Visual Programming and Visualisation of Program Execution in Prolog

Simon HOLLAND

Department of Computing Science
Kings College
University of Aberdeen
Aberdeen
Scotland AB9 2UB

Tel : 44 224 27 2284
Fax : 44 224 48 7048
email (Janet) : simon@uk.ac.abdn.cs

Extended 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 a 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 presented 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. 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 view.

A prototype of this environment, dubbed VPE - short for "Visualising Prolog Execution" - is currently under construction. VPE is shown to provide complete information on Prolog execution (as does the Transparent Prolog Machine (TPM), due to Eisenstadt and Brayshaw (1987) - although TPM has no facilities for visual programing). 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 "Russion doll" fashion. Note that 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 assembling machines in a factory in an orderly fashion from components and blueprints, and "trades union" activities such as "cut" and "not" that restrict or alter normal working practices. This metaphor seems to 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. 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.

Finally, it is shown how VPP and VPE can be extended into a domain independent graphical logic programming tool kit (dubbed the Picture Machine) adaptable to become a domain-specific application kit in any given suitable domain. It is required that there should exist a mapping from a given space of domain-specific diagrams into relations representing the meaning of the diagrams. The Picture Machine (currently under implementation in prototype) should allow non-programmers interested in some domain to manipulate domain-specific diagrams as a way of querying and reprogramming existing domain-specific logic programs.

1

Interim Report on

Visual Programming & Visualisation of Program Execution in Prolog

Simon Holland
simon@uk.ac.abdn.csd

Department of Computing Science
Kings College
University of Aberdeen
Aberdeen
Scotland AB9 2UB

2

OVERVIEW OF TALK

- Visual programming (VPP)
- Vis of program execution (VPE)

- Unique features of VPP + VPE
    - integrated system: one formalism
      prog + complete exec space
    - factory metaphor: non-progs?
    - exec model uses 3 spatial dims

- Visual programming in Prolog
- Visualisation of Prolog execution
- Factory Metaphor
- Visualising list processing

- Implementations
- Related systems,comparisons, origins

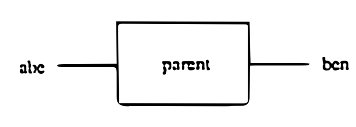
- Hypotheses about VPE and VPE
- Limitations & further work
- Summary & Conclusions

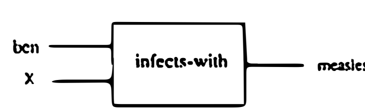
3

VISUAL PROGRAMMING IN PROLOG

Facts in a database in VPP

parent(abc,ben).
infects-with(ben,X,measles).





- Constants and variables - links
- Relations - boxes

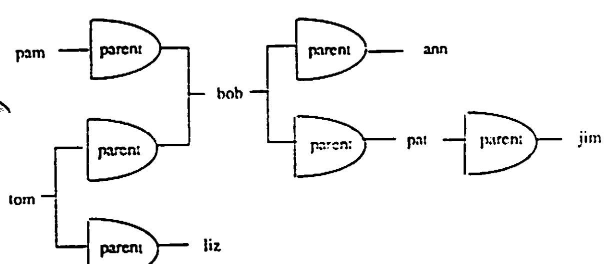
- Box shape does not matter
  (just number of ports and name)

- Upper and lower-case distinction for
  variables/constants as usual

- Ordering of clauses in database
    - left to right, top to bottom
    - but optional numbering system
      spatially ordered view
      numbering system ordered view

4

Clauses with shared constants in database



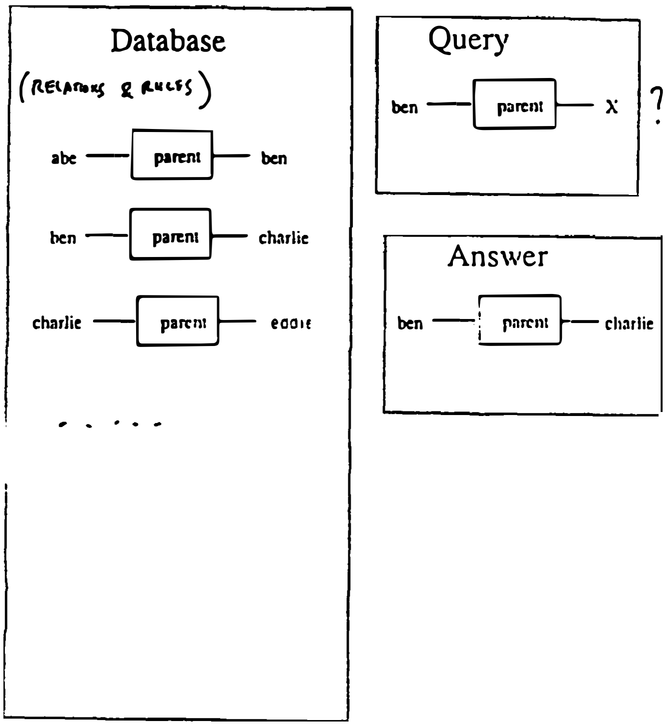
parent(pam,bob).
parent(bob. ann).
parent(tom,bob).
parent(bob, pat).
parent(pat, jim).
parent(tom, liz).

