Effect of format on information and problem solving*

Mukesh J Patel, Benedict du Boulay and **Chris Taylor**
School of Cognitive and Computing Sciences,
The University of Sussex,
Falmer, Brighton BN1 9QH, UK.
Email: mukesh@cogs.sussex.ac.uk

Abstract

This study reports the effect of differences in format of Prolog tracers on Prolog problem solving tasks. Three different tracers (Spy, TPM, and EPTB) in different formats were tested to check for their relative effectiveness in solving five different Prolog problems. 43 subjects attempted to solve each problem with each trace (15 problems in total). Preliminary analysis of solution times and response data indicate that EPTB performed best across all problems. An account for this finding is presented, as is one for a number of interesting interactions between the effects of problem type and trace format, which supports the general conclusion that while format is a significant determiner of access to information, it can also constrain the sorts of problems that could be solved readily with that information.

Introduction

The basic aim of this study was to examine differences in user performance due to differences in Prolog tracer format. The results of the effect of format, particularly on the text-graphics dimension, on the utility of information in solving specific problems encountered by novice programmers are reported. A tracer is a basic tool employed in a wide variety of programming tasks as well as being used by beginners to help them learn how to program. In particular, a tracer can explicate the relation between a program as a piece of text and the actions that it describes. So a trace of specific program is a sequence of changes in the data structures and control flow decisions that have caused changes during program execution. In Prolog the tutorial function of the tracer is more marked than in other programming languages because of the complex nature of the underlying mechanisms of this language. Important aspects of any tracer are (i) what information about internal mechanisms and transient states it is designed to show, (ii) the notation or format in which the information is shown, (iii) the method of controlling the tracer i.e. its interface, and (iv) the intended users and expected tasks for which it will be employed (Taylor, du Boulay & Patel, 1991).

In order to be fully conversant with Prolog it is necessary to understand its complex, internal and largely hidden mechanisms. These operations can be described and/or explained in more than one way (Pain & Bundy, 1987) with different perspective emphasising different aspects, such as variable binding, flow of control, recursion, search space, etc., though most tracers usually provide the minimum information necessary for reconstructing the whole 'story' of a program's execution. Tracers are typically evaluated in terms of variance in *explicit* information about these aspects, and not whether such information is present or absent. The explicitness of information is partly affected by format, and that interaction was taken into account in designing this study.

Differences between Prolog Tracers

This study concentrates on investigating two dimensions on which Prolog tracers vary; the overall amount of explicit information, and, the format of presentation of the information. Three tracers were evaluated. The first tracer, Spy (or Byrd Box) is rather basic with limited explicit information presented in a textual format which is not always easy to comprehend (Byrd 1980). The second one, Enhanced Prolog Tracer for Beginners (EPTB), also textual, was developed by Dichev and du Boulay (1989) to overcome some of the obvious shortcomings of Spy. The third tracer, Transparent Prolog Machine (TPM), is an *idealised* version of a commercial product based on work by Eisenstadt and Brayshaw (1988) which displays certain information such as flow of control and backtracking in graphical representations of AND/OR trees.

How do tracers vary in terms of explicit information? In this context, the term *information* refers to information about when, how and which clauses are matched,

*This work was supported by a grant from the UK Joint Research Council Initiative in Cognitive Science/HCI. The experimental work was conducted using the POPLOG programming environment.

how variables are bound to (and unbound from) particular values at certain points in the running of the program, and the overall flow of control, including backtracking, together with the success or failure of goals. Tracers provide such information more or less explicitly in different ways. For example, Spy does not explicitly indicate which clause of a predicate is being used at any point, whereas EPTB and TPM do. And, Spy refers to program variables by their internal names such as '_405' while, TPM systematically labels variables with letters, and unlike both, EPTB uses the names chosen by the programmer appended with a numerical subscript to distinguish between copies. Though all three methods serve the same function, they are not equally useful.

