

When Visual Programs are Harder to Read than Textual Programs

*T. R. G. Green** and *M. Petre*[#]

* MRC Applied Psychology Unit
15 Chaucer Road, Cambridge CB2 2EF, U.K.

[#] Institute for Educational Technology
Open University, Milton Keynes MK7 5AA, U.K.

In G. C. van der Veer, M. J. Tauber, S. Bagnarola and M. Antavolits (Eds.) *Human-Computer Interaction: Tasks and Organisation, Proc. ECCE-6 (6th European Conference on Cognitive Ergonomics)*. CUD: Rome, 1992.

Abstract

Claims for the virtues of visual programming languages have generally been strong, simple-minded statements that visual programs are inherently better than textual ones. They have paid scant attention to previous empirical literature showing difficulties in comprehending visual programs. This paper reports comparisons between the comprehensibility of textual and visual programs, drawing on the methods developed by Green (1977) for comparing detailed comprehensibility of conditional structures. The visual language studied was LabView, a circuit-diagram-like language which can express conditionals either as 'forwards' structures (condition implies action, with nesting) or as 'backwards' structures (action is governed by conditions, with boolean operators in place of nesting). Green (1977) found that forwards structures gave relatively better access to 'sequential' information, and Gilmore and Green (1984) showed 'backwards' structures gave relatively better access to 'circumstantial' information. These differences were supported in the present study for both text and graphics presentations. Overall, however, the visual programs were harder to comprehend than the textual ones, a strong effect which was found for every single subject, even though the subjects were either experienced LabView users or else experienced users of circuit diagrams. Our explanation is that the structure of the graphics in the visual programs is, paradoxically, harder to scan than in the text version.

1. Introduction

Large numbers of visual programming languages (VPLs) have been invented. Their proponents claim that these languages are easier to understand than textual languages (TLs). Sometimes the claim is based on ill-digested pseudo-psychology about TLs 'not utilizing the full power of the brain' (Myers, 1990, p.100); sometimes it rests on the unfortunately-named so-called 'software science', extended now into the visual domain by Glinert (1990) in an otherwise excellent paper. At other times VPLs are described as 'de-emphasizing issues of syntax and providing a higher level of abstraction' (Myers, 1990, p. 100), a more cautious claim. Comments from professional electronics designers tend to support this last view (Petre and Green, 1992), although the electronics domain is rather different from programming.

Improved legibility of VPLs is an important issue, not only because TLs are notoriously hard to read, but also because of the potential for making use of increased legibility to enable 'display-based problem solving' (Larkin, 1988; Howes and Payne, 1990). Green, Bellamy, and Parker (1987) proposed a cyclical coding model in which programs were developed a chunk at a time -- a chunk possibly but not inevitably corresponding to a 'programming plan' (Rist, 1986; Davies, 1990) -- and in which, as each chunk was prepared mentally and became ready to insert into the material that had been written so far, the programmer had to re-establish the intentions and structure of the existing material. This 'parsing-gnirap' model was supported by detailed observations (Green et al., 1987; Davies, 1989). According to such a model, increasing the comprehensibility will certainly facilitate the programming process. As Fischer et al. (1991) have shown, such 'talking back' from the environment is also typical of interaction with very complex environments

as well as the much simpler ones studied in the laboratory. Although this may seem a glimpse of the obvious, programming languages seem to have been mainly devised for ease of generation rather than for comprehension (in so far as cognitive aspects have been considered at all).

Unfortunately, previous research on comprehensibility of VPLs has been unencouraging (Brooke and Duncan, 1980; Curtis et al., 1989). Nevertheless the developers of VPLs clearly believe in their products: could previous research be wrong or inappropriate? Perhaps so; today's VPLs are usually dataflow languages rather than the flowchart-like languages studied by Curtis et al and others; moreover, today there are commercial languages with real users, in contrast to the subjects of previous studies, who had used TLs for programming and visual notations for documentation; and finally, the dataflow model has allowed some languages to be based on familiar metaphors. Typical among these is the LabView language (Santori, 1990; Hils, 1992), the language that we used in the studies reported here. LabView is closely based on the metaphor of electronic block wiring diagrams.

In this paper we therefore examine the claims that VPLs are highly comprehensible, taking LabView as our type of the VPL. The paper builds on a preliminary paper by Green, Petre and Bellamy (1991) presenting outline data from a sample of LabView users; for various reasons, that paper presents only outline results. The present paper (a) reports results from a further sample of electronics designers, thoroughly familiar with the underlying metaphor of LabView, (b) presents full analyses, (c) relates the findings to the 'match-mismatch' hypothesis (Gilmore and Green, 1984) and the 'cognitive fit' hypothesis (Vessey, 1991), (d) presents a simple model of information-gathering from VPLs and TLs which is sufficient to account for the results, (e) relates that model to the models of graphical information-gathering by Lohse (1991) and of display-based problem-solving by Larkin and Simon (1987), and (f) concludes that future claims of advantages for notation structures and information displays should be more closely related to models of the cognitive processes of information extraction.

