Improving Visual Impact Analysis

Matthew Hutchins[1] and Keith Gallagher[2]
CSIRO Mathematical and Information Sciences
GPO Box 664, Canberra, ACT 2601, Australia.
Matthew.Hutchins@cmis.csiro.au, Keith.Gallagher@cmis.csiro.au

Abstract

Visual impact analysis is a software visualisation technique that lets software maintainers judge the impact of proposed changes and plan maintenance accordingly. An existing CASE tool uses a directed acyclic graph display derived from decomposition slicing of a program for visual impact analysis. In this paper, we analyse the graph display and show that it is semantically ambiguous and fails to show important information. We propose requirements for an improved display based on a definition of "interference" between variables in a maintenance context. The design for a new display is presented, with a series of examples to illustrate its effectiveness. The display is focused on providing a straightforward method to analyse the impact of changes.

1. Introduction

The Surgeon's Assistant [2] is a CASE tool that uses program slicing [1, 6, 7] to assist maintainers and developers of C programs. Specifically, the tool enforces a maintenance model based on the use of decomposition slices of a program [3]. A decomposition slice on a variable is the set of all program statements that contribute to the computation of the variable. It is a generalisation of a program slice, in that it depends only on the variable, and not on a particular point in the program. The Surgeon's Assistant uses decomposition slices to analyse and limit the scope of changes to the program. Once a variable is selected for maintenance, the source code of the program is decomposed into a minimal changeable set of statements and its unchangeable complement. A context sensitive editor allows changes only to the changeable parts. The slice-based decomposition ensures that the changes only affect the computation of the selected variable and any that directly depend on it, all others are guaranteed to be unaffected. As a result, only the changed portion of the code needs to be tested, and no other regression testing is necessary.

The Surgeon's Assistant incorporates a Decomposition Slice Display System that presents a visual representation of the decomposition slices of the program to the maintainer. The display uses a tool called VCG [5] to show the slices as nodes in a directed, acyclic graph. An example is shown in Figure 1. The edges of the graph represent the

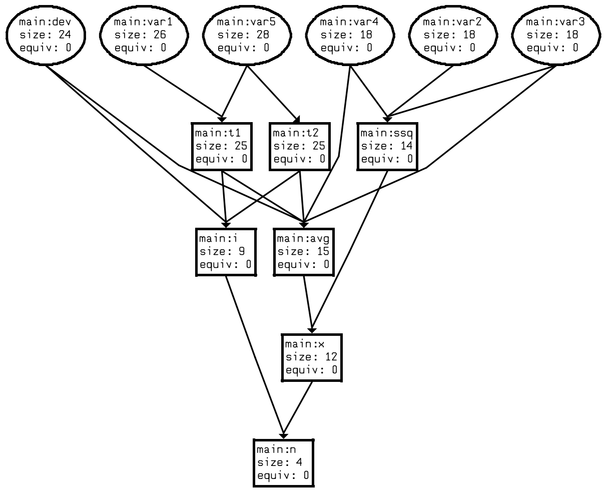

main:dev size: 24 equiv: 0
main:var1 size: 26 equiv: 0
main:var5 size: 28 equiv: 0
main:var4 size: 18 equiv: 0
main:var2 size: 18 equiv: 0
main:var3 size: 18 equiv: 0
main:t1 size: 25 equiv: 0
main:t2 size: 25 equiv: 0
main:ssq size: 14 equiv: 0
main:i size: 9 equiv: 0
main:avg size: 15 equiv: 0
main:x size: 12 equiv: 0
main:n size: 4 equiv: 0

Figure 1: Strong dependence graph for variance program.

strong dependence relation between variables, which is equivalent to the containment relation on the associated slices. That is, a variable *a* is strongly dependent on *b* (there is an edge from *a* to *b*) only if the decomposition slice on *a* is a superset of the slice on *b*. In this case, the computation of *a* relies on the computation of *b*, so any

1. The authors wish to acknowledge that part of this work was carried out within the Cooperative Research Centre for Advanced Computational Systems established under the Australian Government's Cooperative Research Centres Program.
2. On sabbatical leave from Loyola College in Maryland, USA.

change to *b* could potentially affect *a*. Hence, the graph provides the maintainer with a tool to carry out visual impact analysis [2]. The position of a node within the graph offers an indication of the complexity of maintenance on a particular variable. A variable with a node that has no incoming edges (shown at the top of the graph) will have no impact on other variables, while a variable at the bottom of the graph may affect most or even all of the other variables. The facility for visual impact analysis is enhanced by the ability to interactively manipulate the graph, for example by selecting a variable and highlighting all those that depend on it.