- "Common" display of atoms not compulsory
- Can be displayed as separate clauses
- In some situations, can help to show potentially inferrable relationships easily
- NB in complex situations, this style of display may not be helpful

- Editor does not allow shared *variables* between clauses
    - except in queries
    - except within rules
    - (no conjunctive clauses allowed in database)

6

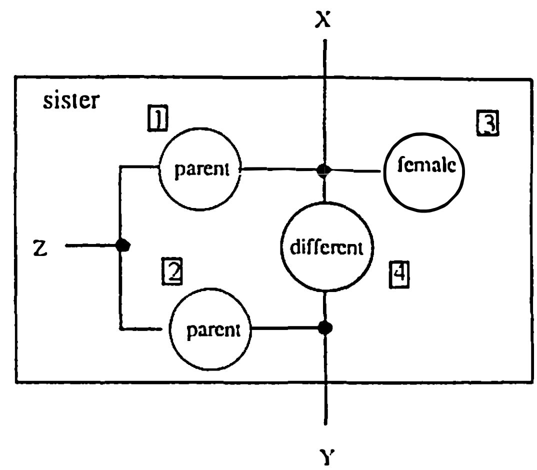
SCHEMATIC OVERVIEW



5

Rules

```
sister(X,Y):-
        parent (Z,X),
        parent(Z,Y),
        female(X),
        different(X,Y).
```



- Within a rule, optional variable & constant sharing (e.g X,Y,Z above)
- As with clause order in program, clauses in rule ordered left to right top to bottom.
- Adjust clause order by moving clauses physically
- Optionally, may use (and alter) numbers to overide default ordering

8

Metaphor /stories (ref)

Logical view

- Program = set of axioms,
- Computation = constrc prf of goal stat from prog.

Constraint satisfaction view

- program = set of constraints, rels or specifications,
- computation = constrc of entity to satisfy constraints.

Advantages

- within familiar experience of beginners
- links to logical account
- extends well to impure aspects of prolog

3 areas on screen,

- stores / warehouse
- the order book
- construction area

Warehouse carries 2/3 kinds of stock,

- objects
- templates
- blueprints

Stock laid out in order to be searched.

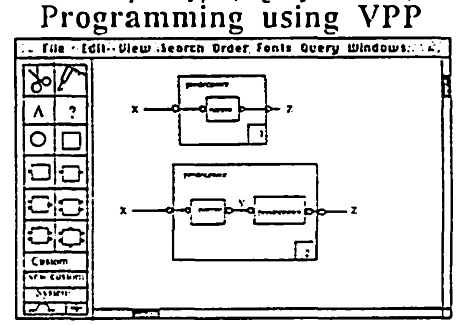
Pure Prolog

- "flashing" as stock inspected,
- copy of matching stock moved to construc area
- "exploded diagram" metaphor
- requests for an alternative design...
- partial failure
  - Sub-component breakdown - 'Polar view' - recursion

7

Current prototype (slightly idealised)

Programming using VPP



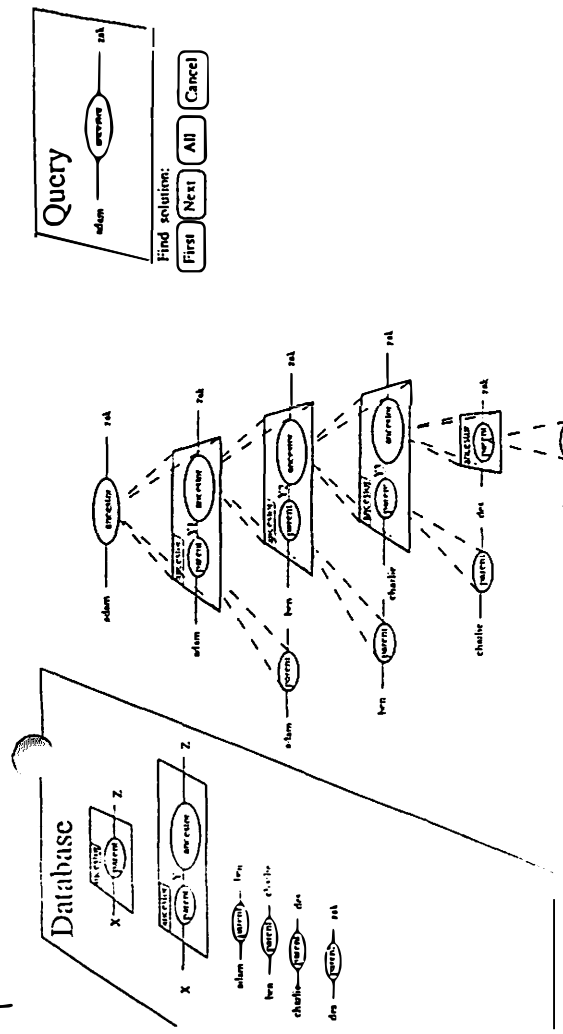
- menu & strip of graphical tools
- windows for - prog/query/answer
- soldering iron to connect up boxes
- scissors

