Interim Report on

Visual Programming & Visualisation of Program Execution in Prolog

Simon Holland
simon@uk.ac.abdn.csd

(Presentation at PPIG 1992)

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

OVERVIEW

- Intro - what do we mean by
  - visual programming (VPP)
  - vis of program execution (VPE)

- Unique features of VPP + VPE
  - integrated system: one formalism
  - 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

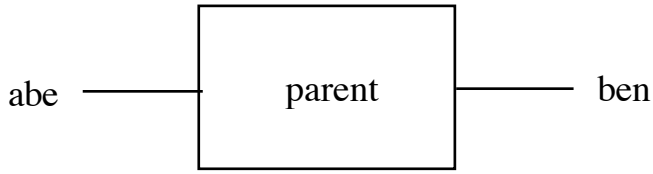
- Hypotheses about VPE and VPE
- Limitations & further work

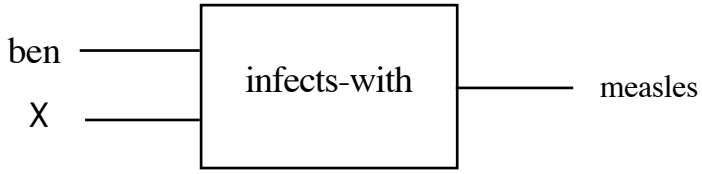
- Summary & Conclusions

VISUAL PROGRAMMING IN PROLOG

Facts in a database in VPP

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





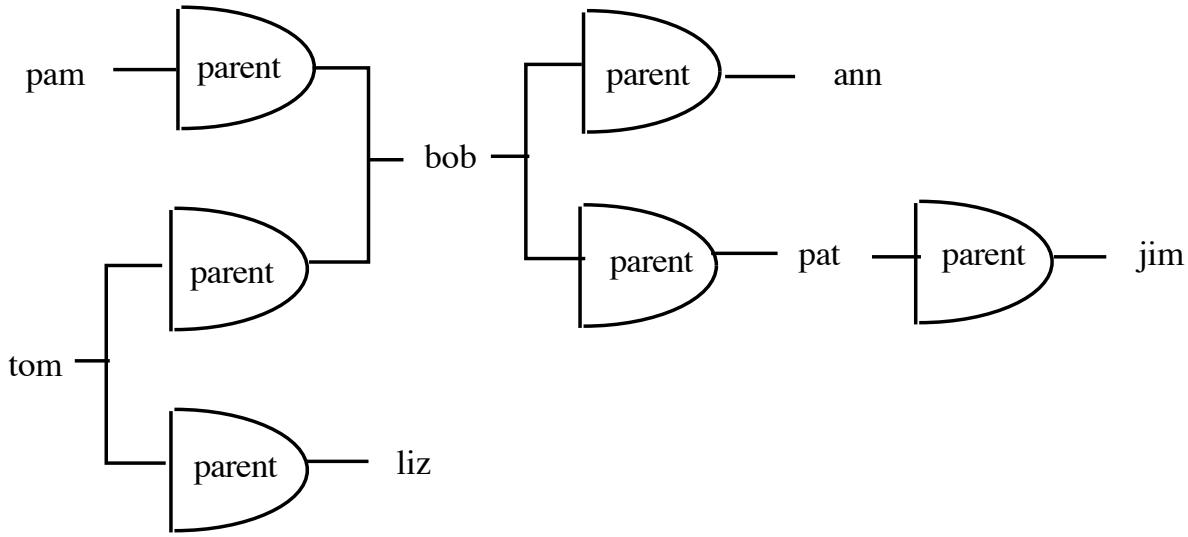
- Constants and variables - links
- Relations - boxes

