoided.

4.1 Transparent database access

The most important new aspect of the PROTOS-L module concept is that PROTOS-L offers two kinds of bodies of a module: *program bodies* and *database bodies*. Program bodies support logic programming with backtracking, whereas database bodies support set-oriented retrieval from external relational databases and user defined deductive databases.

Because access to an external database shall be transparent for the user of a module, at the interface of a module it is not visible, whether or not the corresponding implementation of the body accesses an external database and which kind of data retrieval is used.

For example, the predicates **used_machines** and **step_machines** could be implemented either as in Prolog by a set of facts and rules written down in a program body or by external databases as described in the third section. At the interface it is not visible how the predicates are implemented.

The database module can be used as an interface to a relational database as follows. The implementation of a module is a database body and the facts solving a predicate are taken from a relational database. Furthermore, for every database relation declared in the database body there has to exist a corresponding database relation in the database schema.

For example, the interface given in section 3.2 can be implemented by the database body given in figure 2.

```
interface production_data.
  rel  used_machines :  ?string x      ?int x      ?int x       ?int .
  %                     machine name   used from   used until   for order
  % Which machine is used in which time interval for which order ?
  % If the machine is available in the time interval, then the 4th argument has the value 0.

  rel  step_machines :  ?int x            ?string .
  %                     production step   machine name
  % Which production step can be done by which machine ?
...
endinterface.
```

Figure 1: Abstraction from details of the production data

```
database_body production_data using schedule_DB .
  rel     used_machines :  ?string x      ?int x      ?int x       ?int .
  %                        machine name   used from   used until   for order
  % Which machine is used in which time interval for which order ?
  % If the machine is available in the time interval, then the 4th argument has the value 0.

  dbrel   used_machines    is Machine_Rel( Machinename , From , Until , Order ).

  rel     step_machines :  ?int x            ?string .
  %                        production step   machine name
  % Which production step can be done by which machine ?

  dbrel   step_machines    is Machines_for_Productionstep( Step , Machinename ).
...
endmodule.
```

Figure 2: Implementation of relations containing the relevant production data

This database body requires that there are at least two relations in the database schedule_DB: Machine_Rel and Machines_for_Productionstep. Furthermore, Machine_Rel must have at least the four attributes: Machinename of type string, and From, Until and Order of the type integer. Similarly, Machines_for_Productionstep must have at least the attributes Step of the type integer and Machinename of type string.

However, the database module can also be used in order to program a deductive database. Therefore, 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. Note that the PROTOS-L programmer may program recursive and non-recursive rules in database bodies.

For example, the database body given above may additionally contain two non-recursive rules which are shown in figure 3. The first rule computes, which machines are available in which time interval, and the second rule computes which available machines can be used for a production step in a given time interval.

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 impedance mismatch of other integrations of database query languages into host programming languages, e.g. of the integration of SQL into C.

Different from 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.

4.2 Evaluation strategies for rules

PROTOS-L integrates two evaluation strategies for rules: set-oriented evaluation and backtracking. 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.

Set-oriented evaluation is preferable for the goals of a rule, if many calls of the goals are needed in order to solve the rule or if many solutions of the rule are needed in order to solve a higher goal. However, backtracking is preferable for the goals of a rule, if few solutions of its goals are sufficient in order to solve this rule and if few solutions of this rule are sufficient in order to solve a higher goal. A typical case where backtracking may be preferable is a program execution which computes only few solutions to a subgoal before running over a cut.

In PROTOS-L, the choice of an appropriate evaluation strategy is left to the programmer, assuming that he knows best how much search is needed before a rule can be solved. Whenever the programmer assumes that there are many results of goals needed in order to solve a rule R, 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 few results of goals needed to solve a rule R, he may prefer

```
database_body production_data using schedule_DB .
...
 rel  available_machines :  ?string x      ?int x       ?int .
 %                          machine name   used from   used until
 % Which machines are available in which time interval ?

      available_machines( Machine , From , Until )
                   <--  used_machines( Machine , From , Until , 0 ) .

 rel  step_can_use_machine :  ?int x  ?string x      ?int   ?int .
 %                            step    machine name   from   until
 % Which machines can be used for a production step and
 % are available in a given time interval ?

      step_can_use_machine( Step , Machine , From , Until )
                     <--  step_machines( Step , Machine ) &
                          available_machines( Machine , F , U ) &
                          F ≤ From &
                          U ≥ Until .
endmodule.
```

