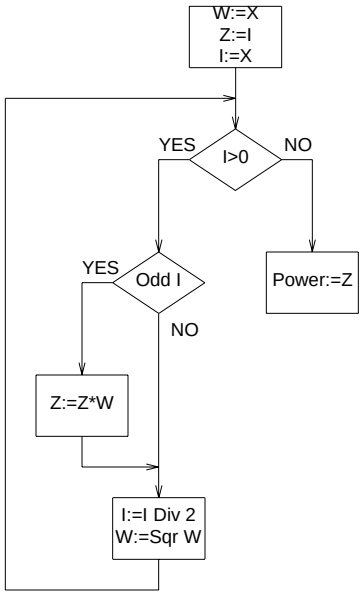
flo—A Language for Typesetting Flowcharts

Anthony P. WOLFMAN and Daniel M. BERRY

Faculty of Computer Science
Technion
Haifa 32000
ISRAEL

ABSTRACT : flo is a language for including *flowcharts* into documents typeset using the UNIX™ ditroff. A basic flowchart can be created with minimal effort by inputting only the basic algorithm written in a Pascal-like notation. The example below illustrates the general capability of flo. The flowchart to the left is obtained from the input to the right.




```
.FL 2
[W:=X;
Z:=1;
I:=Y];
WHILE [I>0]; DO
BEGIN
      IF [Odd I];
            THEN [Z:=Z*W];
      [I:=I Div 2;W:=Sqr W];
END
[Power:=Z];
.FE
```

This input uses default settings except for a sizing parameter in the `.FL` command. flo is a pic preprocessor, which in turn is a ditroff preprocessor. flo lets most of its input pass through untouched; it translates flo commands lying between `.FL` and `.FE` into pic commands that draw the flowcharts.

This paper was typeset camera-ready using flo, pic, ditroff, and other ditroff preprocessors.

KEY WORDS : Flowcharting, Typesetting, Ditroff, Pic.

1. Introduction

Many papers written in computer science deal with algorithms. In many but not all cases, a flowchart, either by itself or accompanied by a program in some programming language, is a convenient representation of the algorithm. Indeed, as a result of recent experiments showing the superiority of flowcharts over pseudocode for helping programmers understand algorithms [Scanlon 1989], flowcharts may be coming back into fashion!

In addition, today, almost all papers in computer science are prepared with the aid of some formatting system capable of preparing output for printing on devices, such as laser printers and phototypesetters, that are capable of drawing arbitrary figures. These formating systems include batch-oriented systems such as troff [Ossana 1976], ditroff [Kernighan 1982], TEX™ [Knuth 1984], LATEX [Lamport 1984], and scribe [Reid 1980], as well as a host of WYSIWYG programs running on systems with high-resolution screens.

Many of these formating systems include facilities by which non-textual material may be included and formated along with the text. These include bibliographical citations, line-oriented pictures possibly with some limited filling, graphs (charts), directed graphs, tables, formulae, source program code, arbitrary POSTSCRIPT™ documents, and back-of-document indices. Space limitations preclude giving more detailed citations.

There exists no suitable tools integrated into the ditroff family for producing and including into ditroff documents flowcharts such that the description of the flowchart is its algorithm rather than its physical layout or topology.

The project described herein was to develop a pic [Kernighan 1984] preprocessor, called flo, that prepares a flowchart given a linear representation of an algorithm. Since there exists a version of pic, called tpic, that can be used with LATEX, flo can be used to prepare flowcharts for inclusion in documents typeset with either ditroff and LATEX. Thus, flo is useful for those situations in which it is felt that a flowchart will help; in other situations, it is obviously of no use.

This paper describes the design and implementation of flo. A complete description of the use of the language is found in [Wolfman 1989]. As is common with papers describing a new formating tool, this paper was typeset using flo, pic, ditroff, and other ditroff preprocessors, preparing output for a POSTSCRIPT printing device. The command lines to print this paper were

```
refer -l -e -n -p refsidx -sADT paper | \
      fix.bibliography.labels > paper.ref
flo paper.ref | pic | tbl | eqn | psroff -mcup
```

All the diagrams in this paper were prepared as flo inputs.