2. The Background

Previous work on the design of programming languages has frequently made the claim that no particular notation is universally best; rather, each notational structure highlights some kinds of information at the expense of obscuring other types. For conditional program structures within a procedural paradigm Green (1977) distinguished between 'circumstantial' (or 'taxon') and 'sequence' information, showing that nested conditionals favoured sequence information, ("Given this input, what happens?"), a claim later strongly supported by the work of Curtis et al. Conversely, Gilmore and Green (1984) showed that a more declarative programming language gave improved access to circumstantial information ("Given this result, what do we know about the input?"). Gilmore and Green postulated that across the spectrum of information structures, performance was at its best when the structure of the information sought matched the structure of the notation, and that a mismatch would lead to poor performance (the 'match-mismatch' hypothesis).

In a parallel but independent line of work, Vessey (1991) investigated the use of tables and graphs for conveying information and for problem-solving. She distinguished between the external problem representation (in our terms, that would be the available data) and the representation of the problem solving task (in our terms, that would be the problem to be solved). When the types of information emphasized in these two representations match, she asserts, "the problem solver uses processes (and therefore formulates a mental representation) that also emphasize the same type of information. Consequently, the processes the problem solver uses to both act on the representation and to complete the task will match, and the problem-solving process will be facilitated." (p. 221).

3. The Languages

Green and Gilmore used simple TLs based (at some distance) on existing full-scale languages. One of these, Nest-INE, was a sequential notation using nested conditionals with the assistance of extra cues, which had earlier been shown to assist both novices and professionals (Sime, Green and Guest 1977: Green 1977). Their other notation, which we shall refer to as And/Or, was based on a production system model, in which the structure was clearly circumstantial. Both of these languages were employed in the present experiment (Figures 1 and 2). The prevailing characteristic is that Nest-INE supports working *forwards*, from the input to the output, whereas And/Or supports working *backwards*, from the output to the input.

For the corresponding VPLs, it so happens that the LabView language is capable of expressing conditionals either as a sequential structure or as a circumstantial structure. The 'Boxes' notation (Figure 3) is unusual in being interactive -- at any one moment, the display only shows one arm of an if-then-else conditional, and the reader has to click on the True/False button to toggle the two arms of the conditional. The result is a sequential structure: given the selector condition, the appropriate part of the case structure can be quickly found. In contrast, given the output, the reader has to search through a variety of cases to find the appropriate case that will generate such output -- although, once found, reading off the input conditions is relatively easy.

The 'Gates' notation, also available in LabView (Figure 4), is seemingly a circumstantial structure. Given an output, it is relatively straightforward to work backwards through to the input side, recording the conditions as one goes; but working forwards from input to output is much harder, requiring the reader to branch backwards at each AND-gate.

4. The hypotheses

Both Brooke and Duncan (1980) and Curtis et al. (1989) found that the advantages of flowcharts over text, if any, were at the detailed level rather than at the overview level. We therefore chose to investigate detailed comprehension of conditionals. In part 1 we investigated question answering, following the lines adopted by Green (1977) and Curtis et al. (1989), in this new context of dataflow VPLs with experienced users. The question-answering task corresponds to "What does this program do?" (forwards) or "What made it do that?" (backwards). In Part 2 we investigated same-different comparisons between programs, which have not previously been studied. When both programs are in the same notation, same-different judgements correspond to such real-world questions as "How do these programs differ?"; when they are in different notations, the real-world equivalent might be "Does this program agree with this specification?"

If the supporters of VPLs are correct, then visual representations will be uniformly superior. The contrary hypothesis is that in Part 1, sequential languages will facilitate forwards questions, and that circumstantial languages will facilitate backwards questions, with no strong assertion about the difference between the graphical and textual modes. In Part 2, our hypothesis is that a mismatch between program structures (i.e. one forwards, and one backwards notation) would slow up performance.

5. Procedure

Examples of stimulus programs are shown in Figures 1-4, covering Text or Graphics crossed with Sequential or Circumstantial. The two graphical notations were derived from LabView, using exact screen images copied into SuperCard. For the Boxes notation (Figure 3), the interactive aspects of LabView were emulated in SuperCard. In half the programs there was at least one outcome that could be reached by more than one route: these are 'multi-path' stimuli. Other programs were 'single-path'.

In Part 1 of the experiment, each subject was shown a stimulus program and after a short interval was then shown either input data (for a *forwards* question) or an output result (for a *backwards* question). Responses were made by mousing on radio buttons. For a forwards question, a single mouse click was sufficient to state the action of the program. Backwards questions for single-path programs were answered by setting 6 radio buttons to appropriate values, True, False, or Irrelevant; for multiple-path programs, 12 radio buttons were set. Examples of these tasks are shown in Figure 5.

