The Prince Project and its Applications

Pierre-Joseph Gailly[1], Wolfgang Krautter[2], Christophe Bisière[3], Sylvie Bescos[1]

[1] BIM, Everberg, Belgium
[2] FAW, Ulm, Germany
[3] CEFI, Les Milles, France

Abstract. The Esprit project Prince aims at development of an industrial Constraint Logic Programing environment based on the Prolog III language. In parallel, the current technology is being validated within the project on representative real-world examples. This paper describes the current status of the three application domains which were selected to demonstrate the applicability and usefulness of CLP. These applications deal with industrial systems engineering, medium term banking planning and jobshop scheduling as well as multiple plants global planning in the chemical industry.

1 General Goals of the Prince Project

The main goals of the Esprit II project Prince (P5246) are to develop a Constraint Logic Programing (CLP) system based on the Prolog III technology and to bring it to high industrial standards as well as to validate the technology by demonstrating its applicability and usefulness on real life applications. Prolog III [6] was developed by the group of Alain Colmerauer and the company PrologIA within the framework of the Esprit I project P1106. This previous project has demonstrated that CLP is an attractive solution for many advanced problems [13]. This paper will focus on application aspects but before that an outline will be given of the CLP system, called Prince Prolog, developed in this project.

Four essential aspects can be seen. First, in terms of expressiveness, it is intended not only to include the constraint domains already existing in Prolog III (i.e. infinite trees, linear algebra (over both floating point and infinite precision rational numbers), boolean algebra and lists; this last one being unique to Prolog III) but also to introduce new domains based on end-users' requirements (finite domains and interval arithmetic are being currently considered). Soundness and effectiveness of constraint solving algorithms receive careful attention. As was already the case with Prolog III, full integration of constraints into the kernel of the language and clear semantics of the interaction of constraints with the rest of the system are considered very important. Second is the efficiency issue: a completely new compiler-based system is being developed, drawing on the experience of two major Prolog manufacturers PrologIA and BIM and in particular the ProLog *by* BIM and Prolog II+ technologies. Besides a new kernel and improved constraint solvers is the third important aspect: investigation of global analysis and precompilation of CLP programs and incorporation of the results into the system. This longer term research is being performed by academic partners from the Universities of Bristol, Leuven and Madrid. Finally, the software engineering environment should provide easy, rapid and reliable development facilities as well as communication capabilities with the external world (Windowing, Databases, Network, other programming languages). In particular, constraints debugging is a new field which still requires much exploration.

Designing tools is one thing, putting them to practical use is another issue. This is even more true in software engineering. Industrial partners are validating the CLP technology by its "mise en œuvre" to solve real life problems. The manufacturers Bosch and MBB companies as well as the research institute FAW are involved in engineering applications for technical systems quality assurance. Two examples are under investigation: satellite attitude control and Failure Mode and Effect Analysis, the latter one being discussed in more detail below. The CEFI research center and La Hénin bank are developing a Decision Support System in the field of medium term banking planning. BIM is applying CLP technology to jobshop scheduling and multiple plants global planning tools in the chemical field. Besides showing the usability of the language, the application partners have provided important feedback to the implementors of Prince Prolog on the language itself, the selection of constraint domains, methodology and tools.

The different applications do not put forward the properties of CLP languages in the same way; however, the presentation will emphasize following characteristics:

- CLP languages are high level, making development and prototyping easier.
- Prolog enables the elaboration of reversible *symbolic* computer based model of real-life applications. CLP adds the domains dimension to the scene; in particular, Prolog III's *linear numerical and boolean* domains will be illustrated.
- Besides the built-in constraints, Prolog III's delayed goals can provide users with some facilities to break the linearity limitations.
- Combinatiorial search can be pruned more efficiently with the help of constraints.

2 The FMEA Application

2.1 Objectives

Increasing the quality of technical systems, in particular, identifying weak points, evaluating the effects of such weaknesses and the associated possible risks as well as determining their causes and investigating alternatives and improvements are very important. Failure Mode and Effect Analysis (FMEA) is a technique aiming at that.