The following example demonstrates the capabilities of flo. Besides the algorithm in a Pascal-like notation, this example has additional commands and attribute settings that adjust the sizes of nodes, spaces between adjacent nodes, and arc placement. This fine-tuned example differs from the purely algorithmic example of the abstract, in which all of the layout is by flo supplied defaults. Given the input[1],

```
.ps 7
.FL
@pic {scale = 1.4} ;
defshape ends shape is oval:
      {ellipse ht $1 wid $2} shapew is 0.6;
stmtshapeh is 0.25 ;
queryshapeh is 0.3 ;
spaceh is 0.25;
spacew is 0.2;
[START] with ends;
[(y1,y2,y3,y4)←(x1,x2,1,0)] shapew is 1.7;
WHILE [y1>y2];
      DO [(y2,y3)←(2y2,2y3)] shapew is 1.2;
LOOP
      IF [y1≥y2] ;
            THEN [(y1,y4)←(y1−y2,y4+y3)] shapew is 1.5;
      EXITIF [y3=1] config is RIGHT;
      [(y2,y3←(div(y2,2),div(y3,2))] shapew is 1.9;
      @up ;
END
[(z1,z2)←(y1,y4)] shapew is 1.1;
[HALT] with ends;
.FE
.ps
```

flo produces the flowchart in Figure 1.

2. Previous Solutions

From the need to provide better documentation for computer programs, a host of

[1] For clarity, the input is shown after processing by eqn. The text that has been processed by eqn is shown in the Helvetica sanserif font to make it standout against the Courier typewriter font normally used to show input. The same holds for all other examples involving eqn text. The full input for this first example appears in [Wolfman 1989].

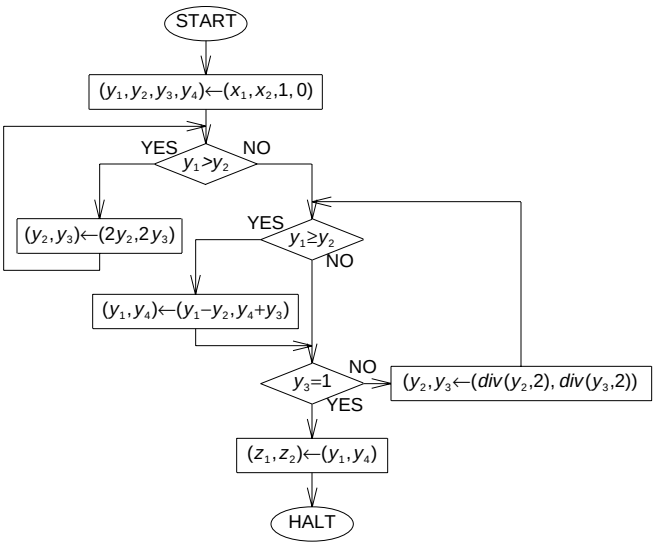


Figure 1: Flowchart with eqn text

programs to generate flowcharts have been written. For a full summary of these programs see [Wolfman 1989]. There are five basic kinds of programs that have been used for drawing flowcharts. The first kind, the batch general flowchart drawing program, e.g., that reported in [Knuth 1963], attempts to be general enough to handle any flowchart, but in fact, handles none very well. Flowcharts are spread over several pages with the use of a common label appearing at the nodes in place of a cluttering, possibly inter-page, arc drawn between the nodes. The other kinds all require the user to work directly with the topology of the flowchart rather than just the algorithm it represents. These include the WYSIWYG flowcharting drawing program, e.g., flo draw 1.10 [Freund 1987], the batch general directed graph drawing program, e.g., dag [Ganser 1988] or drag [Trickey 1988], the batch general figure drawing program , e.g., pic, and the WYSIWYG general figure drawing program, e.g., fig [Sutanthavibul 1988].