- 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

Clauses with shared constants in database

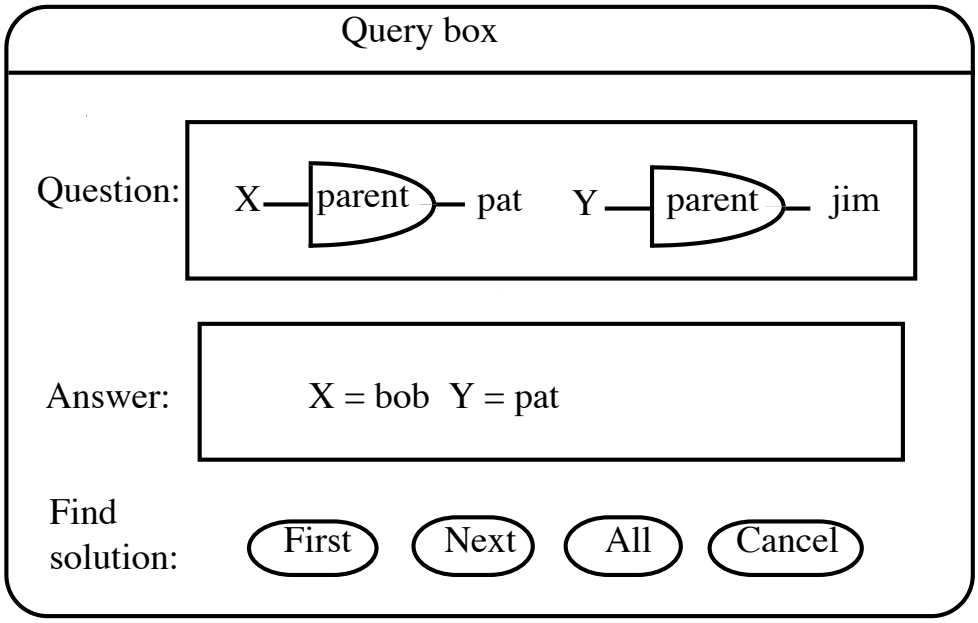


parent(pam,bob).
parent(bob, ann).
parent(tom,bob).
parent(bob, pat).
parent(pat, jim).
parent(tom, liz).

- "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)

Queries



Simple queries
parent(Y,jim?)

Conjunctive queries
Put more than one clause in query box

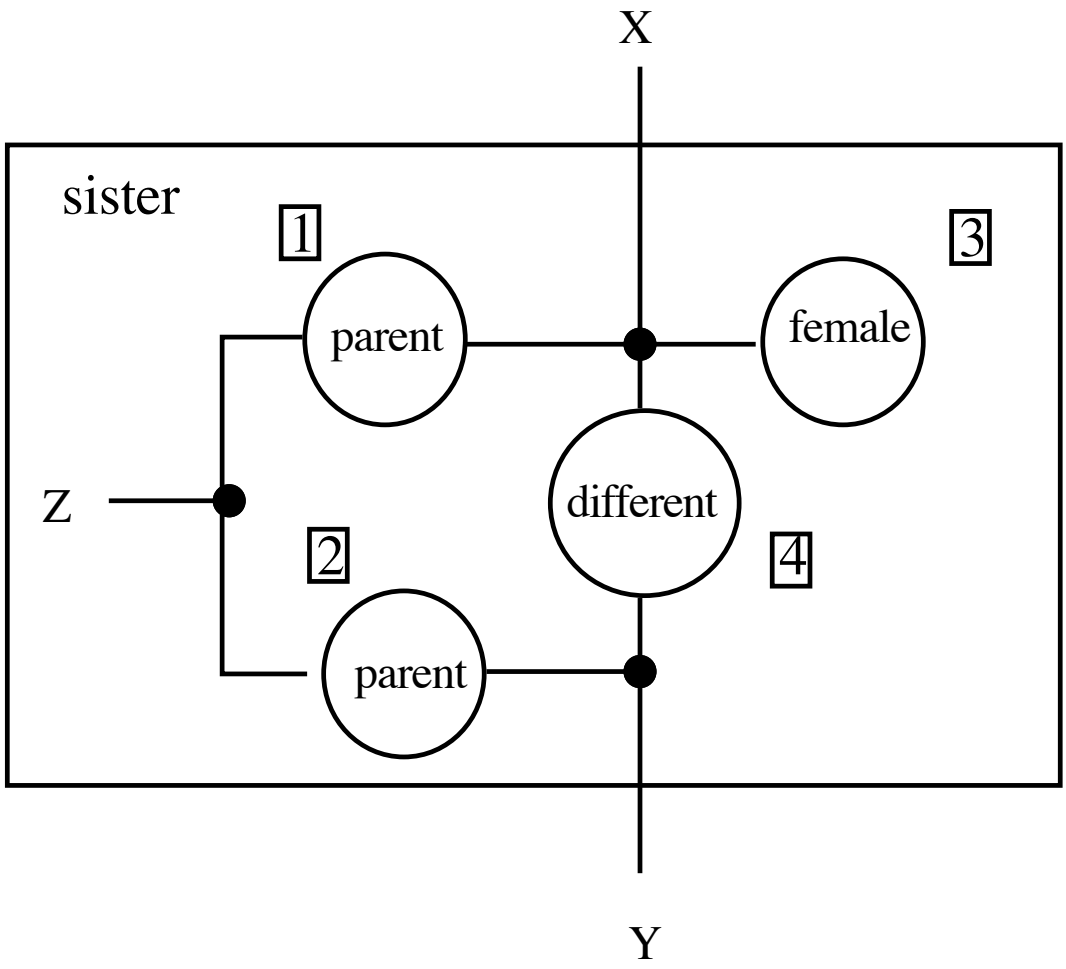
Conj queries with shared vars
Allowed

Disjunctive queries
Pose the disjunctive clauses as queries one after another.