So tracers can provide useful information in different ways depending on emphasis on particular perspective of Prolog, which affects their relative informativeness. Because information is partly implicit (e.g., down the screen ordering) and partly explicit (numerical goal numbers) it is hard to determine the unavailability of information. In order to avoid confusion due to this it was assumed that, as far as the experimental task was concerned, all three tracers provide the same sort of information, and that any main differences in their usefulness is due to format. More realistically, it is obvious that in most cases there would be some interaction between format and information content, and so any explanation of helpfulness of tracers would have to give an account of such an interaction. But our strong assumption of information equivalence is justified because the stimuli problems were designed and tested to ensure a fair evaluation of similar features of each tracer. Further, the same rigour in designing the task material enabled us to determine more clearly the source of such interaction.

So assuming that the information content of tracers is fixed for all relevant aspects of Prolog, how can they vary in terms of format? Tracers can present information in a *mainly* graphic or text format. Aside from trivial cases, the distinction between graphic and textual format is rarely clear cut; even in a fully textual format the serial ordering of information about states at each step in a program run reflects an implicit notion of temporal ordering; information at any step *n* is partly dependent on information at step *n-1* and will itself determine the information content of step *n+1*. Serial ordering information is such a case is not textual but *graphical*. Conversely, a graphical tracer would be of limited value without textual information such as the value of bound variables at particular steps in the run. However, for our purposes a crude distinction between the two categories of tracers is sufficient because our aim is to concentrate on differences in usefulness due to very broad parameters; for example, the difference due to the representation of flow control as graphical AND/OR trees or as linear ordering of textual information.

The following list of general advantages of graphic format also serves to highlight the disadvantages of text format. It is included to anchor some of the basic differences between graphic and text formats. Some of the advantages such as items 7 and 8 are dependent on the extra degrees of freedom provided by computers and bit-mapped screens.

1. Impact.
2. Proportional spatial display can represent degree of relatedness.
3. Clarity in display of non-linearly ordered information such as loops and backtracking.
4. A degree of abstraction can help overcome ambiguity in information.
5. Can display more than one perspective, if information points are related between dimensions. Also possible in textual format.
6. Enables zooming in and out to access essential relationship between information points.
7. Greater possibility of displaying dynamic process (animation) with real time updates. Though possible in text formats they would lack clarity.
8. More *direct* access to information though this depends on the versatility of the display method.

Note that some advantages are not exclusive to graphic formats, and others are of limited value unless augmented with textual information. Aesthetic aspects of graphic representations can improve communication efficiency (Shu 1988), but such improvement often depends on the nature of information being displayed. In the case of Prolog programs there is a limited number of ways in which information about flow of control and backtracking can be graphically displayed. For example, flow of control can be represented as AND/OR trees, OR-trees (Hook, Taylor and du Boulay, 1990) or in terms of flow of satisfaction arrows (Clocksin & Mellish 1981). In such cases the spatial arrangement of certain aspects of information plays a crucial role in capturing essential and important relationships between different parts of a program. This would be expected to be more helpful in solving problems related to flow of control and backtracking. However, major advantages of graphic formats combined with graphic terminals, may be adversely affected by modality mismatch between textual source code and graphic trace, which is a possible source of ambiguity that could affect ease of access to information. Our results indicate that modality mismatch has some adverse effect on subjects' problem solving performance. Thus advantages of graphic format should not be regarded as uniformly beneficial.

Details of tracers

While it seems that a graphic format has certain advantages over more conventional textual format, other

factors can affect their usefulness for particular tasks. Tracers used in this evaluation study are briefly described to illustrate major differences between them. Most of the emphasis is on highlighting the effect of format on information about certain aspects of Prolog, because, though the degree of differences between tracers is partly influenced by how explicit an aspect of Prolog is presented in a trace output, solutions to problems used in this study depend to a very large extent on the effect of format on ease of access to information. Note that the tracers were not used truly as tracers in a dynamic way. In each case an appropriate screen dump of the relevant trace output was shown in its entirety, so users could not 'grow' the trace nor add or delete information from it. Thus, variation between method of control between tracers was eliminated. Our intention was not to examine the tracers 'in the round' but to focus on the influence of format on clarity and accessibility of information.