3. Goal and Requirements

The goal of this project was to enable the user to input an algorithm in some sort of psuedo-Pascal notation and get as output a flowchart of that algorithm. The program, called flo, should behave as follows. The input to flo should be embeddable in ditroff input surrounded by `.FL` and `.FE`. The flo program

should let most of the input pass through untouched. It should transform the flowchart description lying between the .FL and .FE into a pic specification of the requested flowchart, laid out nicely according to the user's needs.

It should be that by inputting only the algorithm, provided that the resulting flowchart is not too big, some flowchart is produced. Moreover, it should be possible to adjust the layout of the flowchart by merely adding to the basic algorithm additional layout information. When the effect of this information is to be global, it should be possible to give it once. When the effect should be local to a particular node or arc, it should be possible to give it as part of the node's or arc's specification. This layout information should look almost like comments added, as an afterthought, to the algorithm, and not like part of the algorithm.

As flo produces pic code, and ditroff forces all pic pictures to fit within a page, flo flowcharts are constrained to fit on one page. To get a multi-page picture, the user must divide the picture explicitly into one page pictures. Since nodes have a certain minimum readable size, this one page limitation limits the complexity of the flowcharts that can be specified. Moreover, nowadays, most programs are built using structured programming techniques, which yield well-nested, goto-less programs.

In building the flo language and program, advantage can be taken of the nature of one-page flowcharts to build a program that draws small, structured flowcharts well, possibly at the expense of slightly less smooth performance for the more complicated flowcharts. For the cases in which the program would yield a layout not what the user had in mind, the language should provide commands by which the user may direct the construction of the layout. The program may also allow the user to fall back to pic and to ditroff if necessary, just as pic allows the user to fall back to ditroff

Equally important is to exclude from flo's requirements the ability to handle other software-related diagrams, such as module diagrams and data-flow diagrams. Certainly, these can be faked using flowcharts elements with nonstandard shapes and inverse translation to a nonsense pseudo-Pascal algorithm. However, it is better to use pic, dag, or drag for a more direct representation.

4. The Development Method

There were two main parts to this project. The first was to design the flo language. The second was to design of the flo program. Which should be designed first, the language or the program? At first, there might seem to be no problem. After all, the language is designed first and then its processor is written.

It is necessary to know the language in order to be able to write the processor. However, it is all too easy to pile feature after possibly unimplementable feature into a language. On the other hand, it is necessary to know what is implementable in order to know what language features are reasonable.

This cycle was broken, by first writing a draft of the language's user's manual, which contained example-laden explanations of all the desired features. As the program was not written yet, the first author hand-translated each flo input example into the pic input that he expected flo would create. After that, he attempted to write a program that would process the language described by the manual, that would translate each flo example into something equivalent to the hand-generated pic input for the example. Whenever this attempt got bogged down in syntactic, semantic, or synergistic problems, missing or unimplementable features, inconsistencies, or just plain messiness, work on the program stopped and another iteration was begun. These discoveries led to changes in the manual and corresponding changes in the program. This process was repeated until a manual and a program were created such that the manual described the entire language handled by the program and the program translated all flo input examples into suitable pic input. Typesetting a manual so related to a program is a also good test of the program.

In this interative process, the second author played an extremely critical, picky customer and user who was particularly grouchy at the slightest sign of inconsistency, nonuniformity, and nonorthogonality.

The manuals for pic and grap [Bentley 1984] exhibit the same relation to their languages. Thus, we suspect that the same method was followed by Kernighan and Bentley in the development of pic and grap.