Although an improvement over a strictly textual presentation of the slices, the strong dependence graph has some weaknesses, and this paper reports on an investigation into an improved visual display for the Surgeon's Assistant. Section 2 presents some observations about the graph display, leading to a set of requirements for an improved display discussed in Section 3. Section 4 presents the design for a new display, and a series of examples are given in Section 5.

2. Observations on the strong dependence graph

2.1 The strong dependence relation

Decomposition slicing can be thought of as a total function from the variables of a program to sets of statements from the program. The function is not *onto*: not every set of statements represents a decomposition slice. It is also not *one-to-one*: sometimes, two or more variables have the same decomposition slice. The nodes in the strong dependence graph are labelled with the names of variables, but there is some ambiguity as to whether they represent variables or their associated slices. The ambiguity is an issue for variables that have equivalent slices. Gallagher and O'Brien [4] report on a method for reducing the complexity of strong dependence graphs by collapsing the equivalent variables into a single node with a thickened border. This represents a semantic shift from showing variables to showing decomposition slices.

The containment relation on decomposition slices is a partial order (it is transitive, reflexive and antisymmetric). The strong dependence graph display exploits this structure to lay out the graph neatly and reduce the number of edges required. No edges are shown from a node to itself, and no edges are shown when the relationship can be deduced from transitivity. This is a strength of the display, as it significantly reduces the visual complexity, as shown in Figure 2.

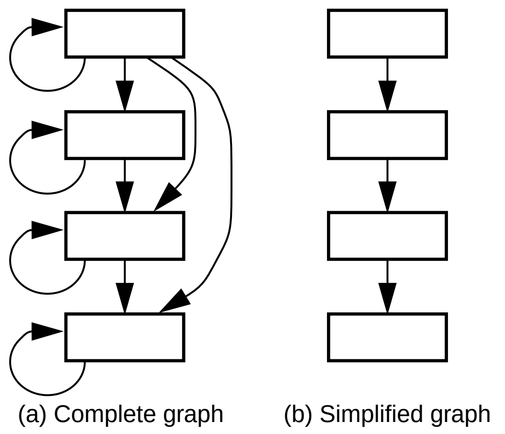


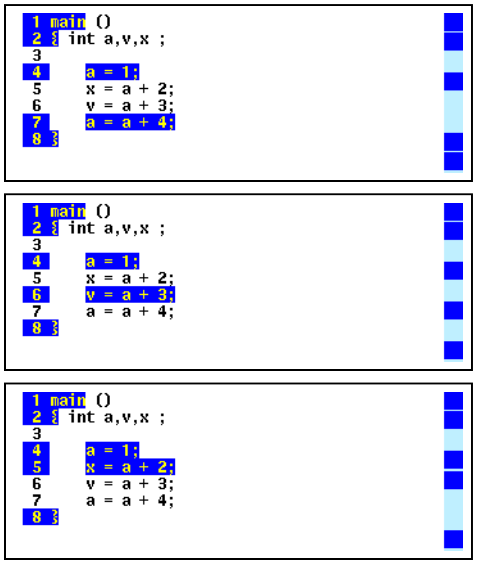
Figure 2: Reducing edges in a strong dependence graph

Because the slicing function is not one-to-one, the strong dependence relation between variables is not a partial order (it is not antisymmetric). Once again, this introduces ambiguity into the display. Should dependence edges be shown between equivalent variables? How should equivalent variables be organised vertically? (Vertical layout is normally used to reinforce the transitive containment relation.)

These observations represent fairly minor weaknesses in the graph display that probably don't affect the comprehension of the display by the maintainer, but do have a significant impact on the extendibility of the display. This investigation was initiated by a desire to include an extra relation (weak dependence) in the graph. To add a new relation to the graph, however, requires a firm semantics for what is shown already, and can also invalidate the “implied” information in the graph, such as the transitive dependence.