In Part 2 of the study two programs were presented side by side, and the subject responded either Same or Different, again by mousing a button. Due to poor design, the subjects in Group 1 (see below) only received comparisons in which one notation was a TL and the other a VPL. The mistake was repaired for Group 2, and all possible comparisons were made.

Two groups of subjects were recruited. Group 1 (N=5) were occasional programmers who had used LabView for their work or at least 6 months. Overall programming experience ranged from 5 to 15 years and covered a variety of languages, Basic and assembly language being the commonest. Group 2 (N=6) were very experienced programmers and designers of advanced digital electronics apparatus with no previous experience of LabView.

6. Results

6.1 Part 1: Forwards and Backwards questions

Errors were few and unsystematic. *Time scores* were transformed to logarithmic units and analysed by ANOVA with three repeated-measures factors: Modality (text vs. graphics), Structure (sequential vs. circumstantial) and Direction of questioning (forwards vs. backwards single vs. backwards multiple). Since the procedure for Part 1 was identical in the two versions of the experiment, but the subject populations differed, we treated the combined results as having an additional between-groups factor, Group (Group 1 vs. Group 2). The ANOVA revealed no interactions involving the Group factor with a significance level below 10% except Direction*Structure*Group, which reached 0.08. There was not even a main effect for Group (i.e. neither group of subjects responded significantly faster overall). It appears likely, therefore, that the results we report will hold for a wide range of subject populations.

The overall pattern of results is illustrated in Figure 6, expressed in the logarithmic units used for statistical analysis. *Match-mismatch*: Forwards questions were answered faster in notations with Sequential structure, Backwards questions were answered faster in notations with Circumstantial structure, as predicted by the 'match-mismatch' or 'cognitive fit' hypothesis. This interaction, Direction*Structure, was highly significant ($F(2, 18) = 21.77, p = 0.0001$). *Graphics v. Text*: Overall, graphics was slower than text. The main effect for Mode was very highly significant ($F(1,9) = 208.6, p < 0.0001$). Moreover, graphics was slower than text *in all conditions*: there were no high-order interaction effects, and in particular Mode*Direction*Structure was far from significance ($F(2,18) = 1.43, p = 0.64$). The overall geometric mean for Text responses was 35.2 seconds, as against 68.1 seconds for Graphics responses. *Size of effect*: Hayes (1981) gives a procedure for estimating ω^2 , the size of an effect in terms of total variance accounted for. The three largest effect sizes were Mode, Direction, and Direction*Structure, for which ω^2 was respectively 25.0 %, 24.7 %, and 5.7%; so about 56% of the total variance was accounted for by these three. *Uniformity of effect*: For each individual subject, we compared the 8 times for text notations to the 8 times for graphics notations. The mean time for graphics conditions was greater than the mean time for text for every single subject.

6.2 Part 2: Same-Different Judgements

Two analyses were made, one of all the responses, the other only of correct responses. Because each comparison employed two notations, the response time was affected not only by the intrinsic difficulty of each notation but also by the individual combination of notations. We were particularly interested in testing for a *match/mismatch effect*. Contrast analyses were used to test the hypotheses that judgements are faster (or slower) when the modalities differ (i.e. one Graphics, one Text), or else when the structures differ (i.e. one Sequential, one Circumstantial). Neither was significant.

We also tested for an *intrinsic difficulty effect*, using linear contrasts for number of Graphics notations presented on each trial (0, 1, or 2) and number of Sequential notations (also 0, 1, or 2). Although these tests had lower power, both contrasts were significant. For Mode, $F(1, 10) = 43.9, p < 0.0001$, showing that the more Graphics components, the longer the response time (Figure 7); for Structure, $F(1,10) = 5.6, p = 0.040$, showing a weaker effect that the more Circumstantial components, the longer the response time (Figure 8). It would appear that intrinsic effects of Mode and Structure were much stronger than match/mismatch effects, and that the intrinsic difficulty of the graphics mode was the strongest effect of all. Error data were not revealing.

7. Discussion

7.1 Visual languages versus Text languages

Our first conclusion obviously concerns the plausibility of dataflow VPLs. Given these results, it can hardly be claimed that VPLs are consistently superior to TLs! The data show that the graphical notations are in fact consistently *worse* than the textual notations. Our tasks were realistic tasks, testing the level of detailed comprehension that had previously been shown to be the best point of flowchart-based VPLs. The language is reasonably successful commercially. The size of the problems we used was not unrealistic (the LabView manual itself gives examples with a comparable number of control components). Our subjects included two levels of programming experience, of familiarity with LabView, and of familiarity with the wiring-diagram metaphor. In view of all this, the poor showing of the LabView VPL shows that dataflow languages are poor at communicating this type of structure.