5. Details of Implementation

5.1. Major Semantic Problems

The program has to be designed in such a way to enable an easy automatic layout in most cases. At the same time, it has to supply a handle to enable the user to control the layout directly if need be. This handle has to enable the user a multitude of alternate layout types depending on flow direction, condition configuration, node size, space between nodes and general flowchart structure The program must also be designed to avoid intersecting loops and arrows. The following subsections discuss the more interesting elements of the solution.

5.2. Bubbles

After examining a great deal of flowcharts, it was noticed that flowcharts took on different dimensions not only depending on their internal structure and node size, but also on the spacing between the nodes. For example, the same flowchart could seem short and fat and long and thin depending only on the spacing between the nodes. Therefore, it was decided to add to flo the ability to control these dimensions. At first, the first author thought only to let the user specify the type of flowchart, i.e., "fat", "thin", etc. Such a general description is too fuzzy to be the basis of an algorithm. On the other hand, to have to specify all the exact dimension is too burdensome on the user. Finally, the first author got the idea of bubbles.

A node's *bubble* is defined as a bounding box around that node. No other node, other node's bubble, or return loop may encroach on a given node's bubble. The only entity that may cross a bubble boundary is an arrow connecting two nodes, and the arrow must be going to or from the node contained inside the bubble. The entire flowchart is therefore a mosaic of bubbles, not unlike the way a TEX-formatted document is a mosaic of boxes, each containing a unit of text [Knuth 1984].

5.3. Chewing Gum

The presence of query (conditional) nodes causes layout problems. After each query there are two sub-flowcharts one for the *Yes* answer and one for the *No* answer. Figure 2 shows a flowchart with two sub-flowcharts marked. The sub-flowcharts themselves may contain query nodes and therefore nested sub-flowcharts. A sub-flowchart is a series of nodes beginning with the first node following a query on one of its answer arcs end ending with the last node under control of that answer of the query or with the last node before looping back to before the query. The special series of nodes beginning with the very first node and ending with the last node or the first query node is called the root sub-flowchart. The dimensions of each sub-flowchart is known only after it has been drawn completely. Each sub-flowchart's position in relation to its sibling sub-flowchart can be known only after the dimensions of both sub-flowcharts have been calculated. As a first approximation, each sub-flowchart is drawn immediately below its query node. When the dimensions of both sub-flowcharts of a query node are finally calculated the sub-flowcharts are pulled as close together as their bubbles allow, as if held together by an elastic chewing gum. This chewing gum has certain similarities to TEX's *glue* [Knuth 1984].