Spy

Spy is a very basic textual (linear) tool and is included in this study because subjects were familiar with it. The version used in this study did not show system goals. This tracer provides most of the basic information necessary for programming or debugging in Prolog, but much of it is implicit. In particular, the relationship between the source code and the trace output is not as clearly displayed as it is in TPM and EPTB, which are both designed to overcome some of the obvious shortcomings of Spy. Given the basic lack of clarity in information presentation, Spy was not expected to perform better than TPM and EPTB.

Transparent Prolog Machine

TPM tracer makes use of a modified and extended AND/OR tree representation, and is an interesting illustration of how graphical techniques can be utilised. The trace outputs used in this study were modified to include all the relevant details which are normally optionally selected by the user. The spatial layout of TPM provides a great deal of information at a glance, particularly on the flow of control and search space. The trace also *seems* a lot less cluttered than Spy or EPTB; it gains in clarity by exploiting some of the advantages of graphic format outlined above. However, the use of a graphical representation of AND/OR trees restricts the screen space available for information about predicates with a large number of arguments and/or long lists, or variables with long names. This constraint is a consequence of the tracer format. This problem can be overcome by including a scrolling facility but the display of essentially textual information is still poor compared to more conventional textual tracers such as EPTB and Spy.

Enhanced Prolog Tracer for Beginners

EPTB is a textual (linear) tracer (Dichev and du Boulay, 1989) with a strong emphasis on giving information about data structures and variable binding in the trace output. It also has some other features not present in Spy, such as information on reasons for failure of goals. The tracer generally makes more use of labels and symbols to describe different sections of a trace output, and a wide range of options for adjusting the degree of detail shown. It is expected to be more helpful (than Spy or TPM) for solving problems involving data structures and variable binding.

Task and Motivation

The task was designed to evaluate differences in performance due to tracer type. Problems used in this study can be roughly divided into two distinct groups. Their grouping was crudely determined by how the solution depended on certain aspects of Prolog. Solution to three problems depended on information related to backtracking, clauses tried and undefined predicates. And solution to two more problems depended on aspects of information about recursion, system goals, goals with variables, and list manipulation. The reader need not be fully conversant with these aspects of Prolog. It will suffice to comprehend that these categories refer to distinct aspects of Prolog and that ease of access to information about them is helpful in programming and debugging tasks in Prolog. The experiment examined differences in performance due to tracer (or format) type. Since information about certain aspects of Prolog varied across tracers, as far as possible problems were designed to address information about aspects of Prolog which were common to all three tracers. So, for example, no problems directly involving cuts were included because the big difference between the information provided by Spy (which is implicit) and TPM (which is very clear). Similarly solutions to problems that depended on explicit information about system goals were excluded because Spy and EPTB, unlike TPM, do not provide explicit information about them. Problems related to such obvious differences in the extent and quality of information were avoided. Reference to quality is important because often what is not explicitly stated in a tracer can often be derived from available information on other aspects of the trace. This makes it more difficult to distinguish clearly between explicit and implicit information in trace outputs, which is also affected by format.

The null hypothesis is that there will be no difference in time or response due to trace output. In other words, format is expected to have no effect on either the speed or accuracy of performance. If, however, a significant difference in subjects' performance is evident then it was assumed that this would be due to format and its effect on accessibility to information. This follows from the assumption outlined above that none of the problems were biased in favour of a partic-

Problem	Group 1			Group 2		
Tracer	1	2	3	4	5	Mean
TPM	55.2	87.1	60.2	75.3	131.9	81.9
Spy	63.8	102.7	77.3	107.4	163.9	103.1
EPTB	70.4	75.8	81.7	69.7	112.4	82.0
Mean	63.1	88.6	73.1	84.2	136.1	

Table 1: ST's (secs.) of Problem by Tracer (n=43)

Problem	Group 1			Group 2		
Tracer	1	2	3	4	5	Mean
TPM	90.7	88.4	72.1	34.9	41.9	66.6
Spy	93.0	58.1	76.7	11.6	27.9	53.5
EPTB	86.1	76.7	79.1	81.4	72.1	79.1
Mean	89.9	74.4	76.0	42.6	47.3	-

Table 2: % Correct Response of Problem (n=43)

ular type of tracer. The preliminary results presented here suggest an interesting interaction between format and information.