In the Promotex project [13], FMEA of single components in stationary states has been studied. The project has shown that Prolog III is well suited to serve as a formal representation language to describe both the function of the components in the correct and faulty states and the structure of the system. In the Promotex system, component functions have been expressed in the form of systems of numerical constraints. In this project, the type of analysis has been extended at two levels: performing FMEA at the system rather than the component level and tackling dynamic aspects. Essentially two tasks are to be performed:

- investigate the consequences of known component failure modes,
- determine whether some faulty (possibly hazardous) system state can be reached as the consequence of failure modes of the components.

The first point can be addressed by classical simulation tools; the second requires more. The "multidirectionality" of Prolog III constraints enable reasoning on dynamic systems. Possible techniques are: numerical solution of differential equations, Petri Nets, methods of qualitative physics, simulation and process theory.

Figure 1 shows an example system: a closed loop speed controlled motor. The motor drives a sensor tooth wheel that produces 4 interrupts per revolution. The interrupt

routine measures the time elapsed between two interrupts and estimates the motor speed. Each time a pulse is generated, the proportional controller changes the voltage with respect to the nominal speed. This system contains continuous elements (motor) as well as discrete events (pulse, interrupt). Possible failures are, for example, sensor wheel slip or tooth breakage; these are represented by parameters in the model. Possible failure effects are speed oscillations or deviations from the nominal speed.

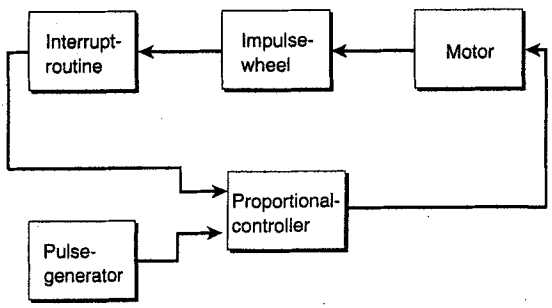

Fig. 1. Closed loop speed-controlled motor

2.2 Continuous Simulation

In continuous simulation, differential equations are solved numerically. If the solving methods are implemented using numerical constraints, the simulation may "run forward or backward", choosing to fix the value of some boundary conditions or system constants, leaving the others unknown and to be found by the simulation. Values cannot be fixed or left unknown arbitrarily as, at runtime, contraints must be linear. In addition to the common simulation run, the Prolog III mechanism allows the following queries:

- given the solution at certain time steps or intervals, infer single parameters of the differential equations. For this purpose, failure modes were modelled as parameters in the system.
- representation of parameters in input and output values as numerical intervals[4].

Example: the speed $n(t)$ of the motor can be represented with the differential equation

$$\frac{dn(t)}{dt} = \frac{-n(t) + k * v(t)}{Tc} \stackrel{\text{def}}{=} f(t, n(t))$$

where n is the speed and t is time; v, the voltage, k, an amplification factor, and Tc a time constant are model parameters. The trapezoidal method for solving a differential equation numerically is:

$$n(t_{i+1}) = n(t_i) + \frac{t_{i+1} - t_i}{2} * (f(t_i, n(t_i)) + f(t_{i+1}, n(t_{i+1})))$$

This can be implemented using Prolog III as follows. The **step** predicate has three arguments: the first represents the initial state in the form of a tuple **<Time, Speed>**; the second contains the system parameters (with $Dt = (t_{i+1} - t_i)$) and the third is a list of the successive states starting from the initial one[5].

[4] This was done using inequality constraints and **maximum** and **minimum** built-in predicates.