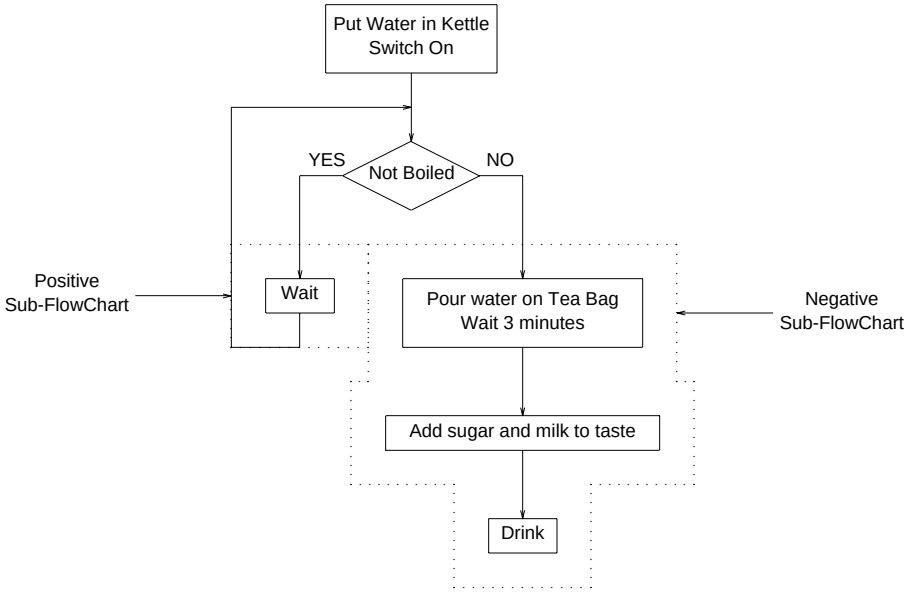


Figure 2: Flowchart with two sub-flowcharts marked

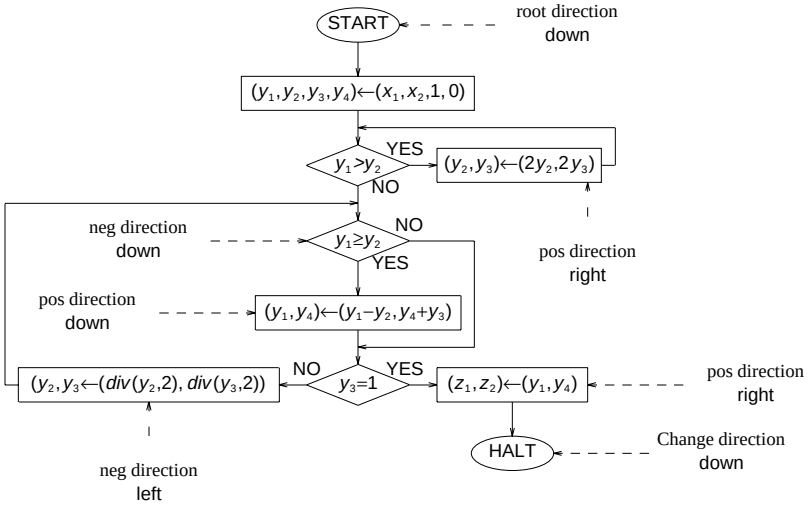


Figure 3: Flowchart with changes of direction marked

5.4. Configurations

After examining numerous flowcharts, it was noticed that the layout of any flowchart is determined by the position of the sub-flowcharts of its query nodes. *Query configurations* were introduced to specify these positions. There are twelve configurations in all, six basic ones and and six variations of these, which reverse the positions of the *Yes* and *No* arcs.

After settling on this method of controlling layout, the major design problem was naming the configurations. They were originally named `C1`, `C2`, `C3`, etc. Reacting to his own difficulty remembering which one is which, the second author suggested using names mnemonic of their layout, such as `O`, `P`, `Q`, `DASH`, `LEFT`, `RIGHT`, and the ...`N` variations thereof.

5.5. Direction & Movement

In order to be able to control layout, it is necessary also to be able control direction of flow. Normally, the direction of flow is determined by the configuration of the last query. For example, the direction of both sub-flowcharts of an `O` configuration query is `down`. The user may want, however, to change the direction of the flow within a sub-flowchart.

The *direction* of the flow is the direction from the last node to the current one. The default direction of the root sub-flowchart is `down`. After a query node, the direction of each sub-flowchart is defined by the query's configuration. A direction may be changed by using the appropriate command. `flo` does not allow changing the flow direction at the beginning of a sub-flowchart because doing so may cause a conflict with the configuration. Figure 3 shows a flow chart in which every change of direction and initial direction of a sub-flowchart is marked. Only when the desired direction differs from the default or configuration-defined direction, must a direction command be given. The direction commands must be used carefully, because they can conflict with the configuration system.

A node may also be moved in relation to its default position. The parameterless `@move` command moves the subsequent node in the current direction a distance equal to the size of the node's bubble. An optional argument provides a multiplier to the movement. The unit of movement was chosen to be the size of the bubble of the next node, because this distance is the most obvious unit to a user who is looking at a version of the flowchart. Figure 4 shows a flowchart in which one, marked, node has been moved a single unit. The figure also exhibits the portion of the input containing the `@move` command.