2.2 The weak dependence relation

When two variables have decomposition slices that contain statements in common, the variables are said to be weakly dependent. Variables that have equivalent slices are weakly dependent, as are any pair of variables where one is strongly dependent on the other. Two variables that are mutually dependent on a third are also weakly dependent. It is possible for variables to be weakly dependent without there being a strong dependence involved, as illustrated by the three decomposition slices of the simple program in Figure 3. All of the slices have statements 1,2,4 and 8 in common, but no slice completely contains any other. The strong dependence graph for this program is shown in Figure 4. Obviously, the strong dependence graph does not capture the relationship between these variables. In particu-



```
1 main ()
2 { int a,v,x ;
3
4     a = 1;
5     x = a + 2;
6     v = a + 3;
7     a = a + 4;
8 }
```

```
1 main ()
2 { int a,v,x ;
3
4     a = 1;
5     x = a + 2;
6     v = a + 3;
7     a = a + 4;
8 }
```

```
1 main ()
2 { int a,v,x ;
3
4     a = 1;
5     x = a + 2;
6     v = a + 3;
7     a = a + 4;
8 }
```

Figure 3: Three slices of a program illustrating weak dependence.[1,2]

1. Thanks to Arun Lakhotia for the program.
2. Slices generated and displayed by *unravel* (`http://hissa.ncsl.nist.gov/~jimmy/unravel.html`).

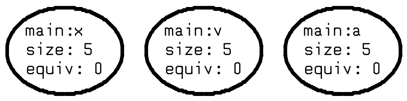


Figure 4: Strong dependence graph for the program in Figure 3.

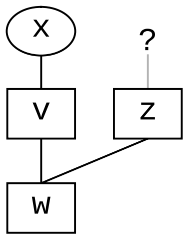
lar, maintenance on any one of these variables could be affected by the others. This example does not invalidate the decomposition approach to maintenance, as long as unchangeability is allowed to dominate changeability. To make a change to variable *x*, the slices on *v* and *a* would be made unchangeable, leaving only statement 5 free to be changed. The problem is in the display, in that it suggests that the whole of the decomposition slice on *x* should be changeable, when in fact the weak dependence prevents this.

A potential solution to this display problem is to include an additional set of distinct edges on the graph to indicate weak dependence. As described above, however, the properties of the strong and weak dependence relations make them incompatible. Weak dependence is not necessarily transitive, so all edges would have to be included. It is symmetric, so edges would have to be undirected (or bidirectional). This would conflict with the layout requirements derived from the strong dependence relation, creating a messy graph with many edge crossings. A further complication arises from the preponderance of weak dependence. In any graph with a single “minimal” node, all of the variables will be weakly dependent on each other, because they will all have some code in common. This sort of dependence does not provide the maintainer with any new information, however. The important goal is to capture the “hidden” weak dependence that is not implied by the strong dependence graph.

3. Requirements for an improved display

3.1 The interference relation

The key requirement for a new display is that it closely address the task of maintenance on a selected variable. In the context of the program decomposition model of the Surgeon's Assistant, the key question the maintainer must have answered is: “if I need to change the computation of this variable, what computations must I be prepared to change, what computations must I leave unchanged, what computations could interfere with the change, and what computations can I safely ignore.” To illustrate this question, consider the slices illustrated in this strong dependence graph:



Suppose *v* needs to be changed, then:

- every statement in *v* is a statement in *x*, so any change to *v* is a change to *x*. Therefore, *x* must be changeable.
- the statements in *w* are a subset of the statements in *v*, but can not be changed. At most, the statements in the difference between slices *v* and *w* can be changed.
- if *v* and *z* are equivalent slices, then any change to *v* is a change to *z*, and so *z* must be changeable.
- slices *v* and *z* have at least all of the statements in *w* in common. If that's all they have in common, then slice *z* can be safely ignored. As long as *w* is unchangeable, no change can affect *z*.

- there could be statements in both *v* and *z* that are not in *w*. In this case, *z* interferes with maintenance on *v*. The maintainer will have to decide to either modify *v* without changing the statements in *z*, or make *z* changeable also.

