Two-dimensional Visual Programming and Three-Dimensional Execution Visualisation in Prolog

Conference or Workshop Item

TWO-DIMENSIONAL VISUAL PROGRAMMING AND THREE-DIMENSIONAL EXECUTION VISUALISATION IN PROLOG

Simon Holland
Computing Department
The Open University, Walton Hall, Milton Keynes MK7 6AA, England
E-Mail: s.holland@open.ac.uk

Abstract
A new, simple, expressively complete visual formalism for programming in Prolog is presented. The formalism is shown to be equivalent to the standard textual notation for Prolog. Some aspects of Prolog programs are identified that appear to be clearer for novices when presented in the graphic formalism, while other aspects of Prolog are noted that may be clearer in the standard textual notation. The design of an implemented computer environment dubbed VPP (short for "Visual Programming in Prolog") is presented that supports visual programming in Prolog using the graphical formalism. Two different implemented experimental prototypes of VPP are discussed.

This paper appeared as
Holland, S. (1991) Two-dimensional Visual Programming and Three-dimensional Execution Visualisation in Prolog. In *Colloquium Digest of the Institute of Electrical Engineers' IEE Colloquium on Visualisation, Virtual World, Virtual Reality*. Pages 4/1-4/4. IEE Electronics Division, Savoy Place, London.

TWO-DIMENSIONAL VISUAL PROGRAMMING AND THREE-DIMENSIONAL EXECUTION VISUALISATION IN PROLOG

Simon Holland

A new, simple, expressively complete visual formalism for programming in Prolog is presented. The formalism is shown to be equivalent to the standard textual notation for Prolog. Some aspects of Prolog programs are identified that appear to be clearer for novices when presented in the graphic formalism, while other aspects of Prolog are noted that may be clearer in the standard textual notation. The design of an implemented computer environment dubbed VPP (short for "Visual Programming in Prolog") is presented that supports visual programming in Prolog using the graphical formalism. Two different implemented experimental prototypes of VPP are discussed.

An extension of the programming environment is described that allows Prolog execution spaces to be visualised in complete detail (or presented in various compressed, pruned or abstracted forms) using a simple three-dimensional extension of the same formalism. This approach is unique in that the same formalism can be used both for visual programming, and then ('stacked' in three dimensions), for complete visualisation of execution. This appears to offer two major advantages over other approaches described in the literature for visual programming or execution visualisation. Firstly, only one simple formalism need be learned, by contrast with systems where two different formalisms must be learned (and mentally interrelated) for programming and execution visualisation. Secondly, compared with systems that use only two dimensions for execution visualisation, clutter and complexity is greatly reduced, and multiple interrelationships can be shown clearly without a need to switch representation.

A prototype of the execution visualisation environment (as opposed to the visual programming environment), dubbed VPE - short for "Visualising Prolog Execution" - is currently under construction. VPE is shown to provide complete logical information on Prolog execution (as does the Transparent Prolog Machine (TPM), due to Eisenstadt and Brayshaw (1987)). Note that, unlike TPM, VPE has the complementary VPP system (which uses essentially the same formalism) to provide full facilities for visual programming. Relationships are identified that are more directly expressed in VPE than in TPM. Particular pruned views of VPE traces are noted that allow recursion to be visualised in an intuitively satisfying nested "Russian doll" fashion. In order to distinguish the formalism for visualising execution spaces from the environment (VPE) that uses the formalism, the notation for visualising execution spaces is dubbed "3D-Prolog execution notation".

A further extension of VPP and VPE for visualising and manipulating lists is presented that can be used to help make clear the action and purpose of commonly occurring list unification programming techniques.

Some widely used prototypical Prolog programming techniques are identified which appear to be particularly lucid in the VPP formalism for lists. It is argued that translation of a library of prototypical Prolog "techniques" into the visual formalism and their examination in VPE may be a valuable way of helping novices to learn key Prolog programming skills.

Uses for VPP and VPE in teaching Prolog to novices, and in building domain specific application kits are discussed. A simple factory construction metaphor or "story" is presented to help novices make sense of Prolog execution traces. The metaphor distinguishes in a detailed way between features of pure logic programming and "impure" procedural features like cut, not, assert, etc. The metaphor makes this distinction by means of a detailed contrast between, on the one hand, assembling machines in a factory in an orderly fashion from components and blueprints, and on the other hand "trades union" activities such as "cut" and "not" that restrict or alter normal working practices. This metaphor may be particularly helpful in helping beginners to understand backtracking, recursion, negation, cut, etc.