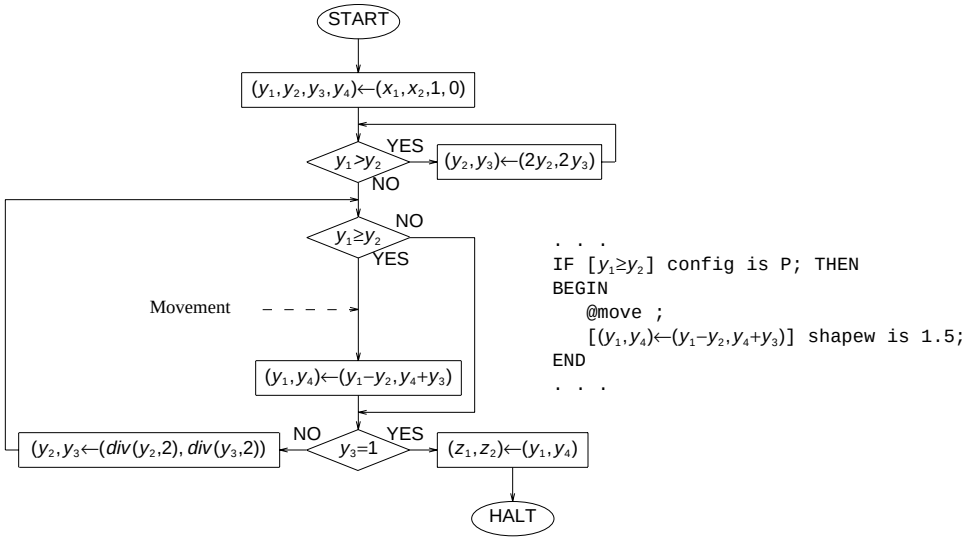


Figure 4: Flowchart with movement marked

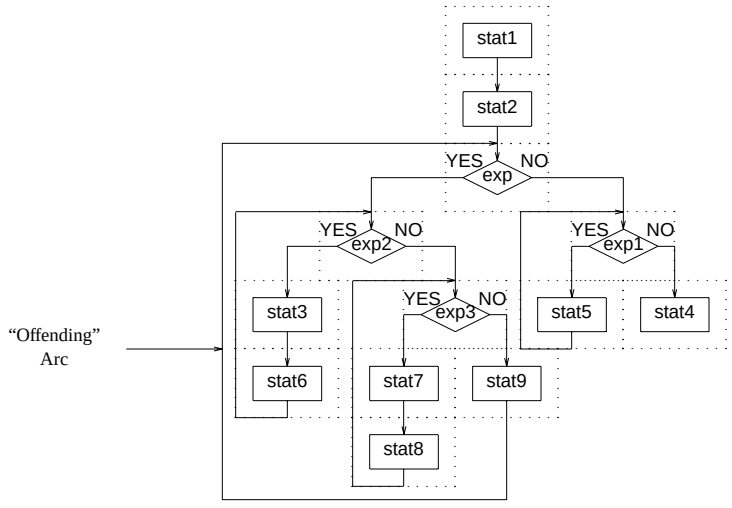


Figure 5: Flowchart with nested loop and offending arc marked

5.6. Routing

As might be expected, the most troublesome arcs to draw are the *loopback* arcs. Fortunately, the language is made up of only the so-called structured control-flow commands, conditionals, while loops, repeat loops, etc., with *no* gotos allowed. This constraint completely eliminates intersecting loops, because all loops are nested one inside the other. It also completely eliminates arcs from one sub-flowchart to another. With flo providing all of these control structures, a goto is hardly ever needed. Ultimately, the user can program goto arcs using embedded pic commands. The extra complexity that a goto command would add to flo's routing algorithm makes having it too expensive.