- boxes types to choose from
- typing tool to name boxes and variables

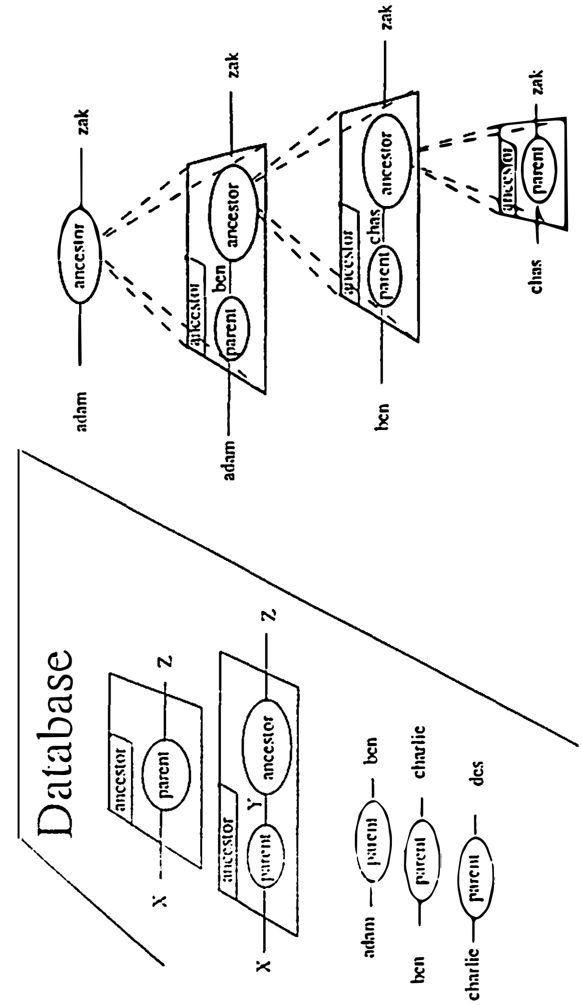
- Boxes may be grown or shrunk for rules.
- Boxes can be moved or deleted.
- Moving boxes en masse - watch wires
- Any size programs - scrollable window
- Can generate text prolog in new window

- magnified, reduced & alternative views
- indexes and find functions
- numbering tool
  - clauses within programs
  - goals within rules
  - docks within a box.

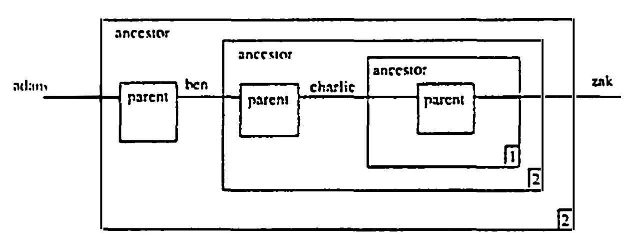
9



10



11



12

Impure features of Prolog: Trades Union Metaphor

• Cut , not, assert, etc

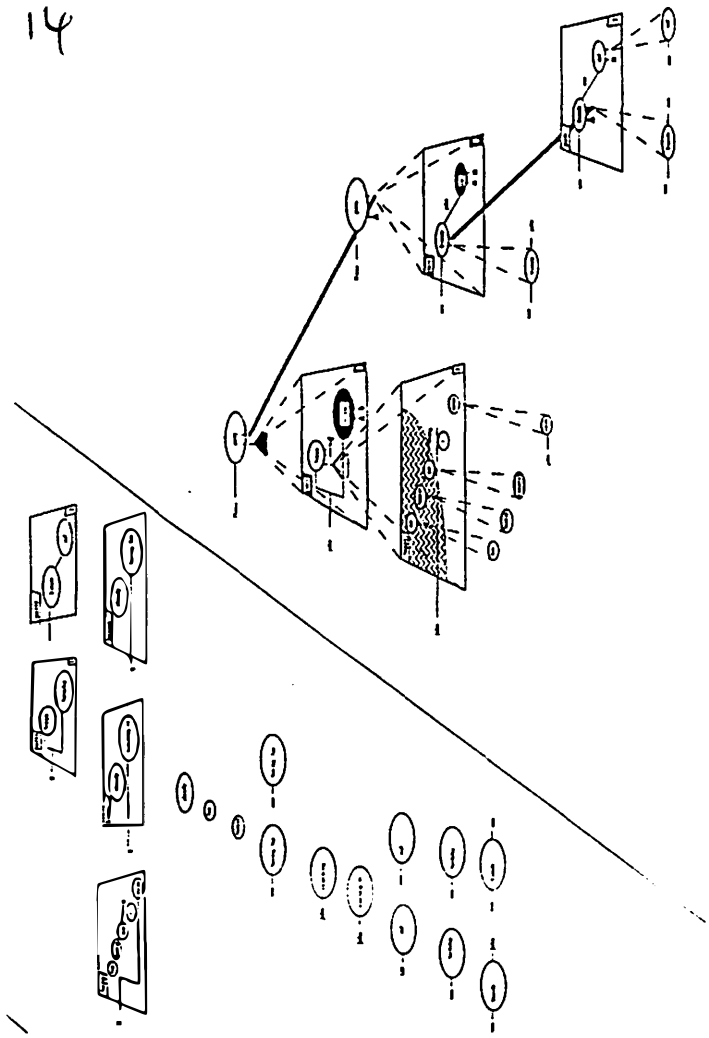
Contrast
assemble components & blueprints,
• alteration of normal work practices.

Cut
• restrictive prac blt into blueprint.
• cordons off any work already done to satisy a blueprint up to ! instruc
• any demand to re-do work in cordoned-off area refused.
• ban on scabbing - no alt blueprints to one subj to indust action
  • symbol...

not
cut fail blueprint,
or, boycott,

if *not(sthAfricOrnges)* in blueprint,
provided *sthAfricOrnges* not in stores,
assembly may proceed
If, *sthAfricOrnges* found in stores,
'not 'operation fails

