A STRATEGY FOR INFORMATION HANDLING IN PROLOG

Fernando Pereira

[** DRAFT **]

29 December 1984

1. Prolog for Information Processing

Logic programming has the potential of supporting the storage, combination and presentation of a wider variety of information in a wider variety of ways than has been possible with any other computer technology so far developed. The reason for this revolutionary new potential is simple: logic programming deals directly with statements of the information on outside world relevant to an application rather than with encodings of that information into bits in a computer's memory and programs that shuffle those bits.

The first step to realize the potential of logic programming is the development of a logic programming system competitive in terms of computation speed with more conventional ways of handling information. Quintus has so far been working on this first step. However, this is just what I said, a first step. What should follow?

The answer to this question is implicit in the opening sentence of this note:``...supporting the storage, combination and presentation... of information ...''. Our Prolog system gives us mainly the means to combine information effectively. We have now to deal with storage and presentation of information.

Before we look at those areas, though, we should examine a bit more closely what we have already achieved, and would be foolish to lose.

By its very nature, any usable Prolog system must include means of storing substantial amounts of information in the form of facts and rules. Note that this is rather different from other systems that fall under the ``programming system'' label. With conventional programming systems, storage of data is seen as a separate matter under the responsibility of the programmer that organizes data structures in memory or in external disk files. The Prolog system, on the other hand, from the information

storage point of view looks rather like a relational database: the user specifies what information he wants stored, but very little (or even nothing at all) of how it is to be stored. The ``how'' is the business of the Prolog system and most users will be glad not to have to bother with such irrelevant (in terms information content) details as to how many bits are used by this table or what data structure is used to access the elements of a table.

Thus, the current Prolog system can already be used not only as an information processing tool but also as an informations storage tool. In the second role, it suffers from limitations that I will discuss in detail below. Nevertheless, it is a very reasonable tool to store databases of the order of a few megabytes, and experience shows that for such databases, in particular when the information stored is anywhere complex (many relations and attributes), it is far more effective than existing relational database systems.

In using the Prolog systemas a tool to store and combine information, we are benefitting from an uniformity which is absent in more conventional tools. Information is stored in Prolog as facts and rules and is processed by the application of other rules and facts. In fact, there is no distinction between what ``is processed'' and what ``processes'': what we have are facts and rules that describe the relevant aspects of our application. Thus, a table of employees, which would conventionally be thought of as ``data'' and a table of compound interest, which would be conventionally part of the ``program'', are conceptually the same kind of object -- predicates -- and exist in the same way within the Prolog system. In contrast, in using a database system such as INGRES, we need to deal with three totally different languages and data representations: the one used to keep the actual information in the database, the query language used in retrieving data, and an external programming language such as C used to write information-combining applications (not to mention a report genration language for data presentation, which I will discuss later in the section on information presentation).

It is hard to exaggerate the importance of the above point: most of the tools of information storage, usually implemented in database systems, are already present in the Prolog system, if only in a somewhat limited form. It is on this auspicious start that we have to build.

2. Information Storage

As I argued above, the Prolog system already provides many of the facilities normally thought as part of a database system. What is it missing, then?

1. **Ability to store huge amounts of information:** The storage capacity of the current Prolog system is limited by the virtual address space of the host machine (16MB under Berkeley Unix on the SUN or VAX). Larger storage capacity requires the use of the file system provided by the host operating system.

2. **Better indexing:** Reasonably efficient access to large amounts of information requires special tables to guide the system more directly to the relevant information. These tables are called indexes and are best maintained automatically by the information storage subsystem.

3. **Better data persistency:** For relatively small databases (a few megabytes) it is reasonable to keep the information between use sessions as a disk file that records the internal state of the Prolog system, what is called a saved state. This is not practical for much larger databases, which must be seen as persistent entities independent of a particular activation of the Prolog system.

4. **Multiple access:** Very often large databases must be accessed simultaneously by several users. This requires special techniques to guarantee that the information seen by each user is always in a consistent state regardless of changes effected by other users.

5. **Reliability:** Databases must be protected from software or hardware failure. Simple copies of older versions of the database is practical for databases of a few megabytes, but more sophisticated methods are needed when size makes frequent copies impractical.

I will call **Prolog storage module** to the component of the Prolog system satisfying the above requirements. A Prolog storage module would allow the Prolog user and the user of applications embedded in Prolog to keep large stores of information and access and process it with the full power of Prolog and of Prolog-based applications such as natural-language front ends. The Prolog storage module would appear simply as an extension of capabilities already present in Prolog and not as detached system

alien to the spirit and techniques of logic programming.

One might argue that all the above requirements have already been solved in one or other of the many existing database systems, so it would be easier to take advantage of an external database system by interfacing it to Prolog. This approach would have the further advantage of making data already kept by the database system accessible from Prolog.