As well as supporting the factory metaphor, VPE is shown to have good low-level perceptual visuo-spatial properties in allowing users to retrace backtracking behaviour continuously with a finger in a "natural" way.

Connections with related work on graphic formalisms for programming in Prolog and Prolog execution visualisation are noted (e.g. Kahn and Saraswat, 1990). Connections with recent work on 3D techniques for the visualisation of flat trees using 3-dimensional cone trees, cam trees, etc. at Xerox PARC are noted. We informally analyse the structure and properties of the notation from an abstract human-machine interaction viewpoint. Limitations and possibilities for further work are identified and discussed.

Dr S.Holland is a Lecturer in Intelligent Tutoring Systems at the Department of Computing Science, Kings College, University of Aberdeen, Aberdeen, Scotland, AB9 2UB.
Tel: 44 224 27 228 Fax: 44 224 48 7048 Email (Janet): simon@uk.ac.abdn.csd

Illustrations

Space limitations make it impossible to give a full illustration of VPP and VPE, but the following figures give a flavour. Figure 4 shows a slightly idealised view of the VPP environment for visual programming in Prolog. Figure 1 shows a relation represented in VPP, and figure 2 shows clauses with shared constants. (Note that there is no *requirement* to show "sharing" of constants (as opposed to "repeating" the constants where needed), but it is allowed and can aid clarity in some circumstances.) Figure 3 shows a rule. Figure 5 shows the complete execution trace of a simple program in VPE that demonstrates many of the features of Prolog execution that novices can find confusing: backtracking, initial success that fails on subsequent backtracking, cut and negation. Figure 6 illustrates one of the ways of representing list relations in VPP and VPE. Figure 6.1 and 6.2 show the two clauses of a simple append procedure. Figure 6.3 shows a "polar" (or simplified) view of the execution of the program in appending a two element list to a a list.

Figures

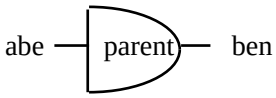

Fig. 1. A relation in VPP.

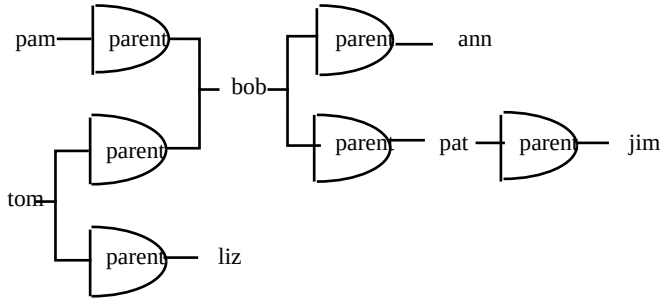

Figure 2. Clauses with shared constants in VPP.

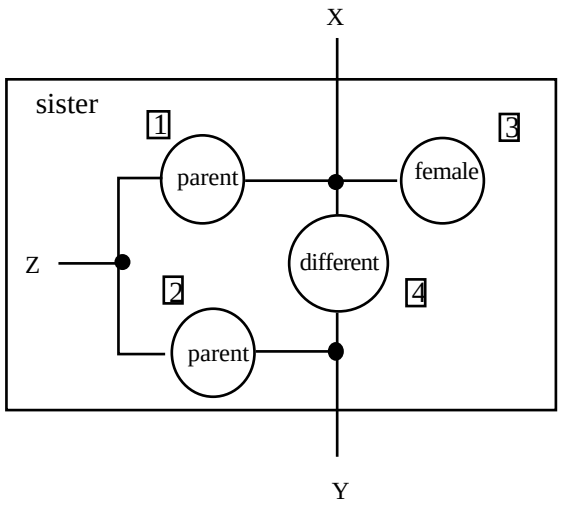

Figure 3. A rule.