The problems of graphical representations showed most clearly in Part 2, the same-different comparisons, where the Gates structure stood out as hard to work with. The sight of subjects crawling over the screen with mouse or with

fingers, talking aloud to keep their working memory updated, was remarkable. This structure was very difficult for our subjects, *even for Group 2, who were very experienced with it*. As a support for reasoning processes it clearly leaves a great deal to be desired.

7.2 'Match-mismatch' / 'cognitive fit'

The results also show, as expected, support for the 'match-mismatch' or 'cognitive fit' hypotheses, thereby confirming the results obtained by Gilmore and Green (1984), Curtis et al. (1989), and Vessey (1991); but the effect of the match-mismatch is less than had been expected. Similarly Sinha and Vessey (1991), comparing Lisp and Pascal as vehicles for learning recursion and iteration, also found that 'cognitive fit' only told half the story, and that there were differences between programming languages that were not explicable in those terms. In the present experiment, the reason may be caused by the problems of tracing program behaviour and gathering information in the Graphics notations.

7.3 Information structures and models of information gathering

The profound difficulties of the Graphics modes in our study seemingly contradict recent developments in display-based problem solving (Payne and Howes, 1990; Larkin, 1988; Larkin and Simon, 1987) and sophisticated models of information-gathering in tables versus graphs (Lohse, 1991).

The reason is not far to seek. The models just cited have all obtained their results by showing that the graphical presentation of data made for improved information-gathering. For instance, Lohse's extremely thorough model predicts time required to answer questions about graphs and tables by allowing for eye-scan times, time to discriminate a symbol or a shape, time to match symbols against working memory, etc. The graphical presentations allow faster scanning and faster discriminations, so the model predicts faster total times. Similar, although less well-developed, arguments lie behind the Larkin and Simon account of how diagrams improve simple problem-solving.

In contrast, the information-gathering process in a VPL is sometimes extremely complex. We illustrate a plausible (but idealised) instance in Figure 9. It would not be difficult to cast this process into the form of a computational model, but the exercise is unnecessary to make our point: *not all graphical structures are equivalent*. Program structures contain 'knots' which force working memory load on the reader. Essentially, information gathering in the Gates notation is equivalent to traversing a maze. Interestingly enough, the text structures used in this study do not contain similar knots; they make a better use of spatial topography than the graphical notations! (Figure 10)

A further conclusion, therefore, is that the study of cognitive processes involved in understanding graphs and tables should not be limited to either the match-mismatch effect, or the faster speeds of scanning and processing graphical symbols. The information structure of the graph must also be considered. In cases like these where the graphical structure contains 'knots' but the textual version does not, the supposed advantages of graphics over text will prove illusory. Instead, performance will be dominated by working memory limitations.

References

- Brooke, J. B. and Duncan, K. D. (1980) Experimental studies of flowchart use at different stages of program debugging. *Ergonomics*, 23, 1057-1091.
- Curtis, B., Sheppard, S., Kruesi-Bailey, E., Bailey, J. and Boehm-Davis, D. (1989) Experimental evaluation of software documentation formats. *J. Systems and Software*, 9 (2), 167-207.
- Davies, S. P. (1989) Skill levels and strategic differences in plan comprehension and implementation in programming. In A. Sutcliffe and L. Macaulay (Eds.) *People and Computers V*. Cambridge University Press.
- Davies, S. P. (1990) The nature and development of programming plans. *Int. J. Man-Machine Studies* 32 461-481.
- Fischer, G., Lemke, A. C., McCall, R., and Morch, A. I. (1991) Making argumentation serve design. To appear in *Human-Computer Interaction*, 6 (3 & 4), 393-419.
- Gilmore, D. J. and Green, T. R. G. (1984) Comprehension and recall of miniature programs. *Int. J. Man-Machine Studies* 21, 31-48.
- Glinert, E.P. (1990). Nontextual programming environments. In Chang, S.-K. (Ed.) *Principles of Visual Programming Systems*. Prentice-Hall.