From this example, we derive the following general definition of interference between variables: variable *a interferes with* variable *b* if and only if *a* is not strongly dependent on *b* and *b* is not strongly dependent on *a*, but the intersection of the decomposition slices on *a* and *b* is not a subset of the union of the decomposition slices of the variables that are strongly dependent on, but not equivalent to, *b*. Put more simply, the decomposition slices on *a* and *b* have statements in common that are not explained by the strong dependence relation applied to *b*. To repeat, this is "interference" in the sense that the maintainer will need to consider and make a decision about any interfering variables before making a change. The interference relation is different from both the strong and weak dependence relations. It is not necessarily symmetric nor transitive.

3.2 Requirements derived from interference

Based on the maintenance question and the analysis of interference in the previous section, we derive the following requirements for an improved display:

- show the set of variables from the program.
- between each pair of variables, show the relationships that affect maintenance: equivalence, strong dependence, interference.
- for each variable, show the set of variables that will need to be considered during maintenance on that variable.
- highlight a set of variables selected for maintenance, and their relationships to the other variables.

3.3 Scalability

One of the most significant problems faced by this or any other system using software visualisation is the issue of scalability — the decomposition slice display must scale up to larger systems. The strong dependence graphs generated by VCG are reasonably effective up to around two hundred nodes, especially when explored interactively. It is an important requirement of any new display to at least not do any worse than this.

Although we offer no general solution to the scalability problem, two approaches to managing some of the complexity of the display are: show or hide information on demand in semantically significant units, and collapse semantically related items to make compound items. Based on these heuristics, the following mechanisms can be used to control the display:

- any variable, or all of the variables from particular procedures, can be included or excluded on request.
- variables with "empty" slices (those of size three, as discussed by Gallagher and O'Brien [4]) can be excluded on request.
- variables that have no interesting relationship to the current selection can be excluded on request.
- equivalent variables from the same procedure can be grouped into a single item.
- the slices of *all* of the variables from a particular procedure can be grouped into a single item.

4. An alternate display

4.1 Basic design

In response to these requirements we have designed a new display for the Surgeon's Assistant that focuses on answering the maintenance question asked in Section 3.2. An example is shown in Figure 5, along with a strong dependence graph for comparison. (The example, and the apparent discrepancies between the new display and the graph, are discussed in Section 5.1.) The general features of the design are:

- variables are shown in rectangular "nodes" arranged diagonally down the display.
- the nodes are labelled with a procedure and variable name.
- grouped variables are indicated textually with a description in braces (no grouped variables appear in Figure 5).
- each variable has an "interference line" that (conceptually) enters the node from the right and leaves through the top.
- the interference lines for each pair of variables intersect precisely once.
- at the intersection for each pair, a symbol indicates the maintenance relationship between them (no symbol appears when there is no relationship). The symbols are discussed in Section 4.3.
- the variables are sorted so that (where possible) interference "flows" in from the right and out the top. Sorting is discussed in Section 4.2.