14



13

Backtracking, cut, not, etc

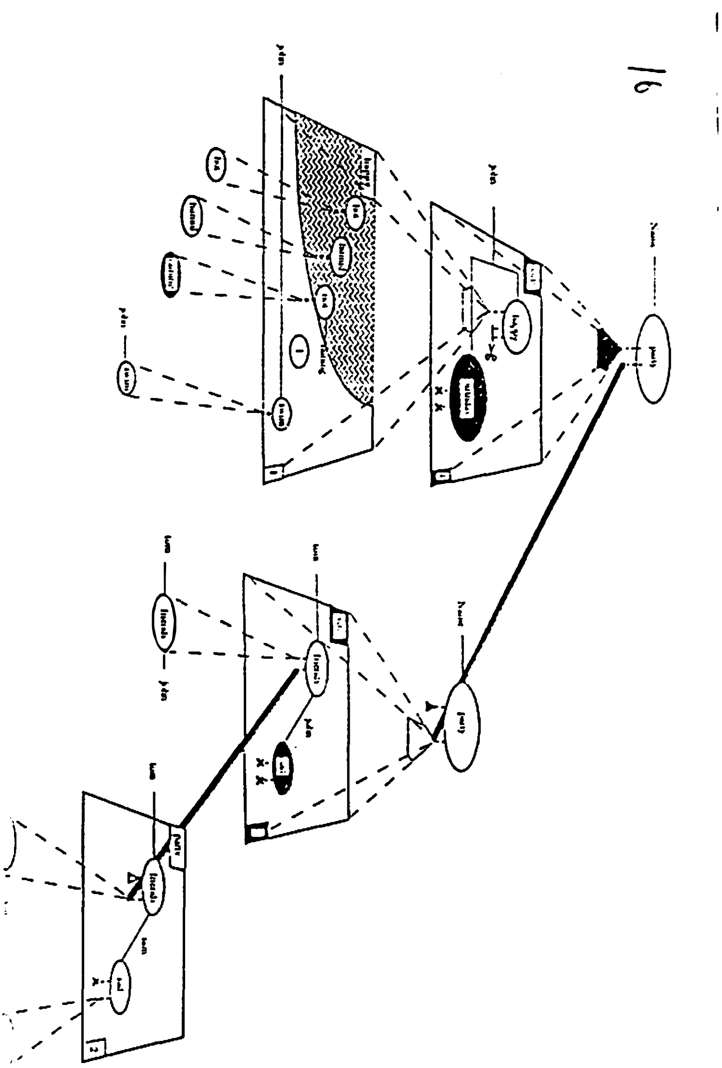
```
party(X):- happy(X), birthday(X).
party(X):- friends(X,Y), sad(Y).
happy(X):- hot, humid, not raining,!,
swimming(X).
happy(X):- cloudy, watching_tv(X).
happy(X):- cloudy, having_fun(X).
cloudy.
hot.
humid.
having_fun(tom).
having_fun(sam).
swimming(john)
watching_tv(john).
sad(bill).
sam(sam).
birthday(tom)
birthday(sam)
friends(tom,john).
friends(tom,sam).
```

Figure 10. A simple example program reproduced from Eisenstadt and Brayshaw (1987).

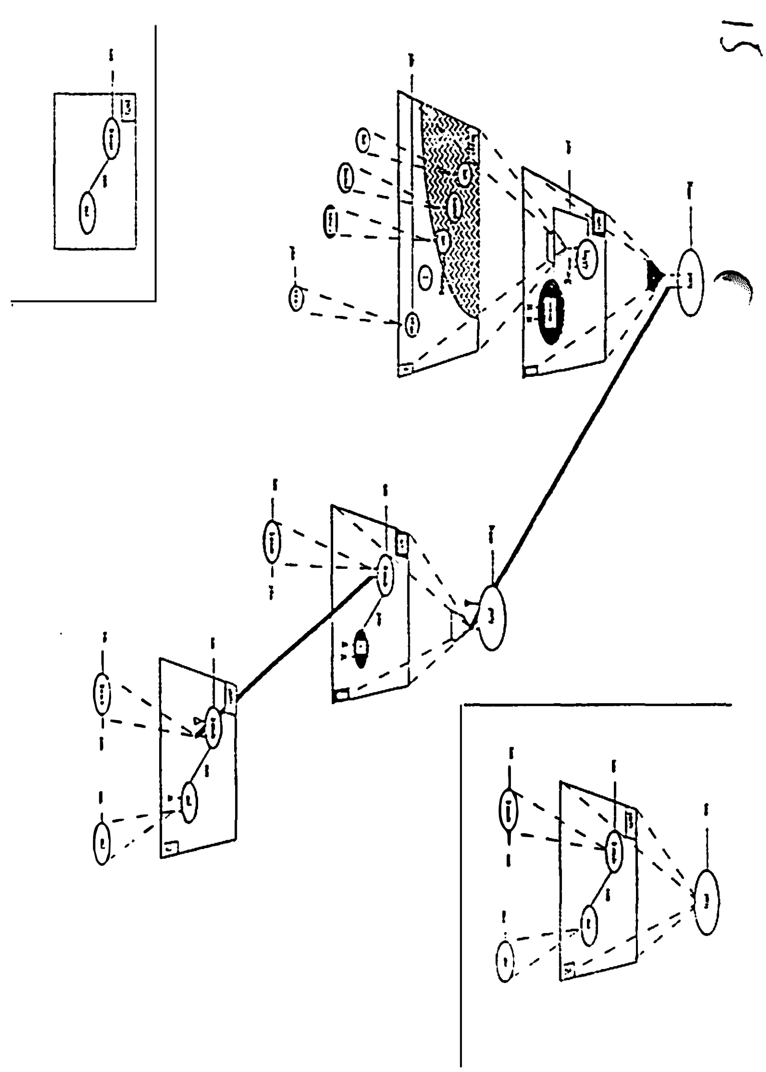
query

party(Name)?