<make diag common - show ans as visula - sev views of answer>

Rules

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



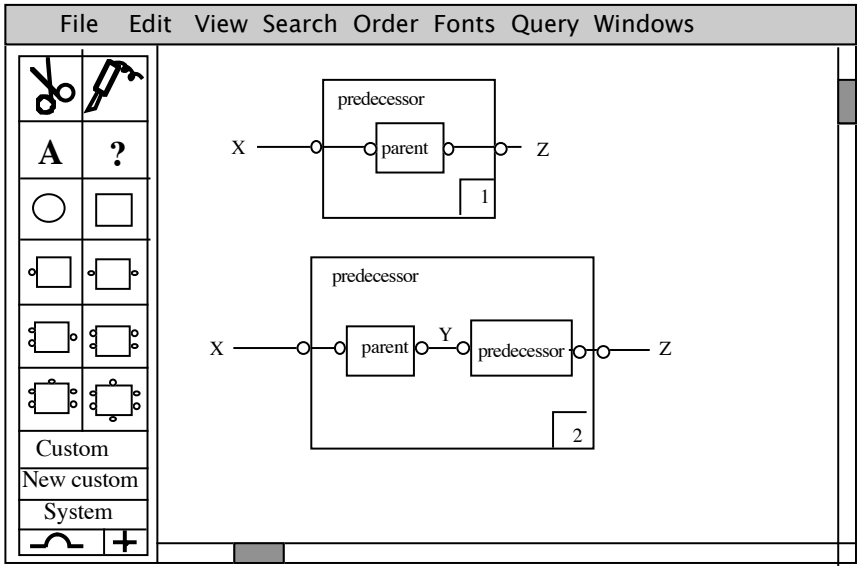
• Within a rule, optional variable & constant sharing (e.g X,Y,Z above)

• 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

Programming using VPP



Current Prototype (slightly idealized)
Docks within box
Schematic overview

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

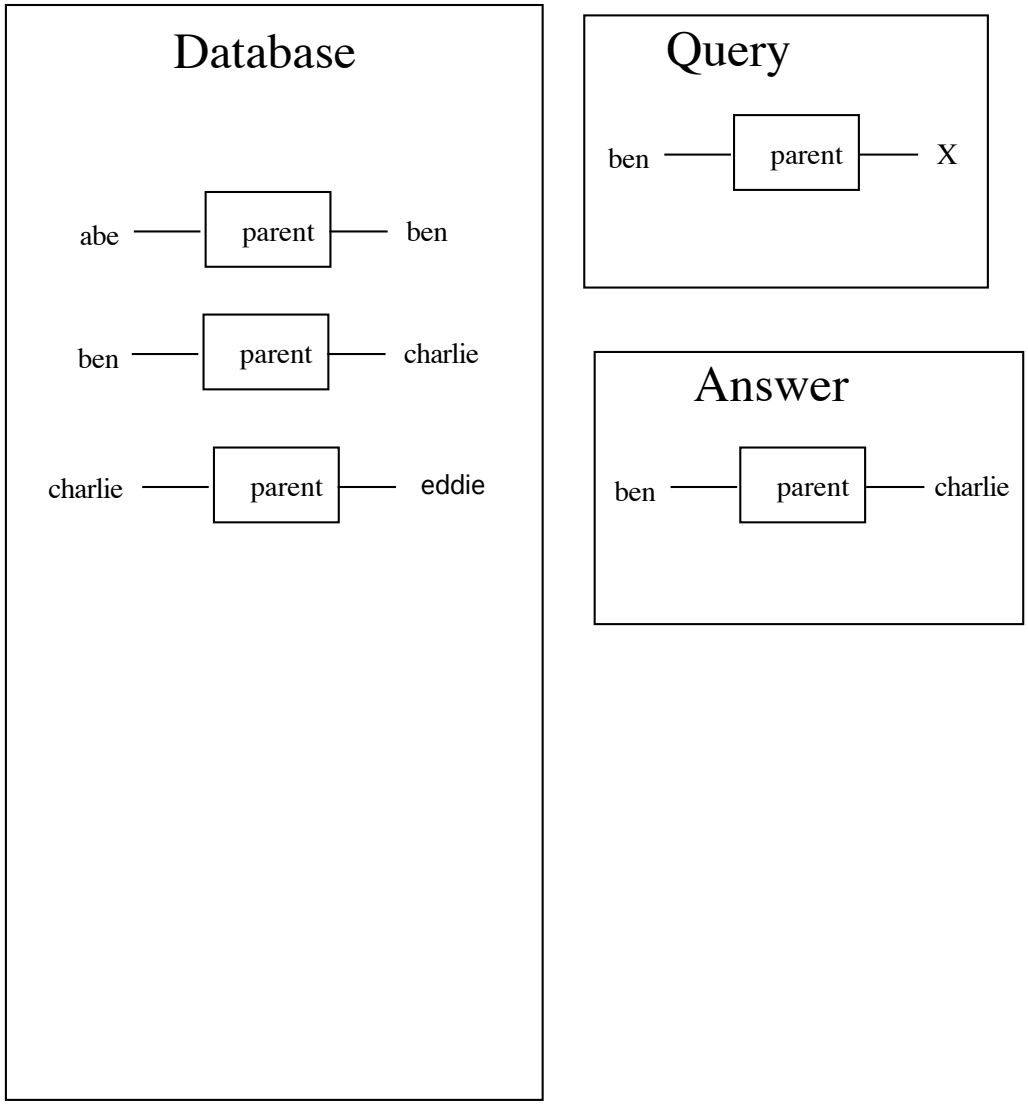
- boxes types to choose from
- typing tool to name boxes and variables