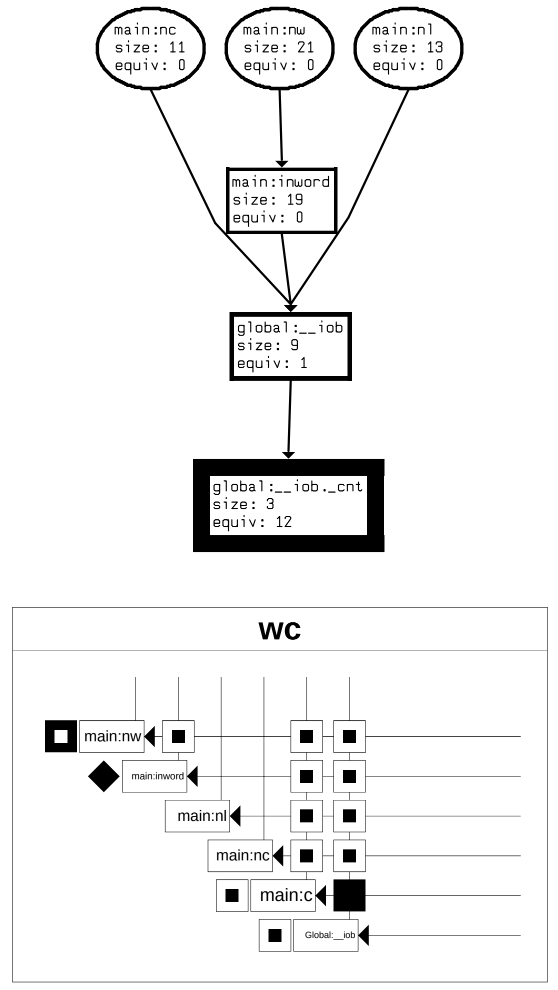


Figure 5: Strong dependence graph and new display for word count program.

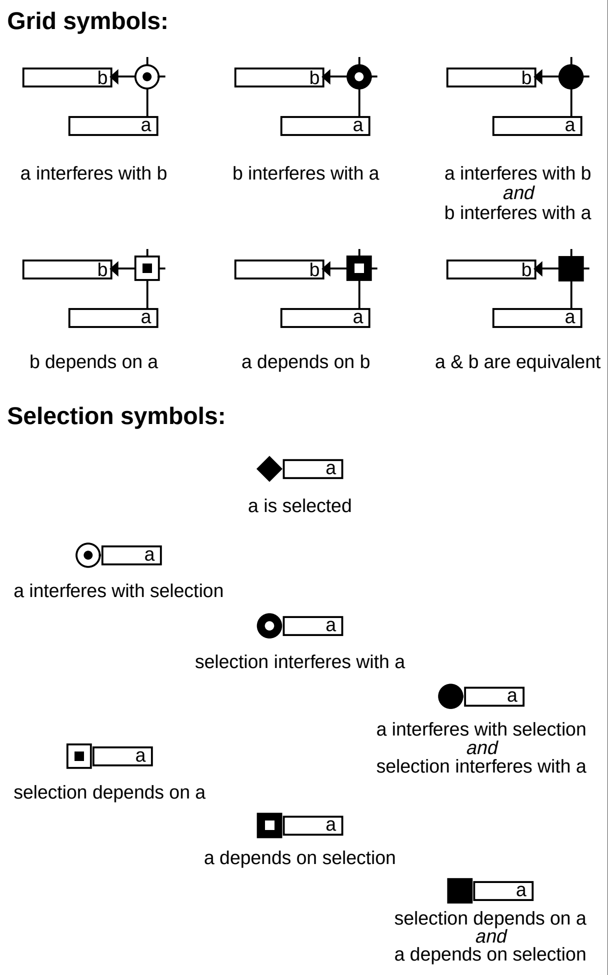
- additional interference symbols appear to the left of the nodes, indicating the relationship with the current selection.
- the display is controlled using the mechanisms described in Section 3.3. In particular, grouping only occurs within the same procedure.

4.2 Sorting variables

One of the key goals of the display was to sort the variables to simplify answering the maintenance question. In short, the answer is: "When making a change to a variable, *look right* to see what variables will interfere with the change, and *look up* to see what variables will be affected by the change." Unfortunately, the interference relation is not an order, so correct sorting turns out to be impossible in some cases. The solution we have adopted is to sort the variables according to strong dependence, which is almost a partial order (with the exception of equivalent variables), and then perform one pass of interchanges to correct for the interference relation where possible. This provides a display where the "look right, look up" rule works in most cases. The remaining cases of "backwards" interference are highlighted in the display.

4.3 Interference symbols

The design of the new display relies on a set of symbols that capture some of the interplay between the different relations that appear. The set we chose is illustrated in Figure 6. The two relations of interference and strong dependence are represented by circles and squares respec-



Grid symbols:
a interferes with b
b interferes with a
a interferes with b and b interferes with a
b depends on a
a depends on b
a & b are equivalent
Selection symbols:
a is selected
a interferes with selection
selection interferes with a
a interferes with selection and selection interferes with a
selection depends on a
a depends on selection
selection depends on a and a depends on selection

Figure 6: Legend for interpreting interference symbols.

tively. The square is meant to feel "stronger" than the circle. The forward and backward relations are represented by symbols which are "negatives" of each other, and add together to give the symbols for mutual relationships. The backwards and mutual cases have more black "ink", because they are important cases that need to stand out. The same set of symbols is used for the selection relationships, with the addition of the diamond to show the actual selection (the diamond shape is arbitrary, but easy to draw).