16



15

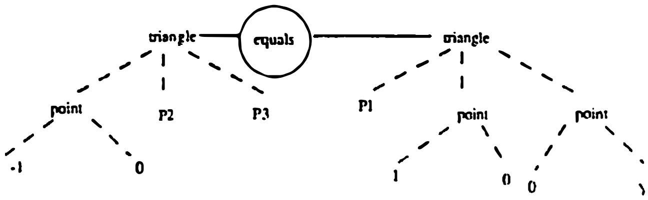


18

Structured terms

Example

equals(triangle(point(-1,0), P2, P3)),
(triangle(P1, point(0,1), point(0,Y)).

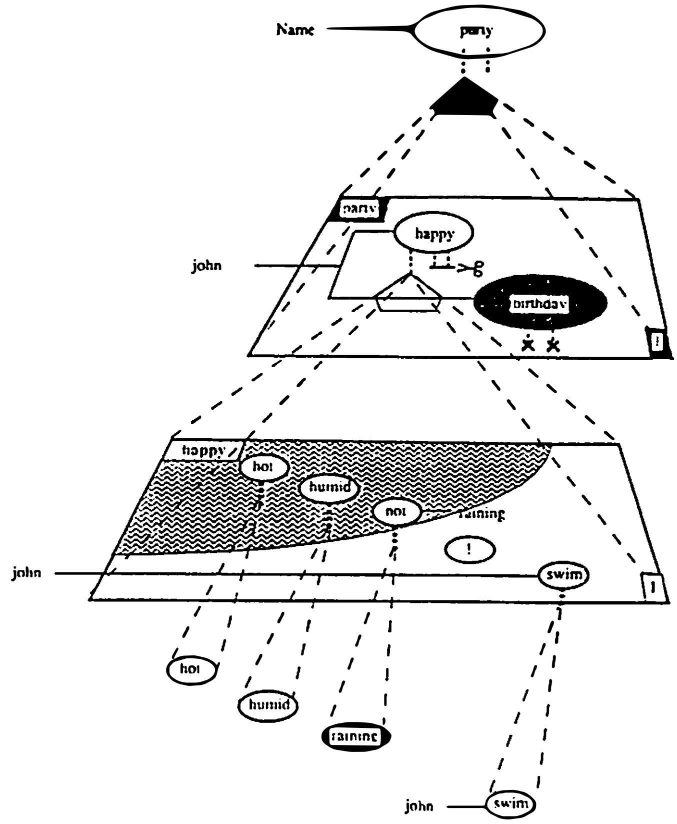


Compound terms (structures)
can be viewed as tree-structured variables.

Must distinguish
- lines showing common occurrences of terms and
- lines showing tree structuring of variables

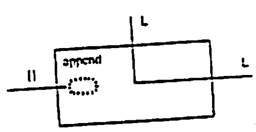
Nesting of components
- nesting of functors and terms within structures shown by dotted lines.

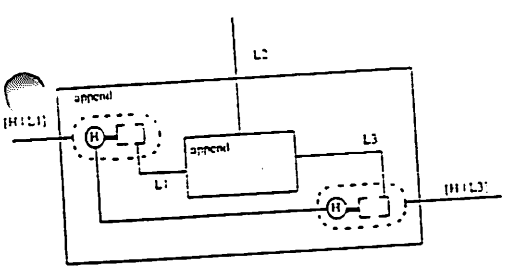
17



NOTATION FOR LISTS

20

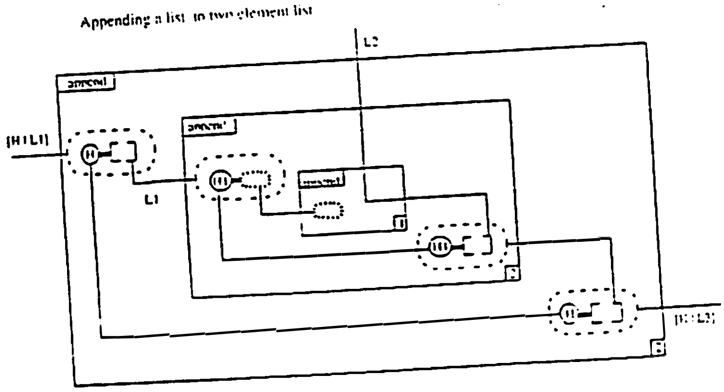




append([],L,L).

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

Appending a list to two element list



19

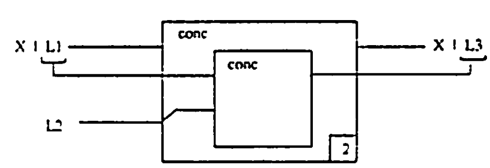
List processing

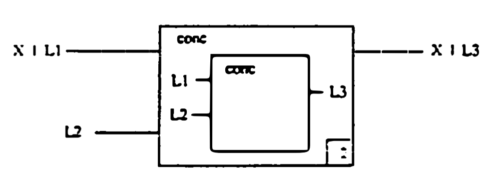
Lists are special case of tree-structured variables
- may be shown as trees using dot functor,
- or conventional textual notation for lists

Identical terms in compound structures
- Need not always be shown by single graphical instance
- Sometimes positively hinders clarity

Example of *unclear* diagram for a rule

conc(X|L1, L2, X|L3):-
conc(L1, L2, L3).