- 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


Database
abe
parent
ben
ben
parent
charlie
charlie
parent
eddie
Query
ben
parent
X
Answer
ben
parent
charlie

BASIC IDEA BEHIND VISUALISING PROLOG EXECUTION (VPE)

Simple view with details of unification, search & alternatives not shown

Simple recursive procedure

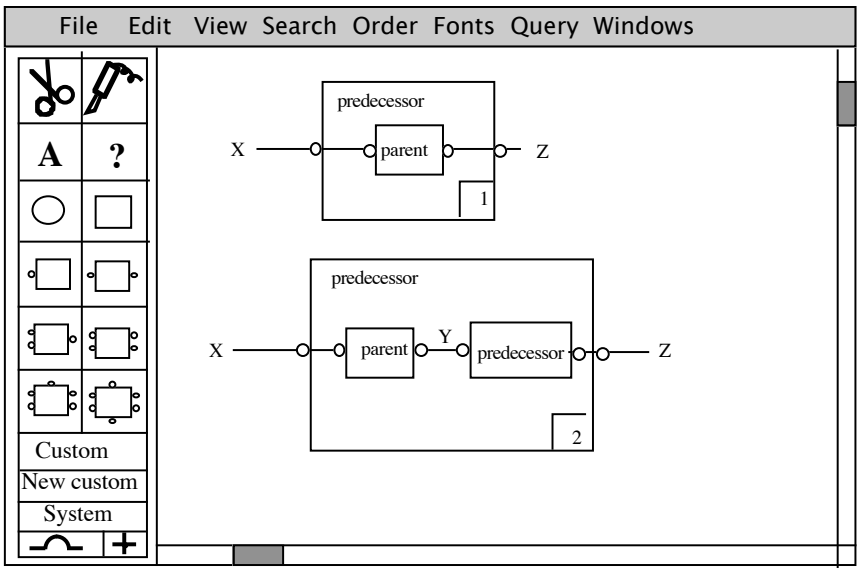
```
predecessor(X,Z):-
        parent(X,Z).

predecessor(X,Z):-
        parent(X,Y),
        predecessor (Y, Z).
```