5. Examples

5.1 Word count

The display for a simple word count program was shown in Figure 5. We see that there is no "hidden" interference in this program — the strong dependence graph captures all of the action. The maximal variables (`main:nw`, `main:nl`, `main:nc`) can be found using the "look up" rule — no interference symbols appear above them in the display. A single variable, `main:inword`, has been selected for change, and we see that `main:nw` will need to be made changeable for this maintenance to be carried out, while the two equivalent variables (`main:c` and `Global:__iob`) will interfere with the change. The two equivalent variables appear as a single node in the graph (labelled "`global:__iob`"), but being in different procedures are not grouped in the new display. The cluster of "empty" variables at the bottom of the graph, shown by the node of size 3 with a thick border, have been excluded from the new display.

5.2 Weak dependence example

The new display for the weak dependence example described in Section 2.2 is shown in Figure 7. The display shows what little information there is about this program: all three variables are mutually interfering. Maintenance on the selected variable `main:v` will require a decision about the other two variables.

5.3 Variance

The strong dependence graph for a variance program was shown in Figure 1. The corresponding interference display is in Figure 8. This display has significantly more information than the graph. In particular, it reveals a pattern of interference between the six variables that appear maximal and unrelated at the top of the graph. For example, maintenance on `main:dev` will be interrupted by the others, particularly `t1` and `t2`.

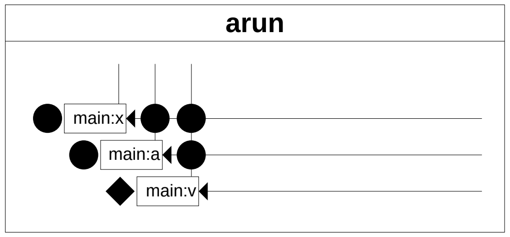


Figure 7: New display for the weak dependence example.

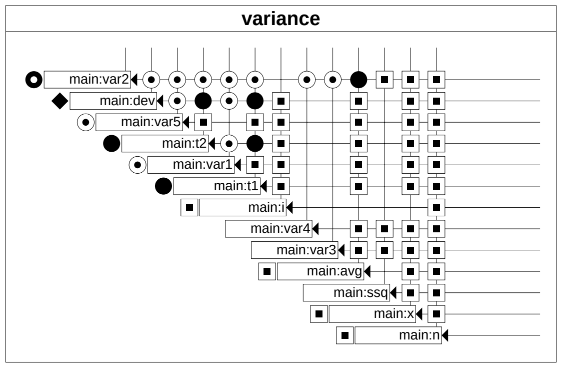
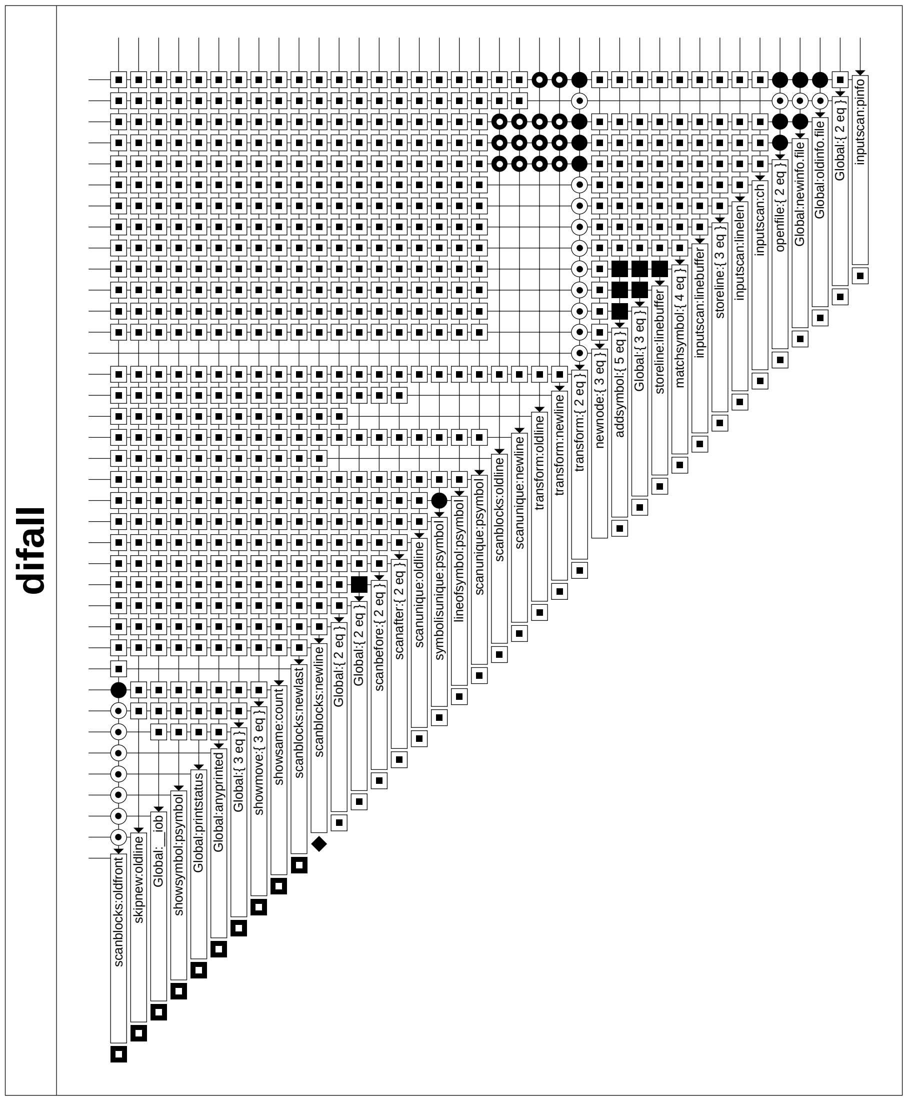