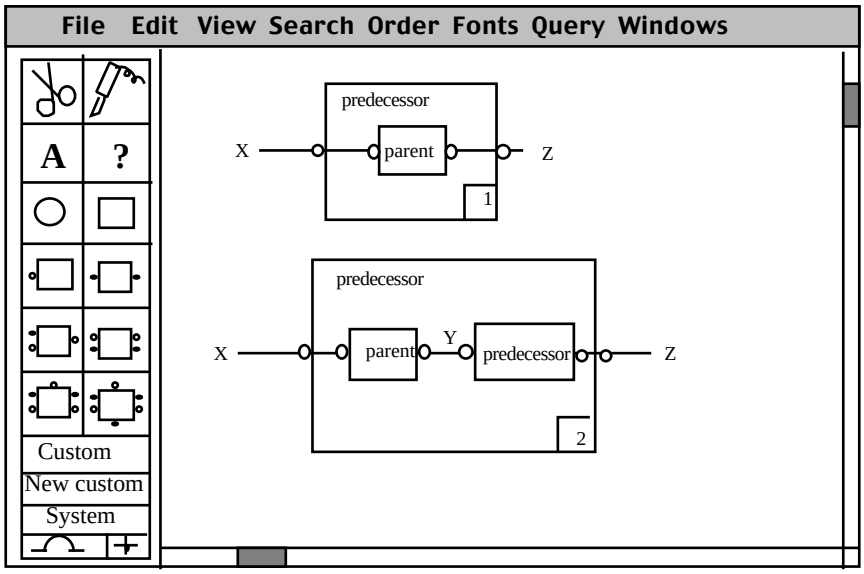

Figure 4. A slightly idealised view of the VPP environment.