- Green, T. R. G. (1977) Conditional program statements and their comprehensibility to professional programmers. *J. Occupational Psychology*, 50, 93-109.
- Green, T. R. G., Bellamy, R. K. E. and Parker, J. M. (1987) Parsing-gnirap: a model of device use. In G. M. Olson, S. Sheppard and E. Soloway (Eds.) *Empirical Studies of Programmers: Second Workshop*. Ablex.
- Green, T. R. G., Petre, M. and Bellamy, R. K. E. (1991) Comprehensibility of visual and textual programs: a test of 'Superlativism' against the 'match-mismatch' conjecture. In J. Koenemann-Belliveau, T. Moher, and S. Robertson (Eds.), *Empirical Studies of Programmers: Fourth Workshop*. Norwood, NJ: Ablex. Pp. 121-146.
- Hils, D. D. (1992) Data flow visual programming languages. *J. Visual Languages and Computing*, in press.
- Howes, A. and Payne, S. J. (1990) Display-based competence: towards user models for menu-driven interfaces. *Int. J. Man-Machine Studies*, 33, 637-655.
- Larkin, J. H. (1988) Display-based problem solving. In D. Klahr and K. Kotovsky (Eds) *Complex Information Processing: the Impact of Herbert A. Simon*. Hillsdale, NJ: Erlbaum.
- Larkin, J. H. and Simon, H. A. (1987) Why a diagram is (sometimes) worth 10,000 words. *Cognitive Science*, 11, 65-100.
- Lohse, J. (1991) A cognitive model for the perception and understanding of graphs. *Proc. CHI 91 ACM Conf. on Human Factors in Computing Systems*, pp 137-144. New York: ACM
- Myers, B. (1990). Taxonomies of visual programming and program visualization. *J. Visual Languages and Computing*, 1, 97-123.
- Rist, R. S. (1986) Plans in programming: definition, demonstration, and development. In E. Soloway and S. Iyengar (Eds.), *Empirical studies of programmers*. Norwood, NJ: Ablex.
- Petre, M. and Green, T.R.G. (1992) Requirements of graphical notations for professional users: electronics CAD systems as a case study. *Le Travail Humain*, 55(1), 47-70
- Santori, M. (1990) An instrument that isn't really. *IEEE Spectrum*, August. Pp 36-39.
- Sime, M. E., Green, T. R. G., and Guest, D. J. (1977) Scope marking in computer conditionals – a psychological evaluation. *Int. J. Man-Machine Studies*, 9, 107-118.
- Sinha, A. and Vessey, I. (1991) Cognitive fit: an empirical study of recursion and iteration. Unpublished MS, Dept. of Accounting and MIS, Pennsylvania State Univ.
- Vessey, I. (1991) Cognitive fit: a theory-based analysis of the graphs versus tables literature. *Decision Sciences*, 22, 219-240.

Note

LabView is a trademark of National Instruments Inc. SuperCard is a trademark of Silicon Beach Software Inc.

We are very grateful to Rachel Bellamy, who helped set up the experiment, and to David Hendry, who improved the quality of this paper.

```
if high :
  if wide :
    if deep : weep
    not deep :
      if tall : weep
      not tall : cluck
      end tall
    end deep
  not wide :
    if long :
      if thick : gasp
      not thick : roar
      end thick
    not long :
      if thick : sigh
      not thick : gasp
      end thick
    end long
  end wide
not high :
  if tall : burp
  not tall : hiccup
  end tall
end high
```

Figure 1: an example of the Nest-INE notation (Text, Sequential).

```
howl :    if honest & tidy & (lazy | sluggish )
laugh :   if honest & tidy & ¬ lazy & ¬ sluggish
whisper : if honest & ¬ tidy & ( nasty & greedy | ¬ nasty & ¬ greedy )
bellow :  if honest & ¬ tidy & nasty & ¬ greedy
groan :   if honest & ¬ tidy & ¬ nasty & greedy
mutter :  if ¬ honest & sluggish
shout :   if ¬ honest & ¬ sluggish
```

Figure 2: an example of the And/Or notation (Text, Circumstantial).

The conditional structure shown is logically equivalent to the structure shown in Figure 1, with suitable change of labels (e.g. Bellow = Roar).