Method

All subjects had been exposed to the Spy tracer and were given a computer-based tutorial designed to explain TPM and EPTB. On reaching a required level of competence in using these tracers subjects had to solve 15 Prolog problems with the aid of trace outputs. The same problem, suitably disguised, was presented three times; each time with a different trace output. Problems were presented in a pseudo-random order. All problems had multiple choice responses. Subjects were asked to solve the problems by referring to the accompanying trace output and to do so as accurately and as quickly as possible. Forty-three subjects completed this task, which together with the initial tutorial took about one hour. Data on time spent on solving each problem and correct response were collected.

Results

Solution times (ST) ANOVA was carried out with subjects as the random factor and Tracer (3 levels) and Problem (5 levels) as fixed factors. All the data was included in the analyses. There was a significant main effect of tracers, $\mathbf{F}(2,84) = 11.32$, $p \leq 0.001$, indicating that ST is dependent on tracer type. Overall, subjects spent significantly more time on attempts to solve problems with Spy, than TPM or EPTB. There was a significant main effect of problems, $\mathbf{F}(4,168) = 42.14$, $p \leq 0.001$. Variance in problems difficulty would be expected to have this effect on ST. Broadly speaking, Group 1 problems (1, 2 and 3) were easier to solve than Group 2 problems (4 and 5), as is evident from ST's in Table 1. However, far more interesting is the significant interaction between tracers and problems, $\mathbf{F}(8,336) = 3.53$, $p \leq 0.001$. ST's vary according to tracers for each problem, and these differences are not consistent across tracers. For example, TPM aided solutions to problems 1 and 3 require less time EPTB . The reverse is the case for the remaining 3 problems. Subjects are particularly slow at solving problems 2, 4 and 5 with Spy. These sorts of differences in mean ST's reflect differences in format, and therefore, ease in access to information necessary to solve problems. For instance, subjects' faster solutions to problems 1 and 3 with TPM is probably due to their superior display of information which can be taken in 'at a glance' which is particularly useful for solving those problems.

Differences in ST's are more interesting when jointly considered with response data. An ANOVA similar to that of ST data was carried out on responses. There was a significant main effect of problems, $\mathbf{F}(4,8) = 32.89$, $p \leq 0.001$, which reflects the varying level of difficulty of problems. On average subjects found Group 1 problems easier to solve. There was also a significant main effect of tracers, $\mathbf{F}(2,8) = 23.06$, $p \leq 0.001$. Overall subjects performed best with EPTB and worst with Spy, as illustrated in Table 2. There was a significant interaction between tracer and problem, $\mathbf{F}(8,336) = 8.92$, $p \leq 0.001$. Subjects performed less well in solving Group 2 problems with with TPM and Spy than they did with EPTB. Taking into account the ST's for EPTB, the correct response data indicates that the extra time was well spent. For Spy, the results show an inverse relationship; higher ST's for problems 2, 4 and 5 are reflected in far fewer correct responses. Even when trying hard subjects were encountering severe difficulties in solving problems (particularly, problem 4) with Spy. Similarly, TPM was less helpful than EPTB for solving problems 4 and 5, though there is no consistent correlation (inverse or otherwise) with ST's.

Apart from Spy accompanied with problem 2, all tracers seem to be equally useful for solving Group 1 problems. But they take different ST's, which indicates that though all three traces have the information necessary to solve these problems, access to it is significantly affected by format. Given that Group 1 problems are less difficult to solve than Group 2 problems, this indicates EPTB is particularly helpful when solving more difficult or intricate problems in Prolog.

Discussion

The results show a major effect of tracer format on subjects' ability to solve Prolog problems. Overall, and as expected Spy had the most shortcomings. It does not provide information on system goals, and the simple, linear text format is not good at conveying information about backtracking, more complicated retry clauses, and information about variables involving list manipulations. TPM is good at giving information about system goals and provides list manipulations' information that *looks* less cluttered than Spy or EPTB. However, EPTB is better at displaying information about retry clauses which explains the high frequencies of correct

solutions to Group 2 problems. EPTB's good performance across all problems reflects the benefits of a format that eases access to a range of useful information. This results in a modest increase in ST's as users have to work through a linear presentation of information, not allowing them to be as selective about focusing on specific chunks as in TPM. EPTB cannot exploit the advantages of spatial representations for those aspects of Prolog which can be better presented in graphic formats such as TPM. However, this shortcoming is offset by other advantages such as better display of information on list manipulations and retry clauses, both of which do not suffer from the constraints of graphic format - this is particularly true for *long* list manipulations.