Present recommendation
- show lists conventionally
- dont try to show common terms if unclear

22

Prototype IMPLEMENTATIONS of VPP

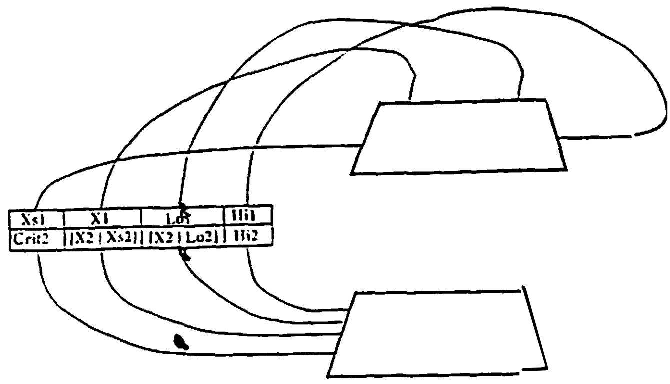
Philip (1991)
Sunview C SPARC
Generates textual prolog code from diagrams
Ad-hoc
Works but has some bugs

Treglown (1991)
X-windows C SPARC
More systematic, uses formal visual grammar ect
incomplete

Larger than a student project.....

21

SHOWING DETAILS OF UNIFICATION



24

VIS PROG IN PROLOG - RELATED WORK

KAHN & SARASWAT
XEROX PARC (1990)

LADRET & RUEHER (1991)

PAU & OLASON (1991)

VIZ OF PROGRAM EXECUTION (LO-TECH)

TPM - Textual, no programming
KAHN & SARASWAT - very hard to read

ROUGH SCULP ↓ GE 3D

INFO VISUALISER
(XEROX PARC)

3D ROTATE ETC

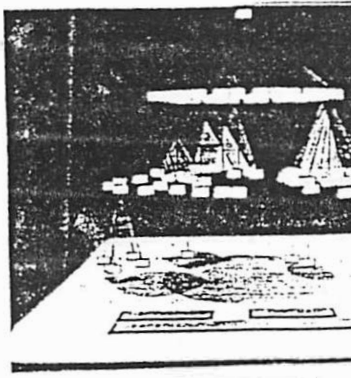
23

Related Work

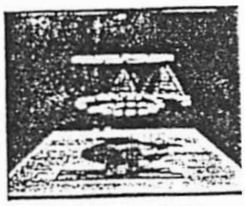
VPP
- Kahn & Saraswat (Xerox Parc 1990)
  - programming similar?
  - not optimised for clarity
  - no 3d execution model - storyboard
- Ladret & Rueher
  - Neat connectivity idea (could borrow)
  - programming rather different
  - no execution model

Kurita & Tamura
- Programming similar
- not so developed
- no execution model

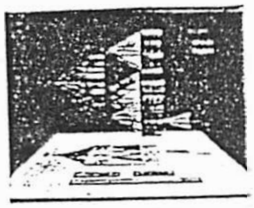
VPE
- Dewar & Cleary
  debugger only -
- Vizzprol
- TPM - best graphic tracer
  - no graphic prog lang associated
  - diff notation for prog & exec model
  - some alternatives not shown
  - does not take advantage of 3D
  - prob better for professs progs (so far)
- Colgan Rankin Spence (Imperial)
  - not Prolog: Eng design
  - some similarities
- INFO VISUALISER (XEROX)



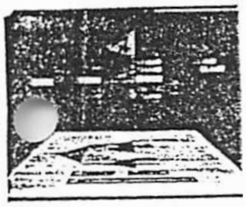
Robertson Plate 1



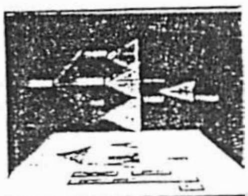
Robertson Plate 2



Robertson Plate 3



Robertson Plate 4



Robertson Plate 5

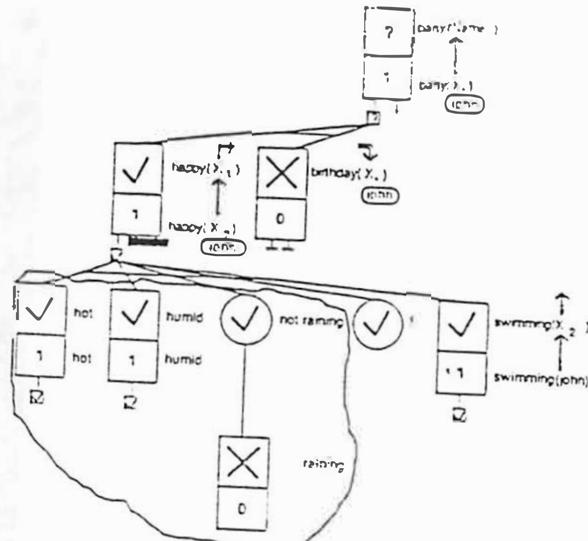
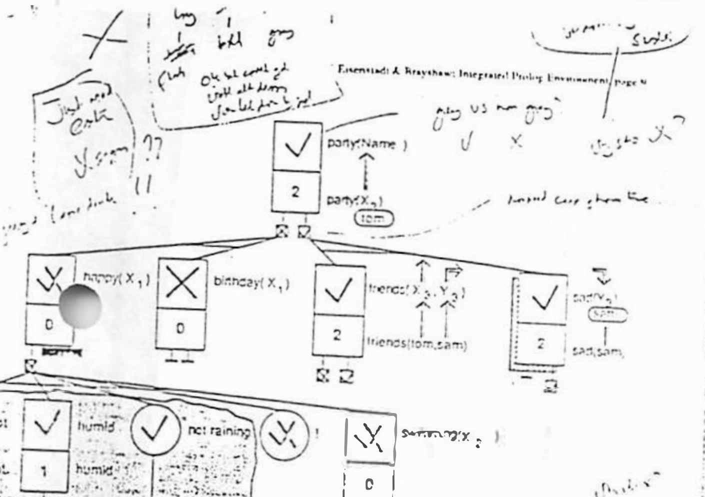


Figure 5 Intermediate AORTA snapshot after the query ?- party(Name). birthday/1 has just failed.