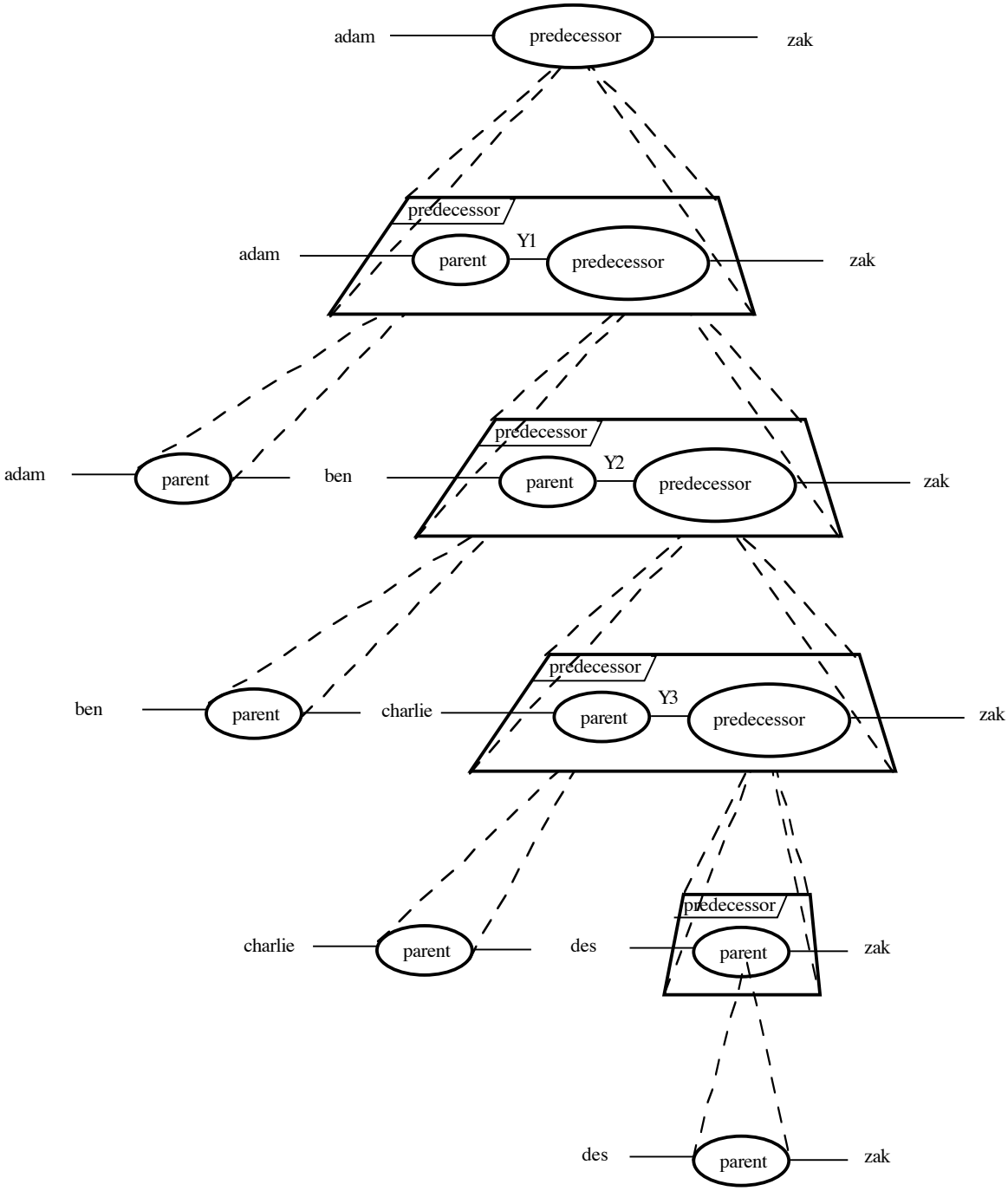
Database of facts

```
parent(adam, ben).
parent(ben, charlie).
parent(charlie, zak).
```

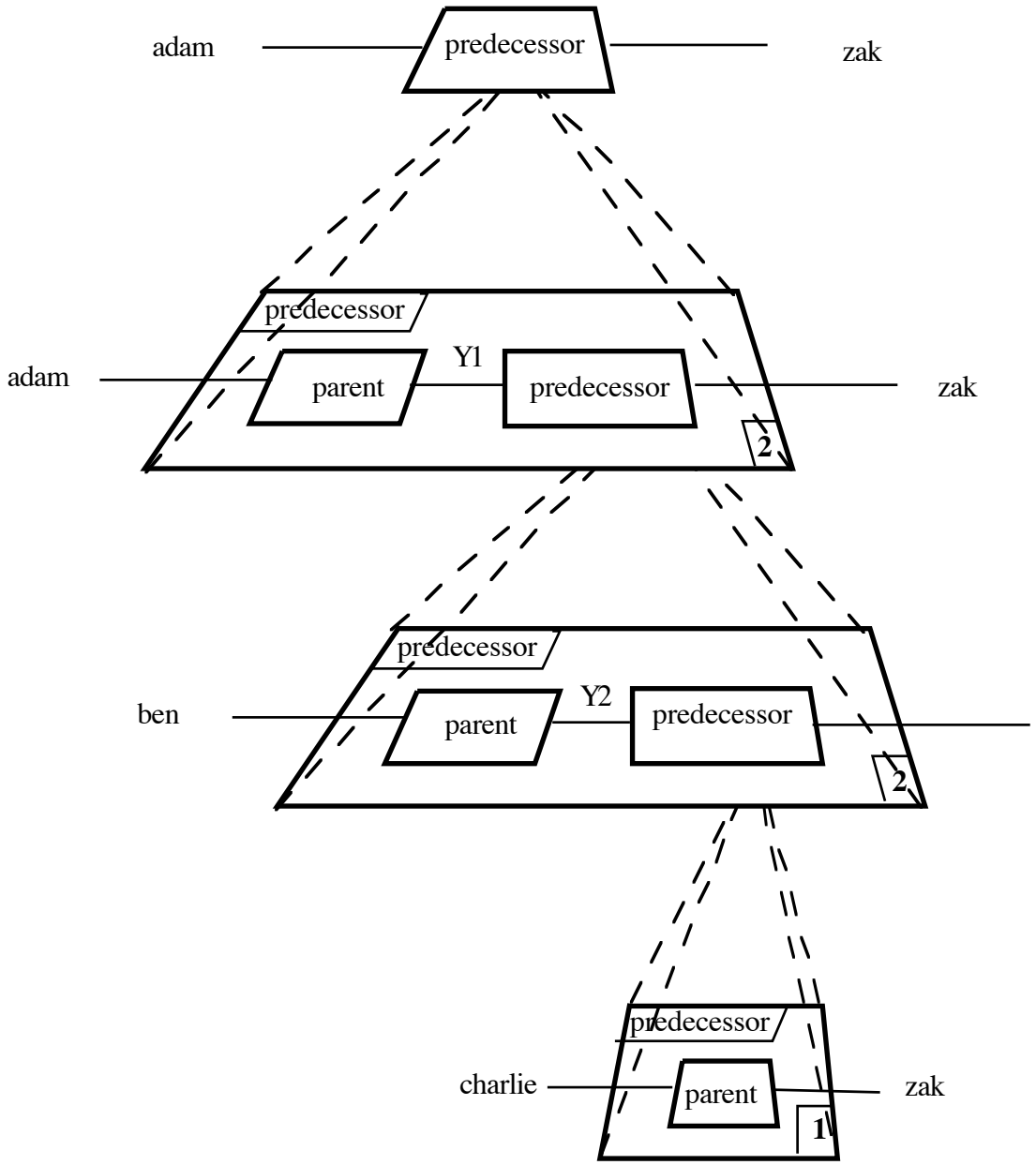
Query:

predecessor(adam,zak)?

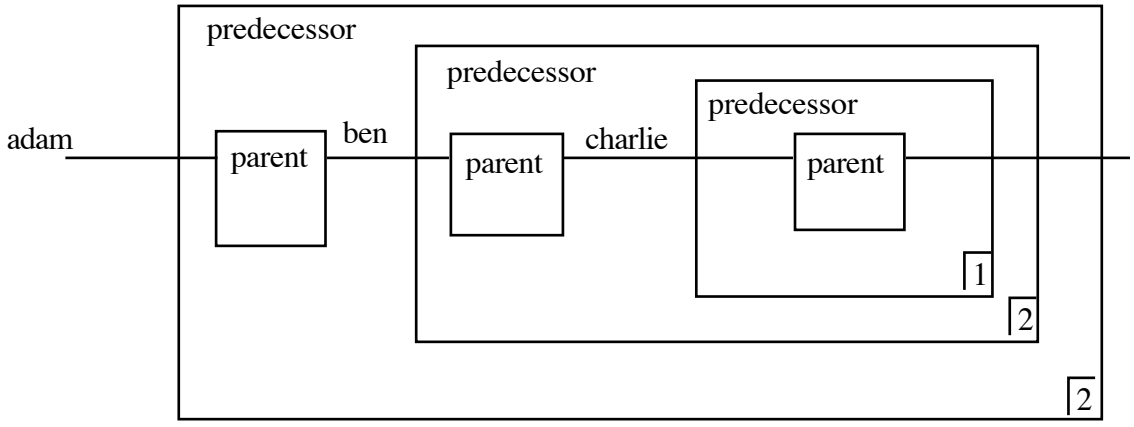

File Edit View Search Order Fonts Query Windows
A
?
Custom
New custom
System
predecessor
X
parent
Z
1
predecessor
X
parent
Y
predecessor
Z
2



where is dtabase graphically? cane common terms?



maybe not show variables
cane all this??



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)?