Unfortunately, things are not so simple. No existing commercial database system was designed to be interfaced to Prolog. Commercial database systems have all sorts of limitations, which are reasonable when the database system is in control of the application but which would make a Prolog interface complicated, slow and detrimental to our understanding of Prolog as a medium in which all data is handled uniformly. This is not the place to discuss the limitations in those respects of specific database systems, but the conclusion is that interfaces to external database systems are for the benefit of specific applications with a large investment in those database systems, not for the benefit of logic programming as an overall solution to information processing problems.

Therefore, although in the short term it might seem attractive to build an interface between Prolog and an existing database system, such an effort could only be justified by specific applications, and not as a means of building up the capabilities of logic programming systems. Ultimately, an interfacing effort would be at the mercy of the whims of database system suppliers and their clients, whereas a storage module properly designed for Prolog would be under full technological and business control of Quintus and would be a major step in moving logic programming from the AI ``ghetto'' into a much wider area of information processing.

As with other developments of logic programming, the development of a Prolog storage module should be organized so that intermediate stages can be made available to users. Features (1), (2) and (3) are closely tied to the use of the host operating system's file system and, with some minimal reliability mechanism, would be present in a first module usable for personal (that is, not shared) information storage. Features (4) and (5) would come later in a full-fledged Prolog storage module.

In parallel, a limited interface between Prolog and an external database system might be developed as a demonstration of concept to attract users with very large specific requirements. It is

practically impossible to antecipate what features of the external database system should be made accessible from Prolog for specific applications and therefore the status of the database interface should be the same as that of other demonstration programs, such as those being written to exemplify to users the possibilities of the C interface.

3. Information Presentation

Very much the same arguments can be made in the area of information presentation as I made above with respect to information storage.

It is of course possible to interface Prolog to various graphics and screen packages available on host computer systems. Our C interface is the means to put together quickly such interfaces, and users should be encouraged and educated to use the C interface to reach out into relevant facilities of host systems.

Nevertheless, logic programming offers a unique opportunity to rethink information presentation problems and create solutions that are much more general and flexible.

The main idea here is that an information display is no more than a pictorial representation of information already available, either directly or through computation, in the information store of an application. The display is build from four kinds of information

- The information to be presented, for example the annual earnings of a corporation over tha last 10 years.
- Rules that describe the pictorial appearance of various classes of information. For example, there might be a group of rules that build histograms for measure/time period tables (such as earnings/year).
- A description of the overall layout of the display, for example, where various windows are in relation to each other.
- A description of the features of the display device, for example size or color capabilities.

Again, it might be argued that some of these issues have already been addressed by the designers of graphics systems, report generators and the like. As before, this argument leaves aside what is most promising about logic programming: the ability to store and manipulate all sorts of information in a uniform, user-accessible way. For example, from a logic programming perspective, the differences between graphical displays, forms and reports are circumscribed to a relatively small area of appearence rules and device details, and it should be possible to change the presentation entirely without having to touch the information to be presented.

We will still be able to use many presentation concepts, such as device independence and windows, that were developed in more conventional settings, and it also clear that in a first system one will have to rely substantially on whatever facilities for information display are provided by host systems.

However, it is very important that the development of information presentation facilities for Prolog benefit be in support of, rather than instead of, the development of logic programming technology as an unified answer to application problems. This means that information presentation should not be seen as an easily dettachable Prolog application, but rather an organic growth of the capabilities of logic programming.

4. A Strategy for Growth

In presenting these views on information storage and presentation, I have started from a guiding conception: that logic programming is not a single specific tool for a particular class of problems, but rather a developing technology capable of providing successively more encompassing solutions to successively wider classes of problems. In this, logic programming technology differs radically from specific classes of tools such as database systems, expert system shells or graphical interfaces.

Logic programming technology should be seen ask the same kind of thing as computer technology in general, deriving its spetacular growth from the ability to be molded to serve both existing and newly created needs at constantly lowering costs. Put in another way, logic programming, logic programming, as computing in general, gives us a new, much simpler and more flexible way of storing and processing information, not only tables as in personnel records, or matrixes of numbers as in structural analysis, or geometric models as in CAD/CAM, or rules

of thumb as in expert systems, but any kind of precise information. This unique ability stems from a close match between the conceptual basis of logic programming and the way in which we ourselves organize our knowledge about the world.

The growing applicability of logic programming will mean that more and more complex problems will be attacked using its tools. It is then clear that the only way to sustain this growth is to provide logic programming systems that are not only more comprehensive, but also faster. In fact, much of the current Quintus Prolog and certainly even more of its future extensions will be written in Prolog itself, **for the very same reasons** that support the use of Prolog in applications. As with computing in general, increased functionality requires even more increased raw computing power. It is an open question whether general-purpose computers, not tailored for Prolog, will be able to satisfy those increasing demands. Furthermore, the natural progression of logic programming technology will impel us to use the technology to organize th whole computing environment in logic programming terms, bypassing entirely the much more primitive tools provided today by operating systems. Going in this direction will clearly mean the introduction of computer systems specifically tailored for logic programming.