Final AORTA snapshot after the query ?- party(Name). party/1 has just succeeded with Name = tom

...stant that birthday fails, the 'backtracking point' (the 'youngest surviving' heir of the either most ...-failed goal) is the *clause* status box corresponding to clause 1 of swimming in figure 6. The ... for backtracking is to identify the backtracking point, then turn that tick into a tick/cross ...tion, and 'carry on'. Since there are no other clauses for swimming, it fails at this point (hence ...cross combination in its *procedure* status box), and likewise for the cut. There are no other ...r backtracking points (the 'darkening' of the frozen cloud indicates that an attempt was made to ... into it), so clause one of happy receives a tick/cross combination to show that it initially won ...il on backtracking. Clause branches two and three of happy have been eliminated by the cut, (for ...eaten only, of course), so the *procedure* status box for happy becomes a tick/cross combination. The *clause* status box for clause 1 of party therefore receives a cross (failure), and clause two ... now attempted. The friends goal succeeds initially on clause one i.e. friends(tom,john), but ...) fails. After backtracking (which leaves a tick/cross in the *clause* status box for clause 1 ... now friends succeeds on clause two with friends(tom,sam). This leads to a new ... of the sad goal. To indicate that there has been a previous invocation at the same ... ...ch space, a shaded *ghost status box* is drawn. The goal sad now succeeds ... ... Notice in figure 1 that X_3 (and hence Name) is instantiated to tom, Y_3 is ... ...at this instantiation is passed to the goal sad. The goal sad(Y_3) ...

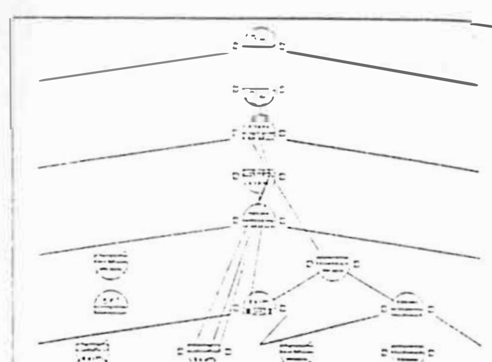


FIG. 7. A Transact display of Pascal statement in fig. 1.

GRAPHICAL DISPLAY ...

...in the tree or in data ... the entire execution ... be too small for deta... and detailed context ... requesting that all ... The new model al... debugger. Typically, ... and the prompt asks ... of a communication ca... format of the de... or wires for a parti... middle of a long pro... any input and outpu...

We solve this by ... a window complex... nodes can be poin... instances of a name ...

An important ... the program is ... which is not obvio... interpreted by the ... may well involve ... views graphically ... routines in a high...

GRAPHICAL DEBU...

Many graphical ... Some of these ... emphasize the ... extent to which ... object also ...

Much of what ... graphical ... monitoring ... some way ... Ass & Krishnat... systems are ... user to see ... where that ... in nature, so ... having to ...

One of the ... of the user ... structure, such ... be displayed ... when the user ...

The Cornell ... him to some ...

parent(X, mother(X)).

Fig. 1 is a corresponding program in Dialeg.l. Procedure declarations are made of following five basic icons.

(1) name-boxes

A "name-box" represents a set of procedure declarations which have the same head predicate name as the label. There are two name-boxes in Fig. 1 with the labels "grand-parent" and "parent."

(2) glass-boxes

A "glass-box" represents a procedure declaration and is an immediate inner square of a name-box. For example the name-box "parent" has two glass-boxes in Fig. 1. The name comes from the sense that the programmer can see its inner structure.

(3) black-boxes

A predicate in the body of a procedure declaration is represented by a "black-box".

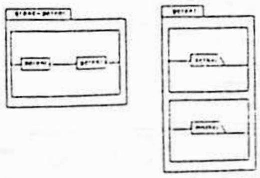


Fig. 1 Pictorial representation of procedure declarations

— 58 —

KURITA & TAMURA (IH SHH)