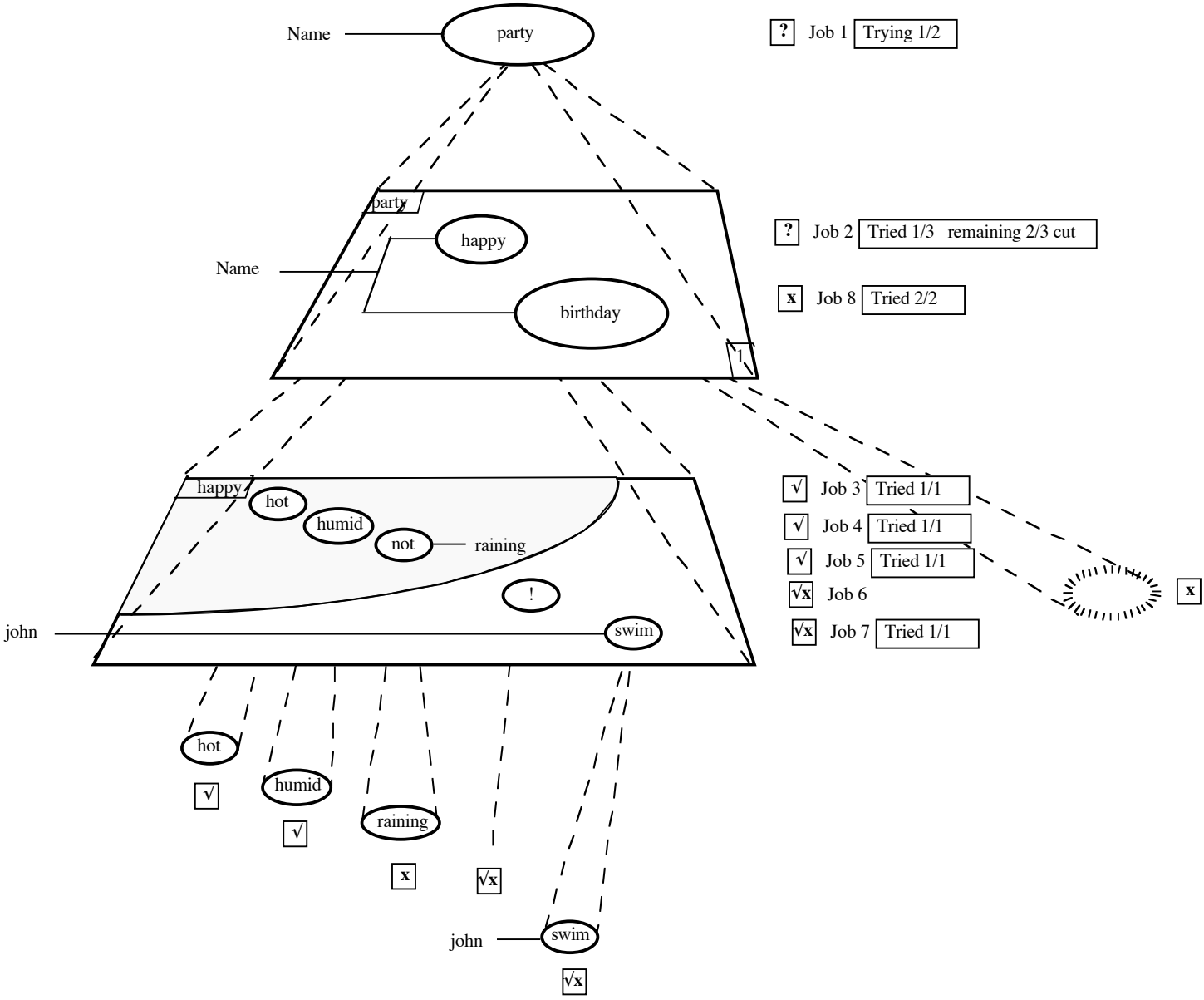


Figure 11. A snapshot of the trace in VPE of the program shown in figure 10 given the query *party(Name)?* up until backtracking begins with the failure of birthday/1.

