Intelligent Support of Computation

Otakar Babka, Robert P. Biuk-Aghai

Faculty of Science and Technology, University of Macau
P.O. Box 3001, Macau (via Hong Kong)
Fax: (+853) 831694; fstdb@umac.mo, fstrpb@umac.mo

ABSTRACT

The presented hybrid environment represents a knowledge-based support of engineering computation. The knowledge-based subsystem is partially realized in Prolog, supporting an engineering software system written in a conventional programming language (typically in C). In an effort (i) to bring the latter component together with the Prolog-oriented subsystem, and (ii) to increase the intelligence of the environment, selected Prolog features are modelled in the environment of the chosen conventional programming language. The intrinsic intelligence of the environment can be increased particularly by registering whether or not variables are installed. This facilitates autonomous control of the computational process, providing not only a user-friendly environment, but also an active and predictive support of the user. Several practical applications were developed based on this concept.

1. INTRODUCTION

Complex engineering computations, assisted by Prolog-like knowledge-based support, are studied in this paper. Most engineering computations are realized in a conventional programming language (typically in languages such as C). On the other hand, the underlying paradigm of the supporting knowledge-based component is often very different (logic programming in our case). So, a crucial problem is to facilitate co-operation of these different components. The presented environment assists integration of both components:

- In an effort to decrease differences of both paradigms, as well as to increase the intrinsic intelligence of the computational component, some important properties of Prolog were partially modelled in the environment of the conventional programming language.

An important feature of Prolog is the concept of instantiated variables. This principle was partially adopted in the computational component distinguishing the instantiated and non-instantiated variables. So, the built-in selective mechanism can recognize potentially feasible procedures of the computational component at a given moment. In the next step, this selective mechanism compares all these potential candidates and selects the optimal one according to a given goal and several constraints. Since this decision-making process can be supported by the Prolog-oriented knowledge-based component, the decisions can be performed mostly autonomously by the system. So, once implanted in the computational component, this typical feature of Prolog can significantly contribute to a fundamental intelligence of the environment.

In modelling Prolog’s features, the selective mechanism plays a crucial role. The principle of this mechanism is described in the next section. The intelligent ability of the presented environment, composing autonomously a proper sequence of computations, is shown in the third section. In the following section, an application for CAD support is described.

2. SELECTIVE MECHANISM

Motivated by the logic programming paradigm, the task of the selective mechanism is to find a feasible and preferably optimum routine for a given goal, under given criteria and constraints. This process consists of two steps: feasibility test and selection.

Feasibility test
Feasibility of the routines is tested and the feasible routines are inserted into the conflict set for the time being. During the feasibility test, each input parameter of the tested routines is checked to determine whether or not
an input value is known at that moment. A routine is feasible only if all input parameters have been instantiated, meaning that all input values have been defined at any given time of computation.

**Selection**
If the number of potential candidates in the conflict set is two or more, the optimum feasible routine of the conflict set is selected according to the given *selective conditions*. These conditions are composed from the criteria and constraints, given by a user or left as default conditions (such a typical chosen condition could be (i) maximum accuracy of results, (ii) minimum or limited machine time of calculation, or several others). The corresponding attributes of potential routines are compared with the given selective conditions during the selection, and the best routine is chosen. This process is augmented by case-based reasoning [4], utilizing experience learned from previous successful cases.

### 3. AUTONOMOUS CONTROL OF COMPUTING

The Prolog-like selective mechanism can autonomously control a computing process, constructing an appropriate sequence of the called routines and other information sources, according to a given goal. In order to check the feasibility of routines, it is necessary to detect whether variables have already been instantiated [1]. On the other hand, it also opens up the possibility to define *invertible* programs [6] integrating routines of different orientations.

**Orientation**
Let the routine $r$ be given as $r(q_1, q_2, \ldots, q_n)$. The routine $r$ can communicate with other objects by:

- *explicit* (i.e. formal) parameters $q_1, q_2, \ldots, q_n$;
- *implicit* parameters $q_{a_1}, q_{a_2}, \ldots, q_{a_m}; m \times n$, which are global variables used as input, output, or input/output variables;
- *function value*, in the case where $r$ is a function: for the purpose of the orientation issue, the name of the function, $r$, is incorporated among the (output) parameters.

The routine $r$ models the (oriented) relationship of these parameters. The use of a parameter can be specified by its *mode*. There are three modes of parameters: input, output, and input/output. An assignment of the mode to each parameter defines the *orientation O* of this routine. A Prolog predicate can usually correspond to several orientations. Such a predicate covering more than one orientation is called *invertible*. However, routines in conventional programming languages commonly are not invertible. One way to model some invertible relations is as follows:

(i) Collect several routines, modelling the same relationship among the same (or almost the same) parameters. They mostly (but not necessarily) correspond to different orientations.