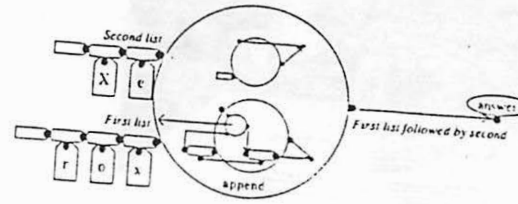


Figure 1: A Simple Example Program to Append Lists

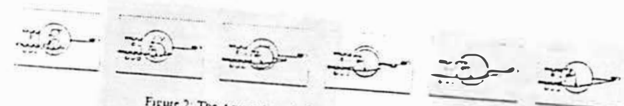


Figure 2: The Animation of a Sucessful Rule Match

Figure 3: The Animation of a Rule Commitment

Figure 4: The Animation of Links Shrinking and Agents Rescaling

3

KAHN & SARASWAT

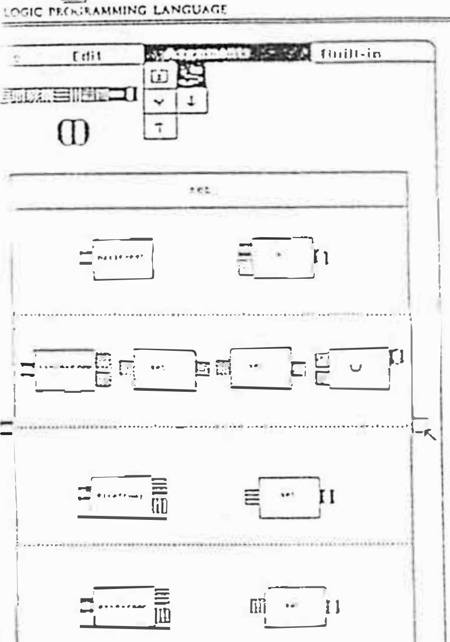


Figure 23(b). Using the standard pattern for defining arguments

corresponds to the following Prolog clauses.

```
set (Ev, [Ev]):-
    basic-evt (Ev).
set (Ev, S):-
    and-decomp (Ev, Sev1, Sev2),
    set (Sev1, S1),
    set (Sev2, S2),
    union (S1, S2, S).
set (Ev, S):-
    or-decomp (Ev, Sev1, Sev2),
    set (Sev1, S).
set (Ev, S):-
    or-decomp (Ev, Sev1, Sev2)
    set (Sev2, S).
```

CAPRET & KOEHLER 1991

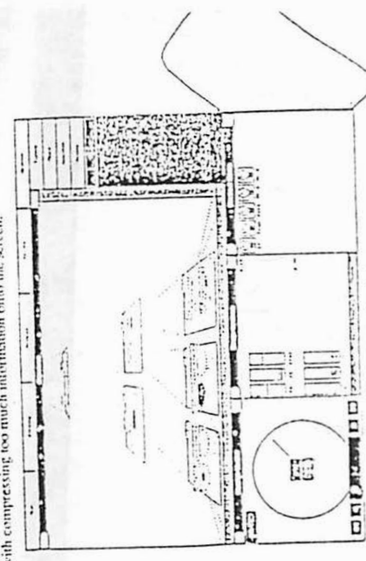


Figure 7 The Cockpit

If the hierarchy is extensive, the scrolling facility helps to avoid some of problems associated with compressing too much information onto the screen.

Objectives, Constraints and Metrics

The value of the objective function (or, at lower levels, of constituent objectives) is not the only information encoded in the hierarchy of sheets. It was mentioned earlier that design requirements can be specified in three ways: as (1) hard constraints, (2) soft objectives

General motivations for Visual Programming(Myers, 91)

- Human visual information processing optimised for multi-dimensional data (Myers,91)
- Flowcharts & indenting known to help (Smith,77)
- 2 D displays of data structures in program visualisation systems known to be helpful (Backer, Myers)
- Higher levels of abstraction can often be shown easily
- Can represent relationships that are hard to verbalise
- Can show multiple relationships concisely and clearly without cognitive overloading
- Structures can be easier to remember. Shu (1988)
- Clarisse (86) : graphical reps can be -
  - nearer to presumed mental reps of problem
  - manips nearer to those performed on phys objs
  - easier to understand & generate for nonprogrammers or programmer novices
- Catalogue of psychological motivations (Smith 1977)
- Use perceptual processing to free up scarce cognitive resources to deal with higher level problems. (Xerox PARC, Information Visualiser)

Origins

Inspired by

- Steele's (1980) notation for constraint programming (electronics DIP metaphor)
- Design of a graphic programming language for beginners for a domain specific constraint-based planner (music) (Holland, 1989)
- Generalised to Prolog 1990
- 3D Execution model devised 1990 but dropped on grounds of impracticability until saw reports of Xerox information visualiser
- Picture machine devised 1990
- Two implementations 1991
- Refined execution model with reference to TPM (Eisenstadt & Brayshaw) 1991 - otherwise developed in ignorance of related work

LIMITATIONS AND WEAKNESSES OF VPP

Current implementations very limited

Following origins, so far optimised for domain specific programming kits & for non-prolog programmers
cf Labview
Max
Melody Machine
Picture machine

Parts of design still being refined - e.g. large scale views, list processing, etc

Studies of users required

Hypotheses

- NB - all open to experimental test
- General in certain cases
  - unloads tasks from cog facs to percept facs,
  - provides easily grouped 'visual caches'
  - exploits gestault percept skills in lieu of problem solving skills
- Integrated formalism :
  - less to learn
  - less cog load in matching source + exec trace
- Factory metaphor
  - makes sense/ usable for novices with no Prolog at all
  - clear story for pure/impure features
- 3D execution model of certain sorts
  - more distinctions clear at a glance using low level percept skills
  - exploits strengths of likely new wave of GUIs

CONCLUSIONS
VPP and VPE

Unique features of VPP + VPE
- integrated system: one formalism
  complete overview
- factory metaphor: non-progs?
- exec model uses 3 spatial dims

Hypotheses

- General
  - unloads tasks from cog facs to percept facs,
  - provides easily grouped 'visual caches'
  - exploits gestault percept skills in lieu of problem solving skills
- Integrated
  - less to learn
  - less cog load in matching source exec trace
- Factory metaphor
  - makes sense/ usable for novices with no Prolog
  - clear story for pure/impure features
- 3D execution model
  - more distinctions clear at a glance using low level percept skills
  - exploits strengths of likely new wave of GUIs

Experimentation & more refined implementation required

FURTHER WORK

- More refined implement & design VPP
- Implement VPE (Holland, Treglown)
  - Instantiation flows
  - Selective views, prune,zoom, 3D rotation
  - Long distance views view

- Various extensions or VPE have been designed which in principle could make it as fully-featured a debugger as TPM, although that is not its primary purpose.

- Experiment with symbology for VPE
  - animation vs notation :
    - "?" marks vs flashing etc
  - lists

- Formative evaluation: experiments with users