[5] The number of simulation steps being specified via constraints on the size of this list.

```
step(V, C, <>).
step(<Ti, Ni>, <Tc,K,V,Dt>, <<Tip1, Nip1>>.L) :-
  step(<Tip1, Nip1>, <Tc,K,V,Dt>, 1),
  {Ti   = Tip1 + Dt,
   Nip1 = Ni + Dt/2 * ((- Ni + K*V)) + (- Nip1 + K*V))/Tc)}.
```

Assume that the motor is initially stopped and that it should reach a speed between 5990 and 6000 during the last five steps of a 500 step simulation. The following query determines the range of possible values for the unknown amplification factor under the above constraints.

```
>step(<0,0>, <0.05, K, 6, 0.001>, L.<<T1,N1>,<T2,N2>,<T3,N3>,<T4,N4>,<T5,N5>>)
   minimum(K,M1) maximum(K,M2) out(<M1,M2>) line fail,
   {L::495,
    5990<N1<6000, 5990<N2<6000, 5990<N3<6000, 5990<N4<6000, 5990<N5<6000}.
```

Prolog III yields the answer: K should be in the range `<998.387, 1000.5>`. When dealing with more complex systems (e.g. the example system), the computation time gets critical because of the large number of choices induced by greater model complexity.

2.3 Petri Nets

Petri Nets are a method for performing discrete-event simulation, that has also been investigated for system safety analysis [10]. In particular, the ability to reverse the simulation process is of interest. To represent a system as a Petri Net, the standard Place-Token nets (that can be analyzed and reversed using linear algebra) are not sufficient. They have to be extended in two ways: addition of delays to the transitions to express the duration of the actions, individual firing conditions, data types and functional instructions in the transitions.

If the firing conditions and functional instructions are restricted to linear expressions, transition firing can be reversed. This is a limitation, but nevertheless an important enhancement over Place-Token Nets. Figure 2 shows the representation of the example system. The petri net is described with the help of the `transition` predicate

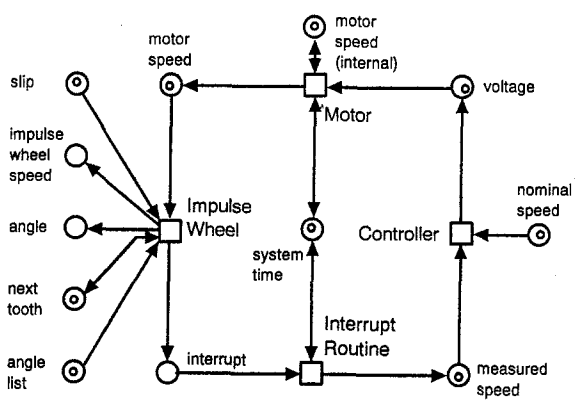

Fig. 2. The example system represented as Petri Net

in Prolog III. For example, the controller is defined by the following clause. It indicates that the measured and nominal speed are related to the voltage via the linear expression below, where R is retrieved from the model via the `const` predicate. Furthermore, this transition does not add any delay.

```
transition("Controller",
           input_places(<<"measured_speed", Ms>, <"nominal_speed", Ns>>),
           output_places(<<"voltage",V>>),
           delay(<0,0>)) :-
   const("R",R),
   { V = R * (2 * Ns  - Ms)}.
```

This transition description can be used both forward (computing output places from input ones) and backward. One of the drawbacks of Petri Nets is their emphasis on the event-oriented aspects of a system, while continuous processes are rather difficult to integrate. Furthermore, describing an industrial system in form of places and transitions requires a rather high level of abstraction.

2.4 Qualitative Methods

In addition to traditional simulation methods, using qualitative reasoning for system analysis purposes is being investigated. Qualitative reasoning describes a system and predicts its behaviour in qualitative terms, e.g. "raising", "oscillating" or "constant". This requires a higher level of abstraction than for a numerical simulation. Various qualitative methods have been proposed; for an overview see [14]. The possible use of Prolog III will be described on two examples:

- Kuiper's QSIM system [7] describes a system in terms of qualitative states, consisting of a qualitative value and a qualitative direction (one of decreasing, increasing and steady) at a time instance or during an interval. The behaviour of the system is then simulated by examining possible state transitions. Allowed transitions are held in a table. Between the functions the relations addition, multiplication, minus, proportionality and derivation may hold. QSIM then gives rules constraining qualitative values and direction of changes. Expressing these rules using Prolog III equality and disequality constraints makes both the programming easy and efficiently filters out forbidden states or transitions combinations.
- Allen's Calculus of Time [1] proposes a model based on time intervals. The relationship between two intervals can be expressed using 7 base relations and their inverses. The goal is to find the transitive relation between more than two intervals. In Prolog III, time intervals can be represented as tuples of two variables representing the start and end points. Constraints are used to express relations between intervals. The modelling of a relation will require to describe explicitly when the relation holds and when it does not hold. For example, the predicate `relation(Name, Truth_val, Int_1, Int_2)` will succeed whenever the relation called `Name` between intervals `Int_1` and `Int_2` has truth value `Truth_val`. The following clauses describe the `during` relationship which is satisfied if and only if the first interval is strictly enclosed in the second (where `1'` and `0'` represent the booleans true and false).

  ```
  relation(during, 1', <S1,E1>, <S2,E2>) :- { S1 > S2,  E1 < E2 }.
  relation(during, 0', <S1,E1>, <S2,E2>) :- { S1 <= S2 }.
  relation(during, 0', <S1,E1>, <S2,E2>) :- { E1 >= E2 }.
  ```

  With boolean constraints, all propositional logic operators can be used. For example, assume three intervals i1, i2, i3; also assume that one of the relations `equal`, `during`, `overlapped_by`, `start`, `started_by`, `finish` holds between i1 and i2. The following query determines which relations may hold between i1 and i3?

```
> positive(<I1,I2,I3>),              % intervals must have a positive length
  relation( equal,          B1, I1, I2),
  relation( during,         B2, I1, I2),
  relation( overlapped_by,  B3, I1, I2),
  relation( start,          B4, I1, I2),
  relation( started_by,     B5, I1, I2),
  relation( finish,         B6, I1, I2),
  relation( meet, 1', I2, I3),
  findall(R, relation(R, 1', I1, I3), L),
{ B1 | B2 | B3 | B4 | B5 | B6 = 1'}.
```

The answer is: `L = [finished_by, contains, overlaps, before, meet]`

Prolog III is well suited to implement the methods of qualitative physics. Yet, the loss of information as a consequence of abstraction is a problem inherent to qualitative methods. It is not yet clear if they can be used for FMEA purposes.

3 The Banking Application

3.1 General Overview

The central point in the development of a Decision Support System (DSS) in the field of medium term banking planning is reversibility, i.e. to bypass the existing frontier between *simulation* and *decision* systems in the financial domain. The DSS has been specified and a Prolog III prototype is being developed.

The application relies on a medium term banking model, mainly oriented towards interest rate risk management. Interest rate risk comes from a possible mismatch between asset and liability structure of the bank's balance sheet: if a loan is refunded by a borrowing of a different nature (i.e. short/long maturity, fixed/floating interest rate, etc.), the future net income of this production will be affected by fluctuations in interest rates. Because such a mismatch usually increases expected earnings and risk at the same time, funding decisions will be stated according to the bank's global risk preference. The model has been built in collaboration with the La Hénin bank. Basically, it can be viewed as a set of interconnected modules, each of them dealing in detail with a particular aspect of banking activity: outstanding loans evolution, funding policy, evaluation of potential risks, expected earnings, environmental constraints, etc. When used in simulation, this model is fed with a set of hypotheses and decisions. It then works out a medium term forecast for the main banking aggregates, under the form of a balance sheet, as well as an evaluation of the risk-return position that has been reached.

Of course, analysis of the consequences induced by alternative decisions (i.e. "*what if*" analysis) is an important step of the banking planning process. Nonetheless, this procedure remains limited, because the ultimate goal of decision makers is to choose an action which allows them to reach a given objective (i.e. "*what for*" analysis). Using conventional languages and approaches, two different methods can be used to perform this goal-oriented search. The first one is to repeatedly run the simulation algorithm with slight variations on the decision variables values (in particular, the funding policy), in order to get closer to the expected risk-return position. This method is time consuming and, more important, has little chance to "smoothly converge" to the desired point, because of the complicated form of a real-life banking model equations.