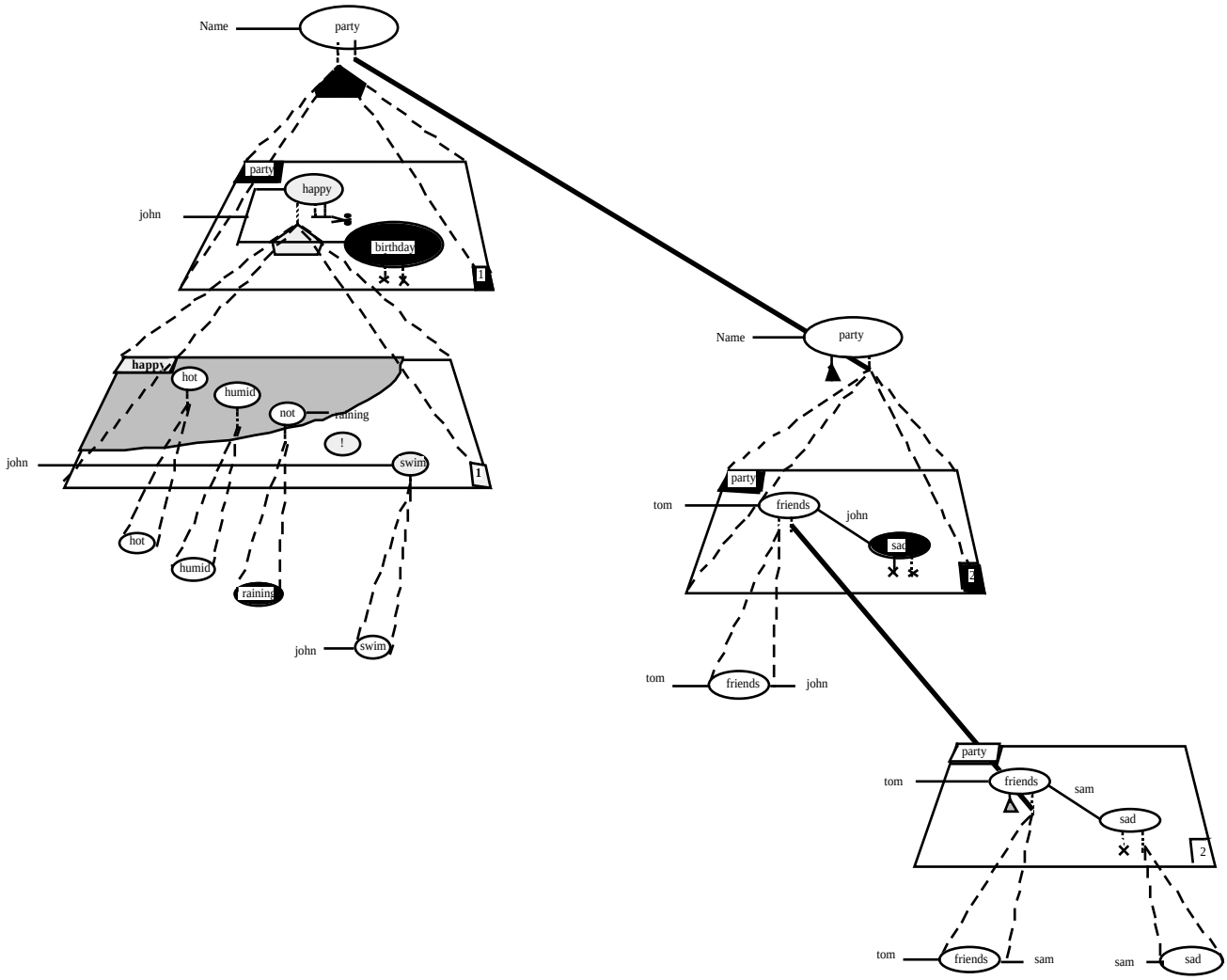

Figure 5 An execution trace

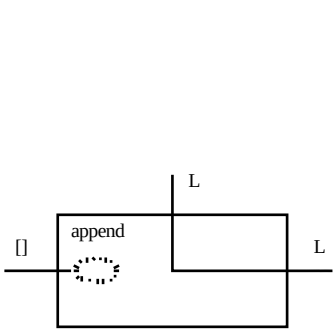

Fig 6.1 append([],L,L).

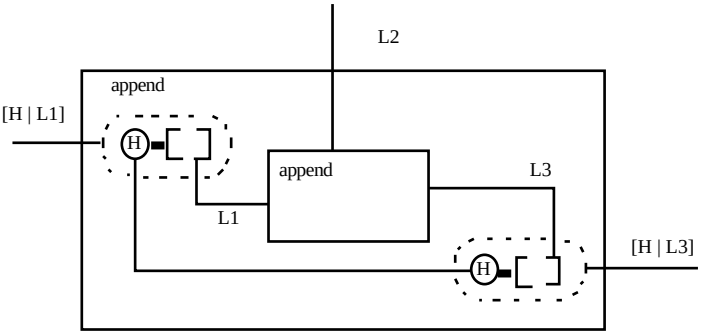

Fig 6.2 append([H | L1], L2, [H | L3]]:-
append(L1,L2,L3).

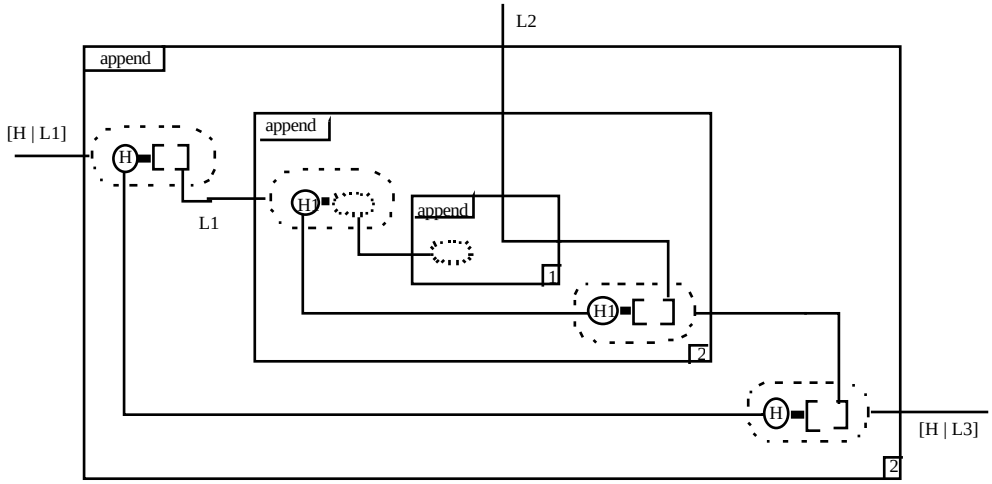

Figure 6.3 Appending a list to two element list (top or "polar" simplified view)

Figure 6: This set of three figures shows a simple append program and one case of its execution. The figures illiustrate one VPP convention for representing lists.

Acknowledgements

This paper is an extended abstract for a revised and extended version now in preparation of Holland (1991). Thanks to David Philip and Mark Treglown for their work on implementing the prototypes of VPP, and to them and Kristina Hook, Ben duBoulay, Thomas Green, Pat Fung, Paul Brna, Tony Priest, Josie Taylor, Rob Lucas and Peter Gray for helpful comments and discussions.

References

Eisenstadt, M. and Brayshaw, M. (1987) An integrated textbook, video and software environment for novice and expert Prolog programmers. *Proceedings of the 2nd International Conference of the Prolog Education Group (PEG 87),* Exeter, UK 8th-10th July 1987.

Holland, S. (1991) Visual Programming in Prolog. Proceedings of the Sixth International PEG Conference on Knowledge-Based Environments for Teaching and Learning. P. 676-684 Rapallo (Genova), Italy, May 31-June 2, 1991.

Kahn, K.M. and Saraswat, V.A. Complete visualisations of concurrent programs and their executions. In IEEE workshop on Visual Languages IEEE Press 1990.