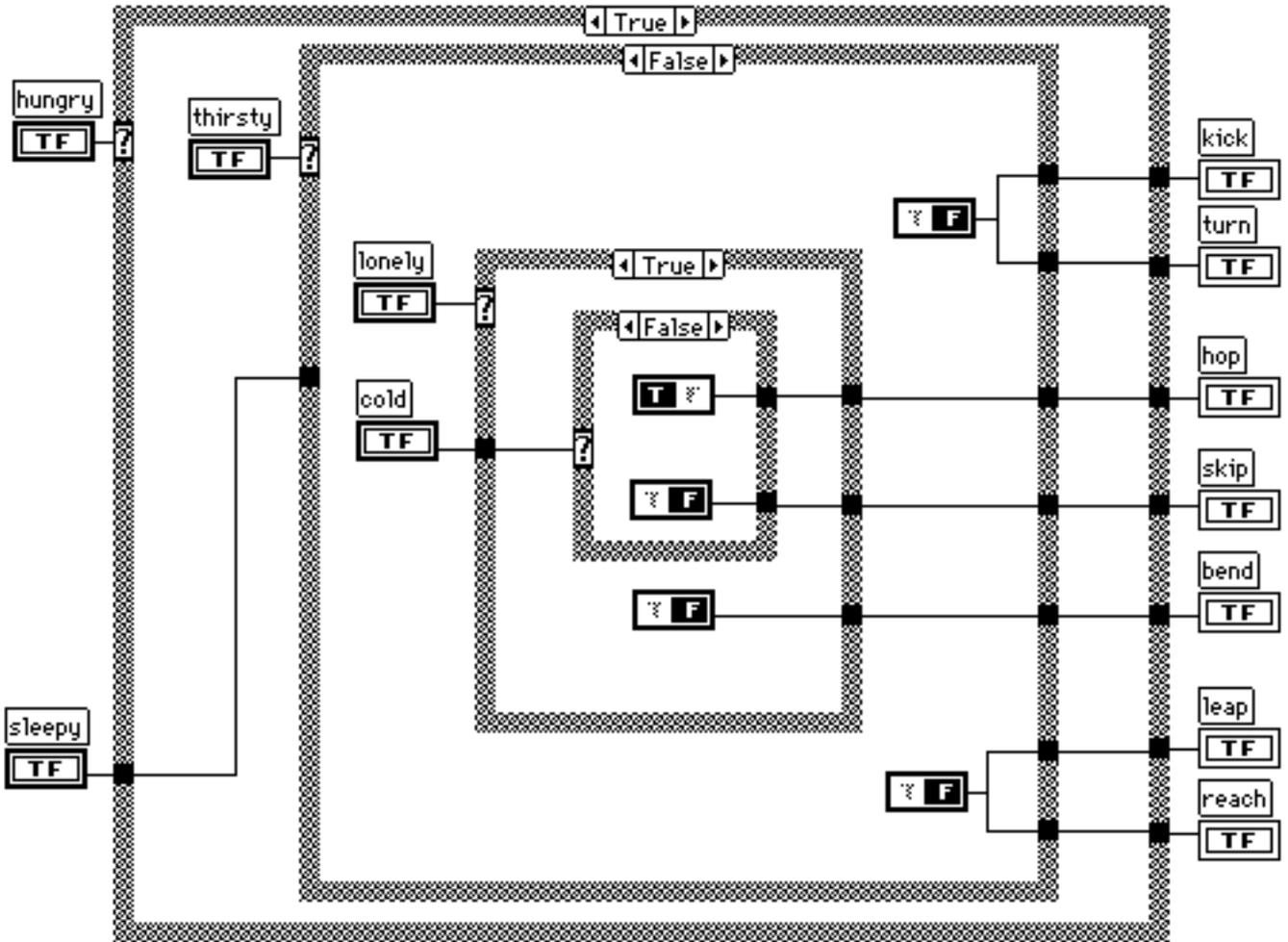


Figure 3: example of the Boxes notation (Graphical, Sequential).

This notation is interactive. Mouse clicks on the true/false buttons toggle between the true and false arms of the conditionals. Only one arm is visible at one time.

The figure shows the state of the display for Hungry = true, Thirsty = false, Lonely = true, Cold = false. The value of Sleepy is irrelevant in these conditions, although it is relevant in other circumstances (cf. Figure 1, where if High is false the values of Wide, Deep etc. are irrelevant). The output is shown by sending a boolean value to each possible output. In this figure, the output is Hop.

The conditional structure shown is both logically and structurally equivalent to the structure shown in Figure 1, with suitable change of labels (e.g. Hop = Roar).

The notation is exactly as used in LabView, although the interactive component was achieved by writing a SuperCard emulation of LabView's behaviour.

IS:
high
tall
thick

IS NOT:
wide
deep
long

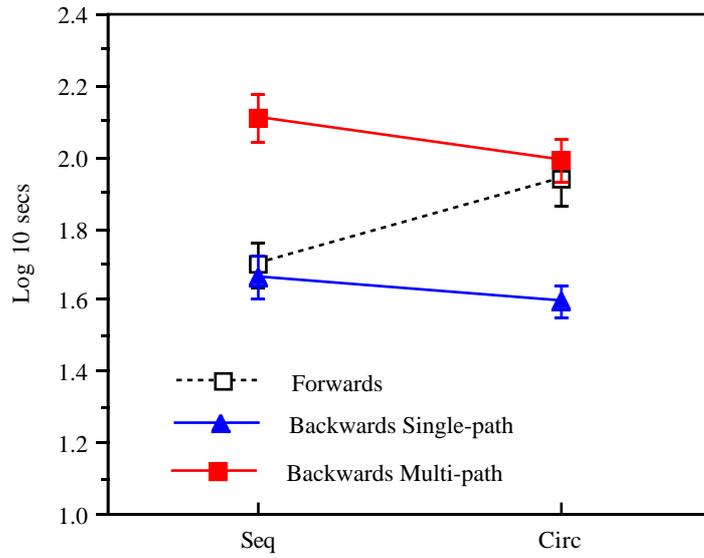
- weep
- cluck
- gasp
- roar
- sigh
- gasp
- burp
- hiccup

OUTCOME:
gasp

	TRUE	FALSE	IRREL	TRUE	FALSE	IRREL
high	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
wide	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
deep	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
tall	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
long	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
thick	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 5: Problem presentation for a forwards question (above) and a backwards question (below). The same presentation was used for each type of notation, appearing below the program on the screen. The circles represent Macintosh 'radio buttons', which were clicked with the mouse to turn them on and off. The backwards questions contained two sets of answer buttons (as shown here) for the multi-path programs, but only a single set for the single path programs. The buttons were initialised to 'IRREL'; this figure shows a correct answer to the question, given the program shown in Figure 1.