The second approach is to build a "reverse" version of the simulation model, which can generate the funding decision according to a given and specific risk-return position. The model has then to be rewritten for this particular purpose, and will have to be rewritten each time the head management wishes to ask another type of query. Besides development and maintenance costs, this solution also implies knowledge duplication.

In this context, CLP languages are of high interest. Their original features seem able to facilitate the development of flexible and powerful DSS in the financial domain [4, 5, 9]. Their declarative aspect and reversibility, extended to the numerical domain, leads to faster and more straightforward implementation of the model. The resulting "knowledge base" remains unique, but can be used for simulations as well as for goal seeking queries ([3] for a qualitative reasoning approach). The search space exploration algorithm, combined with pruning facilities, helps to efficiently solve complicated cases of goal seeking queries (involving piecewise linear, or non-monotonous functions). In addition, the goal delaying mechanism provided by Prolog III can be used to improve the treatment of non-linear equations; [8] provides a formal treatment of this problem.

How the linear part of the model is tackled will be explained first; then the treatment of nonlinearities will be explained. Because of the application domain involved, focus will be on numerical constraints.

3.2 Using CLP in the Linear Case

Prolog III cannot deal directly with non-linear constraints and uses a delayed constraint mechanism to handle multiplicative non-linearities[6]. Assume, for the moment, that the banking model is just a set of linear equations and inequalities. Then, this set can be directly implemented as a set of numerical constraints (S), which therefore represents the structure of the bank's behaviour. In this context, to "run" the model means to add to S another set (Q), which represents the user's query. This last set is mainly made of simple equations of the form *variable* = *value*. The resulting set $A = S \cup Q$ then represents the "answer" to the user's query. This very simple approach allows to integrate simulation and decision model, because of reversibility. The user is therefore free to ask any question that make sense to him.

Furthermore, this approach can be combined with the backtracking mechanism of CLP languages in the following way. When the banking application is started, the set S is created. From this unique "root", the software then develops small branches, each of them corresponding to a user's query. At each step, the system follows a branch in order to solve a query (i.e. to add a set Q and to display the result), and then backtracks to the root, ready for the next step. By doing so, it is possible to avoid an important part of the computational cost overhead induced by the non-specialisation of the decision support system. All this is summarised in Fig. 3.

3.3 Dealing with Hard Constraints

The actual model also contains non-linear equations. Two classes of non-linear equations can be distinguished: those corresponding to disjunctive constraints which can therefore induce a whole search process during goal seeking analysis (e.g. non-strictly monotonous functions, piecewise linear functions,...) and the others (e.g. involving logarithmic or exponential function or quadratic functions over positive numbers,...).

[6] I.e. when, at runtime, the added constraint involves the product of two unknown variables.

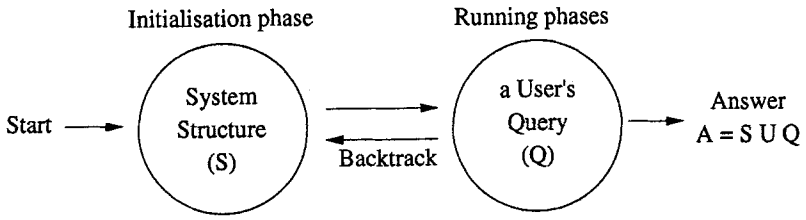

Fig. 3. Functioning of the application in the linear case

Consider a model made of linear equations and non-linear equations of the second class. During the initialisation phase of the application, only linear equations can be introduced in the set S application. The others have to be delayed, hoping that each set Q will bring enough information to introduce them in the whole set A. Consider the following non-linear equation:

$$P = \frac{C}{R} + \frac{NR - C}{R(1+R)^n}$$

Depending on the known variables, the following methods can be used:

- if R and n are known, the equation becomes linear,
- if all variables except n are known, n can be determined by calculus,
- if all variables except R are known, it can be iteratively approximated.