The linear nature of textual formats imposes fewer constraints which partly explains EPTB's better performance, and the remarkable difference in TPM's performance between Group 1 and Group 2. Preliminary analyses of ST's and responses highlight two major effects of format on information accessibility. First, graphic formats benefit from greater impact and clarity, and perform best for solving problems associated with the relationship with various clauses in a Prolog program. Second, this advantage only holds for simple programs with short list manipulations and relatively few retry clauses; programs with long list manipulations or many retry clauses severely test the limitations of graphic tracers such as TPM version evaluated in this study. Graphic formats are far more constrained by a lack of space than linear textual formats. In the latter more space is created by simply adding another line; things are rarely that easy to fix in a graphic format tracers, particularly those, like TPM, which represent certain basic information in an AND/OR trees. Augmenting AND/OR trees with textual information can help overcome some of these constraints to a limited extent. However it is not clear that this difference would scale up to large programs written by experts where it has been argued (e.g., Eisenstadt & Brayshaw 1988) that information about the overall *shape* of the Prolog execution tree is advantageous.

Given the findings reported here it seems that for the presentation of information about Prolog programs executions, a textual format tracer would be adequate for novice programmers. This does not imply that graphic formats are inherently unsuitable, but the results indicate that advantages of graphic format vary between different aspects of Prolog. For complicated and inter-related information such as that of a Prolog program execution, gains in clarity of graphic format result in less than adequate treatment of related aspects (though a graphic based approach is probably far more effective for teaching Prolog). Linear textual formats are not similarly constrained; more than one aspect of a complex piece of information can be simultaneously presented.[1] We are at present working on a prototype (mainly textual) tracer tool which incorporates the relative advantages of each format type (Taylor, du Boulay & Patel 1991). It seems that though format has a significant effect on ease of access to information for problem solving, much of this effect is modified by other variables such as the amount and quality of information and its relevance to problem solutions, and until we have a much better understanding of this complex interaction, linear textual tracers such as EPTB are more effective in helping Prolog programmers solve programming problems.

[1] This approach could increase the time required to assimilate all the information but that is a separate issue.

Acknowledgement

We acknowledge comments from R. Noble and M. Eisenstadt.

References

Byrd, L. 1980. Understanding the control flow of Prolog programs in Tarnlund S. ed. Proceedings of the Logic Programming Workshop, 127-138.

Clocksin, W.F. and Mellish, C. 1981. *Programming in Prolog*, Springer Verlag.

Dichev, C., and du Boulay, J.B.H. 1989. An Enhanced Trace Tool for Prolog. In Proceedings of the Third International Conference, Children in the Information Age, 149-163. Sofia, Bulgaria.

Eisenstadt, M. and Brayshaw, M. 1988. The transparent Prolog machine (TPM): An execution model and graphical debugger for logic programming. *Journal of Logic Programming* 5(4):277-342.

Hook, K., Taylor, J. and J.B.H. du Boulay. 1990. Redo "Try Once And Pass": the influence of complexity and graphical notation on novices' understanding of Prolog. *Instructional Science* 19 (4-5):337-360.

Pain, H. and Bundy, A. 1987. What stories should we tell novice Prolog programmers? In Hawley, R. ed. Artificial Intelligence Programming Environments. Ellis Horwood.

Taylor, C., du Boulay, J.B.H., and Patel, M.J. 1991. Outline Proposal for a Prolog 'Textual Tree Tracer' (TTT), Cognitive Sciences Research Paper-177, School of Cognitive Sciences, The University of Sussex.

Shu, N. C. 1988. *Visual Programming*, New York: Van Nostrand Reinhold.