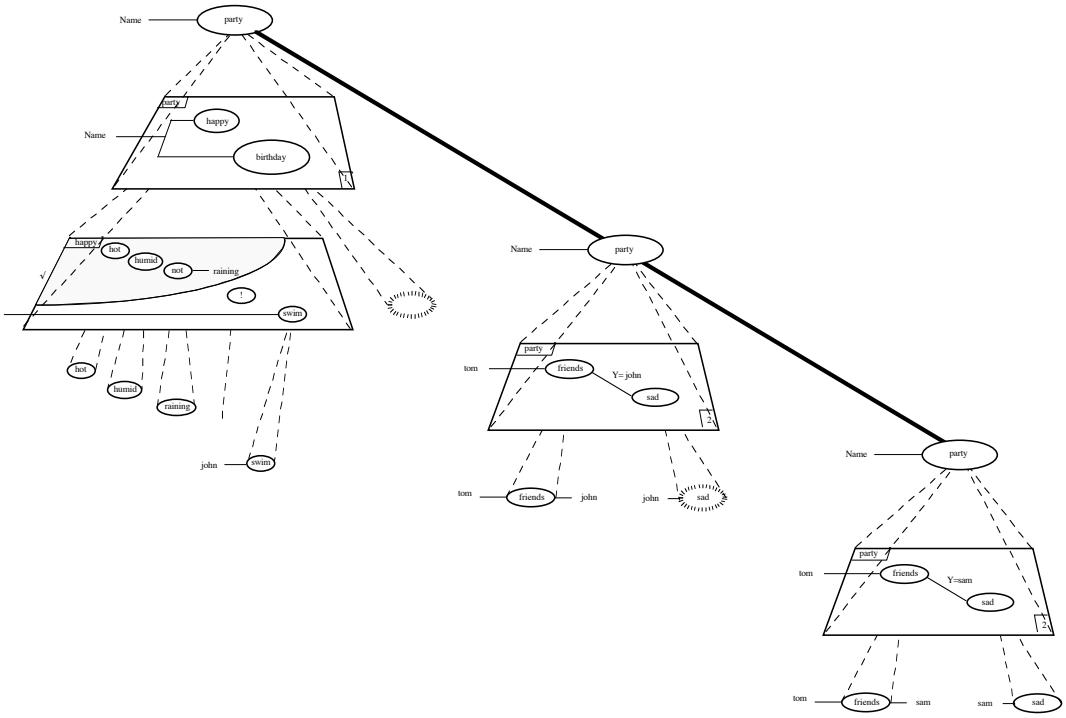


Figure 12. An outline trace in VPE of the complete execution space of the program shown in figure 10 given the query *party(Name)?*

POSSIBLE BENEFITS OF VPP and VPE

- In cases where non-programmers need to understand or modify simple Prolog code, but may not be interested in learning Prolog, VPP may score well.

- The "factory construction" metaphor outlined in Holland (1991) which gives a homely rationale for the execution of a Prolog interpreter may be useful in this context

- In databases and rules with not overly complex shared terms, VPP can allow inter-relationships to be noted rapidly without having to memorise variable names, scan for matches, so lessening load on short term memory.

- VPE may be useful for inspecting complex relationships broken down into their sub relationships (i.e. proof trees where branches that proved resulted in failure are ignored). Visual 3D presentation of this kind of information may be useful cf Info Visualiser

- VPP makes it quite clear that predicates and structures are different kinds of objects. This may help to avoid some misconceptions .

- VPP can help to elucidate the tree-like nature of structures by showing their form graphically.

- VPP can on occasion help to clarify aspects of Prolog even to slightly more experienced programmers. For example, it reveals graphically that the predicate 'not' could not take a predicate as an argument, but must take a term or structure (figure 13). This might help to avoid persistent misconceptions about "not" and the absence of second order behaviour in Prolog in general.

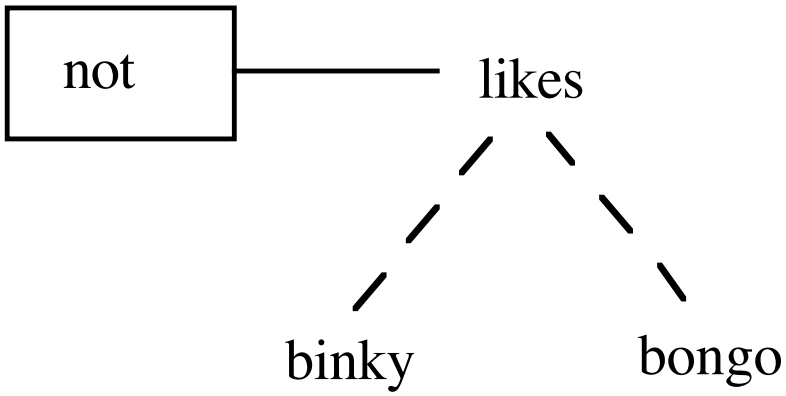


Figure 13. Representation of not(likes(binky,bongo).

LIMITATIONS AND WEAKNESSES OF VPP

Current implementation limited

Parts of design need refining

Shared term idea not always helpful

Not aimed at general purpose programming for professional programmers.

May help modest Prolog beginners

Aimed primarily at *non-programmers* using Prolog for partic applications e.g. ITS, Music programming, Lab control

RELATED IDEAS AND SYSTEMS

Inspired by

- Steele's (1980) notation for constraint programming

- Attempts to design a graphic programming language for a constraint-based musical planner (Holland, 1980)

Most closely related existing system

- TPM (Eisenstadt & Brayshaw)

FURTHER WORK

- More refined implementation of VPP
  - Implement VPE (Holland, Treglown)
  - Instantiation flows
  - Selective views, prune,zoom, 3D rotation
  - Summary 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.

- Formative evaluation: experiments with users

CONCLUSIONS
VPP and VPE

•VPP (short for Visual Programming in Prolog).

• Create & edit Prolog programs graphically
• Queries can be constructed in a similar manner.

• VPP may make some aspects of programming easier for beginners & non-programmers
• VPP programs can sometimes be less clear than textual equivalent

Not aimed at general purpose programming for professional programmers.
May help some beginners
Aimed primarily at *non-programmers* using Prolog for application-specific graphic "construction kits".

• Extension of VPP dubbed VPE (Visualiser for Prolog Execution)

• Animated 3D model of program execution using same building blocks as VPP
• VPE could in principle be extended fully-expressive debugger

• Further development & study required
• Implementation
 • currently two prototypes of VPP
  • Philip
  • Treglown