(a) The two Graphics notations (Boxes and Gates):



(b) The two Text notations (Nest-INE and And/Or):

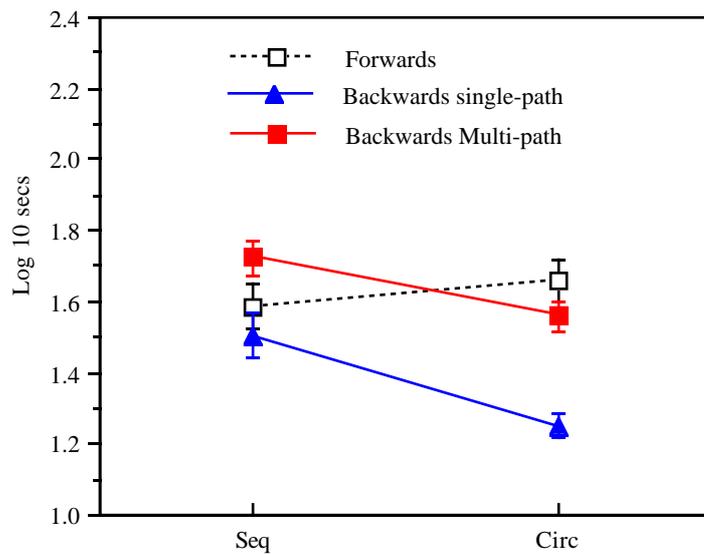


Figure 6: Response times in Part 1, pooled over both groups, showing the Direction * Structure interaction: for Forwards questions, notations with Sequential structure (i.e. Nest-INE and Boxes) are faster than Circumstantial, but for Backwards questions Circumstantial structures are faster. Note that times for the Graphics notations are slower.

Above: response times to programs expressed in Sequential Graphics ('Boxes') and Circumstantial Graphics ('Gates'). Below: response times to programs expressed in Sequential Text ('Nest-INE') and Circumstantial Text ('And/Or').

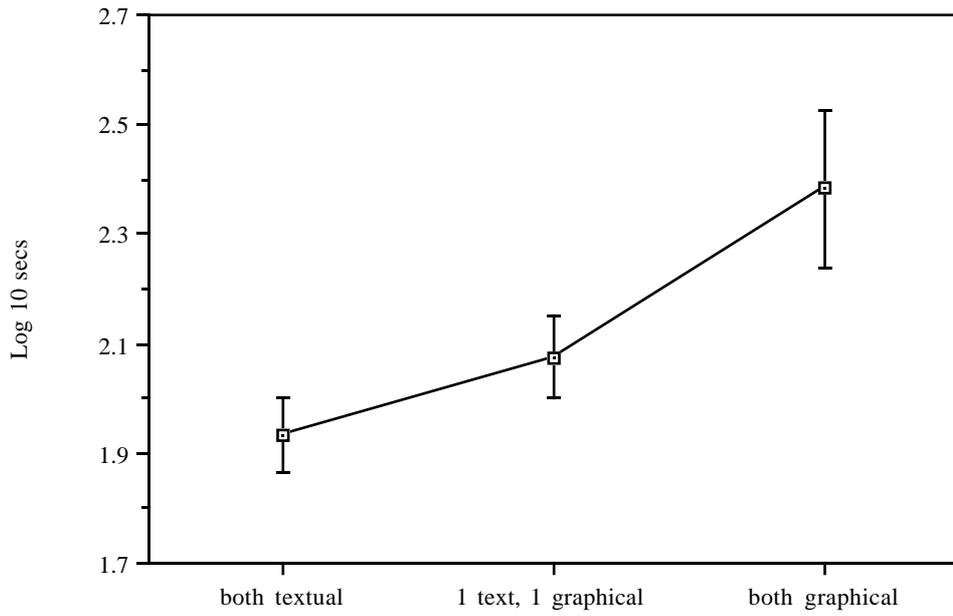


Figure 7: Intrinsic effect of Mode in same-different judgements. Comparing two textual notations was fastest; comparing two graphical notations was slowest. The difference is surprisingly great.