(ii) Encapsulate this collection of similar routines in a family. The most significant differences among these routines (also called members of the family) are usually their orientations, but they can differ also in other properties (e.g. accuracy, used algorithm, and others). For a given purpose, the *selective mechanism* can select the appropriate member of the family, according to the needed orientation and other factors. Thus, despite the fact that conventional languages do not support invertibility, the invertible relationships are accomplished with the help of the families (collecting and encapsulating routines of various orientations) which are supported by the selective mechanism.

Two main features were mentioned above:

(a) The sequence of proper routines is composed autonomously. This increases substantially an intrinsic intelligence on a very general, domain-independent level.

(b) Similar routines, corresponding to different orientations, are encapsulated in the family, modeling an invertible relationship. So, this Prolog-like feature is implanted in the environment of conventional programming languages. In addition, this approach also enables the invertibility of arithmetical computations, which is not usually possible in Prolog, since this is restricted by the built-in-predicate “is”.

### 4. EXAMPLE

The features mentioned above are illustrated in the following application from mechanical engineering.

**Individual girder**
Suppose there exists an individual girder (Figure 1), with two defined relations:

(i) Let there be a *family* of routines, represented by the *template*
expressing a relationship among force $f$, reactions $a$, and $b$, and distances $l$, and $r$ of the individual girder. As mentioned previously, the family collects several routines expressing the same relationship and among the same parameters, however each of these routines can correspond to a different orientation. Each needed orientation is represented by one or more member routines. For instance, the call: 

```
force_relation(F, a, b, l, r).
```

will return both reactions $a$ and $b$ as results (lower-case identifiers indicate uninstatianted variables, capitals stand for instantiated objects). A different orientation could be represented by the call: 

```
force_relation( f, A, B, L, R).
```

It will return force $f$ and right-hand distance $r$, for given reactions $A$, and $B$, and for left distance $L$.

Since the selective mechanism can choose the appropriate routine among all members of the family (corresponding to various orientations), the above family is invertible.

(ii) Besides the relationship of forces and distances, also deformation of the strained girder is defined. Let the

```
defformation(I, E, G, d, x, F, A, B, S_a, S_b, l, r).
```

be the routine expressing the girder deformation $d$ in the point defined by distance $x$, $E$ is Young's modulus of elasticity, $G$ is the shear modulus, $I$ is the section moment of inertia, $S_a$ and $S_b$ are shifts of the supports (it can be caused by a deformation of an underlying girder), other parameters have the same meaning as above.

**Set of girders**
Both `force_relation` and `deformation` were defined corresponding to the individual beam only. With the help of the selective mechanism, these defined relationships also represent a sufficient apparatus for a set of girders $G_1$, $G_2$, $G_3$, and $G_4$ in Figure 2 (left hand side). For a given force $f$, the deformation (depression) of the whole set should be calculated.

The software system uses the routine `deformation` and several orientations of the routine `force_relation` (encapsulated in the family), defining deformation and forces of one individual girder only. The system did not receive any knowledge about sequencing of these routines for sets of girders. However, despite this fact, the presented mechanism can solve the task autonomously and with only minimal domain-specific information.

**Calculation**
The following sequence was designed by the selective mechanism. The sequence represents the computing process depicted in the right-hand part of Figure 2.

```
[fr(G_1), fr(G_2), fr(G_3), fr(G_4), def(G_1), def(G_2), def(G_3)],
```

where, e.g., symbol $fr(G_i)$ means the call:

```
force_relation(f, a_i, b_i, L_i, P_i)
```

and represents the calculation of the reactions $a_i$ and $b_i$ for the girder $G_i$. Similarly, symbol $def(G_j)$ means call of the routine `deformation` for calculation of a local depression on the girder $G_j$. This achieved result is consistent with our intuition: it is obvious that (1) `force_relation` firstly should be applied to the upper girder $G_4$, then to $G_3$, and only then to the remaining ones, to define all forces and reactions. (2) Only after that, routine `deformation` can be applied, but in the reverse sequence, bottom-up, because of superposition of the depressions.
Recalculation

Invertibility can enable a simple recalculation. This recalculation is inevitable for an optimization of the design: Assume that all reactions were calculated, but the real maximum load weight of some actually used girder supports will probably be higher than previously calculated (in order to use only standardized supports, the girders of the same or next higher value will actually be used). Therefore, it would be suitable to recalculate this task and the admissible value of force \( f \), as well as new values of deformations under these modified conditions. The modified task would need a considerably modified sequence of the calculation. With the help of the selective mechanism, the new sequence for recalculation is designed:

\[
[fr(G_3), def(G_3), fr(G_4), def(G_4), fr(G_5), def(G_5), fr(G_6), def(G_6)].
\]