This simple strategy has been programmed in a generic way, using the **freeze** mechanism of Prolog III. A set of predicates has been designed, which facilitate the implementation of these non-linear equations. Using this toolbox, it is only necessary to "declare" the methods which can be used to incorporate an equation under the form of a linear constraint in A, and the conditions under which each method can be activated. Of course, this approach does not transform a CLP language into a general purpose mathematical solver, counter-examples (leading to "deadlocks") can be found easily. However, because of the particular semantics of the models which must be dealt with, these deadlocks can be precisely identified, and then solved according to the previous strategy.

Non-linear equations of the first class are handled in nearly the same way, taking care of the virtual combinatorial explosion they could induce in goal seeking mode. In the case of a piecewise linear function, this danger is clearly shown by the corresponding disjunctive writing. For a classical non-monotonous function, the problem arises from the fact that its inverse function is not univoque. Obviously, these equations cannot be activated during the initialisation phase of the application (see Fig. 3). The corresponding goals are therefore delayed, waiting for more information. Then, a strategy should be stated to organise for their best handling. In the application, a set of heuristics has been designed to achieve this task. This additional feature leads to the general structure drawn in Fig. 4.

The presentation only showed some particular aspects of the financial application. Other aspects include temporal shift detection or sensitivity analysis. In the future, one of the most ambitious features which will be addressed is the design an interface module, aimed at assisting users in their quest for a good decision. To do so, this module will have to "interpret" sets of constraints returned by the system.

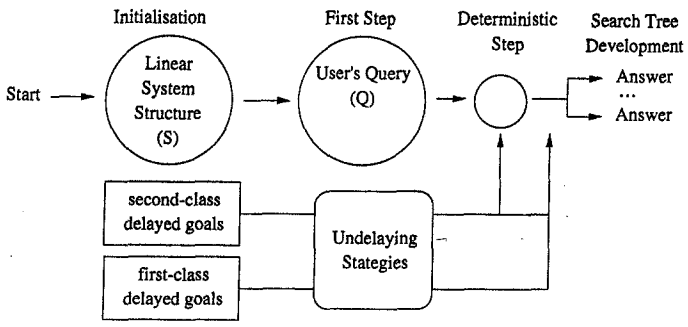

Fig. 4. Functioning of the application in the general case

4 Scheduling Applications

Job-shop scheduling of a chemical plant is a real-life example taken from the background of the EUREKA project PROTOS [2, 11]. A set of products has to be produced on different apparatus. A given apparatus can be used in the production of several products (multipurpose apparatus environment). Each of the products is produced in a single process where each process may have several (1 to 3) production variants (i.e. recipes). Each production variant is decomposed into a finite number (from 5 to 20) of steps. For each production step, one apparatus will be chosen from a set of alternatives. The production steps have to be performed in a specific continuous sequence (production cycle). When a larger quantity than the one delivered in one production cycle is required, the cycle is repeated. The problem consists in finding a time interval within the earliest possible starting date (all inputs have to be available) and the latest possible due date (management requirement) for the production process of each product.

The second example deals with the global planning of several plants. The whole production of a large Swiss pharmaceutical company is split over several plants. The aim is to compute a global production plan for all these plants. Up to now, no such global plan existed and all the coordination and production process adjustments between the different plants are achieved through phone calls between plant managers; there is no global control. This scheme works because of the plant managers' experience, but there is a high risk of the result not being optimal. If a good global plan could be provided, ensuring that no major coordination problem should occur, then each plant could make local optimisations as long as the constraints imposed by the global plan are respected; also the resulting production process would become much closer to optimality. As a side effect, this global plan would also reduce the need for the phone call based coordination, although it is not expected to suppress it totally.