In computing the layout of the nodes, any non-nested loop can be drawn immediately, because its loopback arc has only to go round its own local sub-flowchart, whose dimensions are known. A nested loop is trickier. Its loopback arc may have to go round a sub-flowchart that has not yet been drawn. Call this sub-flowchart the *offending sub-flowchart* and the arc the *offending arc*. Then, the exact layout of the offending arc can be determined only when the offending sub-flowchart has been drawn. The chewing gum pulls the arc to hug the bubble of the offending sub-flowchart. Figure 5 contains an example of a nested loop. The bubbles are shown to show how the offending arc's path is determined.

5.7. pic *Macros*

In order to facilitate debugging of flo and to make it easy to add new constructs in the future, all of the basic building blocks and routing patterns are implemented as pic macros in two files, db.pic and route.pic, includable from pic input. The pic input that flo creates is but a series of invocations of these macros. db.pic contains the shape macros, and route.pic contains the routing macros. Both of these files are copyd at the beginning of each flowchart definition.

This approach greatly reduced the development time of flo. For a great deal of the bugs found during the development, it sufficed to change the macro definitions, and it was not necessary to recompile the flo program.

5.8. Data Structure

Each flowchart drawn by flo is represented as a directed tree graph in which each vertex represents a sub-flowchart and the edges represent the parent-to-child relation among the sub-flowcharts. The vertices are implemented as data records and the edges are implemented as pointers to these data records. Each vertex's data record contains a pointer to the list of the individual nodes in the vertex's

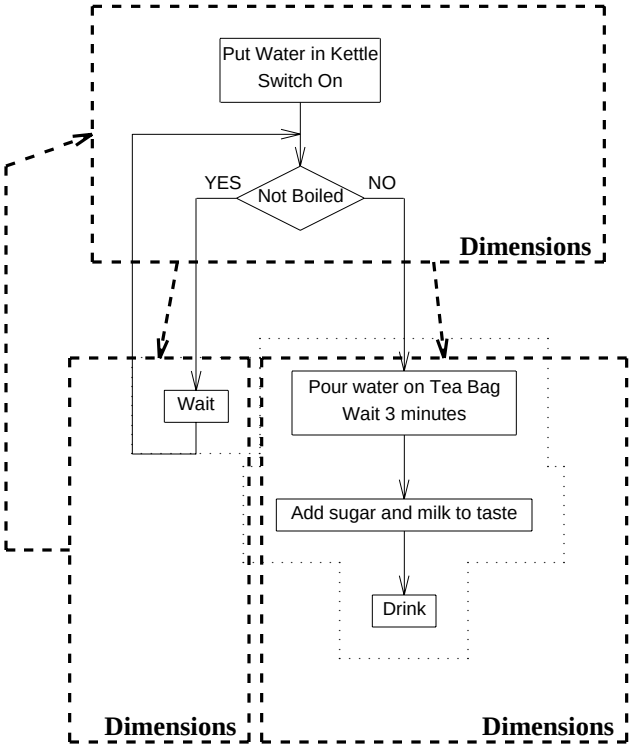


Figure 6: Flowchart superimposed on its data structure

sub-flowchart, and the calculated dimensions of the vertex's children. The list elements contain the text and attributes of the nodes. If a vertex has no children, and its sub-flowchart's nodes are in a loopback path, then the vertex's record contains a pointer to the node to which the loopback arc is directed. Figure 6 shows the data structure of the flowchart of Figure 2.

6. Conclusions

In most cases, from just the algorithm, flo produces a correct and aesthetically pleasing flowchart, as required by the goals of this project. The global and local attribute specifications allow the user to customize the appearance of the flowchart.

The main advantage of flo over some of the programs mentioned in Section 2 is that usually all that is needed is the basic algorithm. The user hardly ever needs to be concerned about the layout, as determining the layout is flo's job. If any changes are required in the algorithm, flo automatically rearranges the layout. On

a WYSIWYG system, the user has to rearrange the layout, and with dag and drag, the user has to input the new topology of the flowchart.

A big drawback of flo is one inherited from pic. Like pic, flo cannot draw a node to fit round any text. The user may either specify the size of each node locally to fit exactly round its own text or specify a single global size large enough to fit round the largest text item.