The process of recalculation is also outlined in Figure 2. Notice that a different orientation of force_relation was used for the recalculation. The correct routine was (automatically) chosen inside the force_relation family. Even a task as simple as the presented one can demonstrate that the selective mechanism can support a user considerably. The task would at least require the user’s basic knowledge of both mechanics and the software system (furthermore the user’s time, energy, concentration, and others, if the user should “manually” call the correct routines, in the correct order, and with the correct parameters). On the other hand, the presented selective mechanism can autonomously control the computing process and without any specific knowledge about sequencing. The mechanism needs only the input/output mode of parameters of all routines and other sources of information.

5. KNOWLEDGE-BASED SUPPORT OF CAD

The presented environment was implemented in several industrial applications. Some typical issues of the knowledge-based support can be illustrated with the help of a CAD application. Here the CAD software system is incorporated into the Prolog-oriented knowledge-based framework.

One of the very frequent issues of engineering calculations is the gradual refinement of the resulting parameters during the iterative process. An efficient knowledge-based support can substantially improve efficiency and convenience of this type of processes. The presented framework can also contribute significantly to the optimum solution. The knowledge-based system can autonomously guide this search process or at least considerably improve its convergence.

The presented framework was implemented for the design of a gearbox. The system guides a user to design
mechanical elements of optimum or sub-optimum parameters. The presented system consists of the following parts:

1. The preliminary design of shaft parameters.

2. A set of verification modules, where it is checked if the suggested preliminary parameters of the shaft fulfil the given criteria. This means that the shaft’s deformation module calculates maximum bend and inclination for the tentatively design shaft, the critical speed module calculates its critical speed, etc.

In this way, this application serves as a typical example of a wider class of similar tasks. There are basically two types of modules:

1. The required results are calculated immediately from inputs by modules of direct calculation. The preliminary design can serve as an example of the direct calculation. Inputting the needed main parameters, the preliminarily designed shapes of the mechanical elements are returned, along with other important parameters of these elements.

2. For some types of calculations, such a direct approach cannot be realized. By using an indirect (or backward) calculation, it is checked if previously estimated parameters fulfil the given criterion. Modules of the indirect calculations often tend to predominate in typical engineering applications, mainly for two reasons: (i) The number of criteria is lower than the number of parameters of a designed object. (ii) An appropriate algorithm for the direct calculation does not exist, or cannot be used effectively. Modules of critical speed and shaft’s deformation, as well as other verifications and tests of the presented application, are all examples of these indirect modules.

Using the indirect modules, the required parameters are estimated before the first iteration. Then, during every iteration, the parameters are checked by the indirect calculation and modified before the next iteration until they fulfil the given criterion. Two crucial factors of this iterative search process are:

(i) the initial estimation of the parameters;

(ii) the sensible modification of the parameters before the next iteration.

Without an appropriate decision-making process, the optimum cannot be found. The knowledge-based framework can, therefore, support this process very strongly, involving several search strategies and both general and domain-specific knowledge concerning the given problem.

In conformity with the above facts, the main parts of the gearbox were first roughly designed by the preliminary module. Then these parameters were checked by the verification modules. One of these modules, the critical speed module, calculates the critical speed of the tentatively designed shaft, since the critical speed depends on the shape and material of the shaft. However, since we can accept the shaft’s critical speed only if it is within a required interval of permitted values, the shaft has to be designed by an iterative process. The shaft’s shape is modified and the critical speed is calculated and checked consequently for this modified shaft in each step of the iterative process. However, it is a very difficult task to decide which part of the shaft should be extended or reduced in order to change the critical speed to the required extent. Moreover, any modification can have several other consequences that might potentially violate the given constraints. If a traditional software system is used, the computing process can be controlled optimally only by expert users.

In an effort to open the system also to less experienced users, and simultaneously improve both the process of the design and its resultant product, the knowledge-based subsystem is attached to the traditional software. Hence, even novice users can now successfully use this extended software system. The subsystem is capable (i) to understand the design process and also (ii) to suggest a promising next step. The user can be guided in using the software properly and effectively. With the help of the knowledge-based framework, the trial-and-error approach is substituted by a highly effective process.

The framework can also improve other features of the CAD software. Several earlier states of the design can now be stored, with the possibility to recover them later. Further, experienced users can also choose not to accept the system’s advice, but to make their own decisions. However, there would still be the possibility to query the system for any required information whenever needed. As an extension of the presented iterative process, the framework can also support the design of multiple alternatives. When designing an optimum product, these alternatives can be compared according to a given selective criterion, and the best alternative can be chosen.

REFERENCES