As it is far too complex to take into account all details of the local data of each individual plant, the considered global planning tool is based on an approximation of the local reality: the abstraction of individual machines in machine groups (A machine group is a set of machines located physically close to each other, and each order can be completely executed using only machines within one machine group). Thus, the output of this tool is only a "rough" global plan, that will then be further refined at each plant, by the local scheduling tool.

Both examples were implemented; essentially, two types of constraints were used: precedence constraints, straightforwardly expressed by numerical inequalities, and non-overlapping tasks sharing the same resource, which can only be expressed by choices.

5 Conclusions

The previous examples have shown the interest of Constraint Logic Programming in several real life application fields. The experience of the end-users who developed these examples is that CLP and Prolog III are good software-engineering tools for dealing with such problems, leading to reasonably concise and elegant solutions. In particular, the ease to implement, and hence compare, variations of algorithms combining heuristic and symbolic computation with constraints, has been appreciated.

Both the technical systems and banking applications have shown CLP's adequacy to define models which encode both structure and functionnality. The possibility of avoiding as much operational aspects in the model's definition is very important and enables to use the same model in several ways: direct execution or simulation and backward execution or goal seeking analysis. This form of reversibility extends the knowledge representation paradigms and increases their declarativity for non-symbolic domains; linear numerical and boolean examples were given. Non linearities raise difficulties but some practical, even if not general, methods to deal with them were outlined.

Applications have also shown the current limitations of the Prolog III interpreter. Efficiency, a common concern, is being improved by the development of the Prince compiler. Having other or more specialised constraint domains (such as being able to use constraints as choices in scheduling) would increase the expressive power, and hence the ease of use of the language. Such topics are currently being intensively studied in the framework of the Prince project, but this falls outside the scope of this paper.

The authors would like to thank the members of the Prince teams and in particular Paul A. Massey, for fruitful discussions and comments on earlier drafts of this paper.

References

1. Allen, J.: Towards a General Theory of Action and Time. AI V.23, pp.123-154, 1984.
2. Appelrath, H.-J.: PROTOS: Prolog Tools for Building Expert Systems - a Project Overview. Procs. of the 1st. PROTOS Workshop, September 1989.
3. Berndsen, R., Berthier, F.: Goal Seeking in Qualitative Reasoning: an Implementation in CHIP. in Procs. of IMACS international symposium, March 1991.
4. Berthier, F.: Solving Financial Decision Problems in CHIP. Procs 2nd Conf. on Economics and Artificial Intelligence - CECOIA 2, pp. 233-238, 1990.
5. Broek, J., Daniels, H.: A Constraint Logic Programming Approach to ALM Modeling in Banks. Comp. Sci. in Economics and Management, Kluwer Acad. Press, 4(2), 1991.
6. Colmerauer, A.: An Introduction to PROLOG-III, CACM, Vol. 33, N. 7, July 1990.
7. Kuipers,B.: Qualitative Simulation. AI, V. 29, pp.289-338, 1986.
8. Jaffar, J., Michaylov, S., Yap, R.: A Methodology for Managing Hard Constraints in CLP Systems. Procs. of the ACM SIGPLAN PLDI Conf., pp. 306-316, June 1991.
9. Lassez, C., McAloon, K., Yap, R.: Constraint Logic Programming and Option Trading. IEEE Expert, 2(3), 1987.
10. Leveson, N.G., Stolzy, J.L.: Safety Analysis Using Petri Nets. IEEE Trans. on Software Engineering, Vol. SE-13, No.3 (1987), pp.386-397.
11. M. Nussbaum and L. Slahor. Production Planning and Scheduling: A Bottom-up Approach. Procs. of the 1st PROTOS Workshop, September 1989.
12. ESPRIT-Project 5246: PRINCE. Report 17: Specification of the FMEA application.
13. ESPRIT-Project 1106: Further Development of Prolog and its Validation by KBS in Technical Areas. Final Report Part 2: Validation. 1990
14. Weld,D.S.,de Kleer,J.: Readings in Qualitative Reasoning about Physical Systems. Morgan Kaufmann, 1990.