Figure 3: Computation of available machines from the production data

```
database_body production_data using schedule_DB .
...
 rel    single_step_product :  ?string x      ?int x            ?int .
 %                             product name   production step   time needed
 % Which products are produced by a single production step,
 % and how many days are needed for this production step ?

 dbrel  single_step_product     is
        Single_Step_Products( Product_name , Production_step , Time_needed ).
endmodule.
```

Figure 4: Products which require a single production step

```
database_body order_data using order_DB .
 rel    order :  ?int x    ?string x      ?int x .
 %               ordernr   product name   due date
 % Which order for which product shall be performed until which due date ?

 dbrel  order     is Order_Rel( Order_Id , Product_name , Due_date ).

endmodule.
```

Figure 5: Accessing the order database

```
module planning algorithms.
imports production_data , order_data .
...
 rel  servable_order :  ?int x    ?string x      ?int x .
 %                      ordernr   product name   due date
 % Which order for which product can be performed on some available machine ?

      servable_order( Ordernr , Product , Duedate )
      <-- order( Ordernr , Product , Duedate )
      & single_step_product( Product , Step , Duration )
      & step_can_use_machine( Step , Machine , Duedate--Duration , Duedate ) .
endmodule.
```

Figure 6: Computating servable orders from the order data and the deductive database containing production data

an evaluation by backtracking and use a cut at that place of a program, where no more answers to a goal are needed. In this case, he codes the same rule R in a program body, and the PROTOS-L system evaluates the rule by backtracking.

Let us continue our example. If it is assumed that many calls of `step_machines` are needed in order to solve the rule `step_can_use_machine`, then the rule `step_can_use_machine` is preferably implemented in a database body as shown in figure 3, because the database body performs a set-oriented evaluation of the rule. However, if it is assumed that only few solutions of `step_machines` are needed in order to solve the planning problem, and therefore backtracking is preferred, then the rule should be implemented in another program body instead of the database body. Hence, whether a rule which accesses database relations should be implemented in a database body or in a program body depends on the desired evaluation strategy for this rule.

4.3 Accessing 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 that import all the predicates they need from the database modules.

In our example, the production data and the order data are stored in different databases. Nevertheless, the knowledge of both databases has to be integrated for the production planning process.

Assume that the database body `production_data` implements a further relation `single_step_product`, which contains information about each product that can be produced by a single production step and about the time required for that production step (c.f. figure 4).

Assume furthermore, a relation `Order_Rel` is stored in a different database and therefore is implemented in a different database body (c.f. figure 5).

Then the PROTOS-L programmer can integrate the knowledge of both databases in the program body outlined in figure 6. The rule for `servable_order` computes which orders for single_step_products can be performed by some available machine.

5 Summary and conclusion

The PROTOS production planning applications need a tool kit which supports data and knowledge structuring, which integrates logic programming and database access and which supports rapid prototyping.

PROTOS-L provides the facilities which are needed for our production planning system applications. The module concept supports rapid prototyping, knowledge structuring and transparent database access by hiding the implementation of predicates from the user of a module. Additionally, database bodies facilitate the programming of deductive databases. Furthermore, PROTOS-L supports the integration of knowledge from multiple databases.

Efficient set-oriented evaluation strategies are used for rules contained in database bodies and backtracking is used for rules contained in program bodies. The PROTOS-L programmer selects the appropriate evaluation strategy for a rule, simply by coding it in a program body or in a database body. Because program body rules and database views are expressed in the same way, the PROTOS-L programmer has to learn only one single language. This avoids the impedance mismatch.

Finally, the example showed how to implement small fragments of a production planning and scheduling system within PROTOS-L.

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, 1990a] S. Böttcher. Development and programming of deductive databases with PROTOS-L. In L. Belady, editor, *Proc. 2nd International Conference on Software Engineering and Knowledge Engineering*, Skokie, Illinois, USA, 1990. (to appear).

[Böttcher, 1990b] 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*, 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.

[Sauer, 1990] J. Sauer. Design and implementation of a heuristic planning algorithm. In H.-J. Appelrath, A.B. Cremers, and O. Herzog, editors, *The EUREKA Project PROTOS*, Springer-Verlag, 1990. (to appear).

[Sauer *et al.*, 1989] J. Sauer, G. Micheaux, and L. Slahor. Wissensbasierte feinplanung in PROTOS. In *Proceedings 3. GI-Kongreß Wissensbasierte Systeme, Informatik Fachberichte*, Springer-Verlag, 1989.

[Slahor *et al.*, 1990] L. Slahor, F. Reuter, and H. Schildknecht. Scheduling problems: a user's perspective. In H.-J. Appelrath, A.B. Cremers, and O. Herzog, editors, *The EUREKA Project PROTOS*, Springer-Verlag, 1990. (to appear).

[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.