There are certain algorithm constructs that flo does not support. For example, flo cannot handle certain types of nested *Repeat* loops. To handle them, the user has to fall back on pic commands. This problem never occurred in real life; it occurred only during testing during which all sorts of anomalous algorithms were presented to flo. In fact, the authors have yet to encounter a real live algorithm that flo cannot handle.

On the whole, the authors have found flo a useful and easy-to-use tool, that seems to work even under fire. The authors found that the flowcharts flo produces are more symmetric and better aligned than flowcharts laid out by hand in the early versions of the user's manual!

flo has withstood the ultimate test! A member of the first author's master's thesis examining committee, hell-bent on tripping up flo, gave to the first author an algorithm to flowchart. flo worked the first time and produced a flowchart that was pleasing *even* to that committee member!

Among the suggestions for future work are (1) an include file facility, enabling the user to keep a file of commonly used macros and global changes and include them in any flo input, (2) to be able define shape by pic macros, (3) to be able to define construct macros, e.g,. a *for* construct macro, and (4) to be able to specify arcs as splines. All these enhancements can be added with relative ease. All that is needed is to add the appropriate definition to the lexical analyzer and to write the handling procedure. In some cases, the handling procedure can be borrowed from the pic program.

Acknowledgements

POSTSCRIPT is a trademark of Adobe Computer Systems. TEX is a trademark of the American Mathematical Society. UNIX is a trademark of AT&T Bell Laboratories.

REFERENCES

[Bentley84] J.L. Bentley and B.W. Kernighan, 'GRAP — A Language for Typesetting Graphs, Tutorial and User Manual', Computing Science Technical Report No. 114,

AT&T Bell Laboratories, Murray Hill, NJ 07974 (December, 1984).

[Freund87] G. Freund, 'flo draw 1.0', Program for IBM PC-Compatibles (1987).

[Ganser88] E.R. Ganser, S.C. North, and K.P. Vo, 'DAG—A Program that Draws Directed Graphs', *Software—Practice and Experience*, **18** (11), 1047-1062 (November, 1988).

[Kernighan82] B.W. Kernighan, 'A Typesetter-independent TROFF', Computing Science Technical Report No. 97, Bell Laboratories (March, 1982).

[Kernighan84] B.W. Kernighan, 'PIC — A Graphics Language for Typesetting, Revised User Manual', Computing Science Technical Report No. 116, Bell Laboratories (December, 1984).

[Knuth63] D.E. Knuth, 'Computer-Drawn Flowcharts', *Communications of the ACM*, **6** (9), 555-563 (September, 1963).

[Knuth84] D.E. Knuth, *The* TEX*book*, Addison-Wesley, Reading, MA, 1984.

[Lamport84] L. Lamport, LATEX*: A Document Preparation System*, Addison-Wesley, Reading, MA, 1984.

[Ossana76] J.F. Ossana, 'NROFF/TROFF User's Manual', Technical Report, Bell Laboratories (October 11, 1976).

[Reid80] B. Reid, 'Scribe: A Document Specification Language and its Compiler', Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA (October, 1980).

[Scanlon89] D.A. Scanlon, 'Structured Flowcharts Outperform Pseudocode: An Experimental Comparison', *IEEE Software*, **6** (5), 28-36 (September, 1989).

[Sutanthavibul88] S. Sutanthavibul, 'fig', Program implemented at the Computer Science Dept.,, University of California, Berkeley (1988).

[Trickey88] H. Trickey, 'DRAG — A Graph Drawing System', in *Electronic Publishing '88*, ed. J. André and H. van Vliet, Cambridge University Press, Cambridge, UK, pp. 171-182, (1988).

[Wolfman89] A.P. Wolfman and D.M. Berry, 'flo—A Language for Typesetting Flowcharts', Technical Report, Computer Science, Technion, Haifa, Israel (December, 1989).