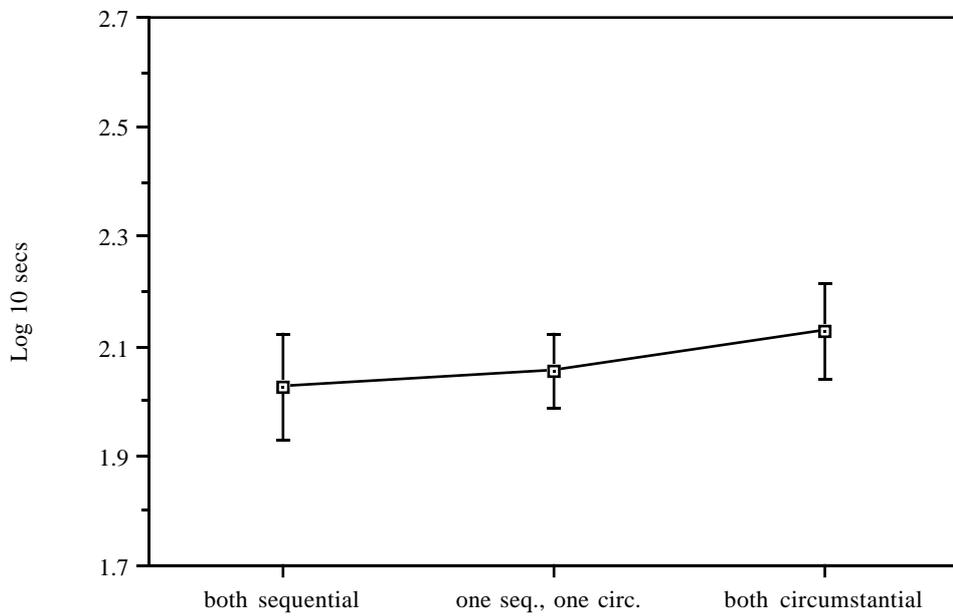
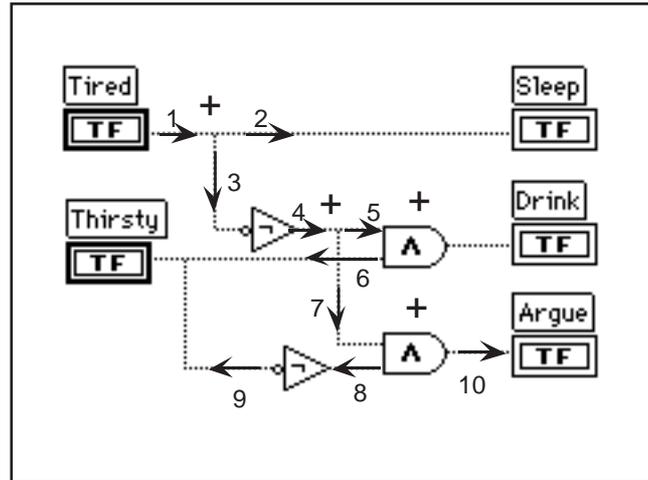


Figure 8: Intrinsic effect of Structure in same-different judgements. The comparisons in which both notations were Sequential were slightly faster than those in which one or both were Circumstantial.



Consider answering a forwards question for a small Gates program (see figure). Suppose the input is Tired=False, Thirsty=False. A plausible account of the cognitive process (agreeing with detailed observations of subject strategies made in the experiment) would be:

Take first truth-value, Tired=False. Scan to find Tired in program. Propagate False along the wire. Place finger at choice point and consider each direction in turn.

- Propagate horizontally first. On reaching next component, Sleep, abandon this branch, since we haven't sent a True to an output yet.
- Now propagate downwards. On reaching the inverter, start to propagate True. On reaching junction, place finger at choice point and consider each direction in turn.
 - Propagate horizontally first. On reaching and-gate, place finger on it and start to search backwards. Search reaches input Thirsty. Look up truth-value, which is False. Return to most recent finger; abandon this branch since the conjunction is false.
 - Return to previous finger. Start to propagate downwards. On reaching and-gate place finger on it and start to search backwards. On reaching inverter, remember to invert next signal found. Search reaches Thirsty which is set to False so inverter is set to True. Return to most recent finger and continue propagating True. Search reaches Argue. Stop with output 'Argue'.

Figure 9: Typical search pattern for a small Gates program, following typical strategies observed in our subjects. The task is to discover the output, given the input Tired=False, Thirsty=False. Numbers in the diagram show order of examining segments; arrows show direction of travel; pointing hands show junctions where fingers (mental or physical) must be placed. Even in this extremely simple case the process is not trivial.

```
Sleep:  if Tired
Drink:  if ¬ Tired & Thirsty
Argue:  if ¬ Tired & ¬ Thirsty
```

This is the And/Or version of the program shown in Figure 9. Given the same input (Tired=False, Thirsty=False) a plausible account of the solution process would be simply:

Try first line. Try first predicate, Tired. No match with the input (Tired=False). Abandon this line.
Try next line. First predicate matches. Try second predicate, Thirsty. No match with input. Abandon.
Try next line. All matches successful. Stop with output 'Argue'.

Figure 10: Search pattern for a small And/Or program. Even a very unsophisticated algorithm, such as 'try each line in turn', is very easy to apply to this notation, with a minimum of working memory required.