Figure 8: New display for the variance program.

5.4 Further examples

Three further examples are shown in Figure 9, Figure 10 and Figure 11. The "difall" program shows some "backwards" interference, where an appropriate sorting of the variables was impossible. It also illustrates grouping of equivalent variables in procedures. The large blocks of strong dependence correspond to a relatively "deep" strong dependence graph. It is difficult to separate individual variables for maintenance in this program. The "rl" example shows a complex pattern of interference and "backwards" interference, with relatively little strong dependence — maintainers beware! The node labelled "`Global:{ all }`" groups all of the global variables. The "pdevdwg" example gives some indication of scalability. The original program had over 300 variables, yet with grouping the display fits comfortably on a printed page. The sparse upper left area of the display shows a large set of maximal or near maximal variables that can be changed with relative ease. There is a large block of equivalent variables at the bottom that could cause some trouble. No variables have been selected in this example.



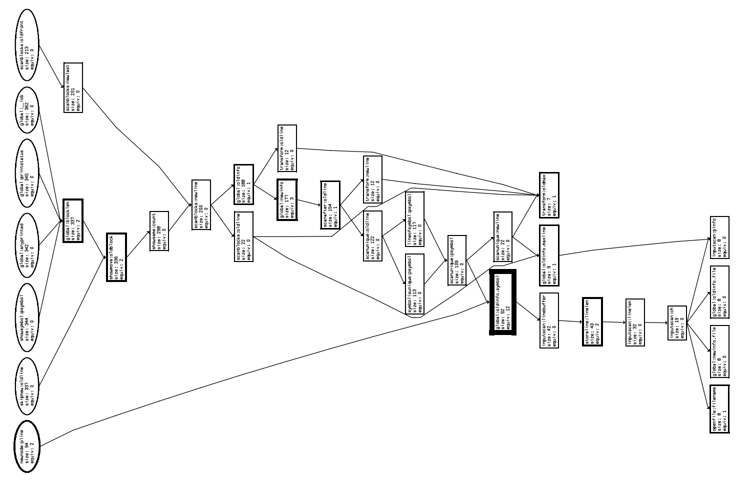


Figure 9: Strong dependence graph and new display for textual difference program.

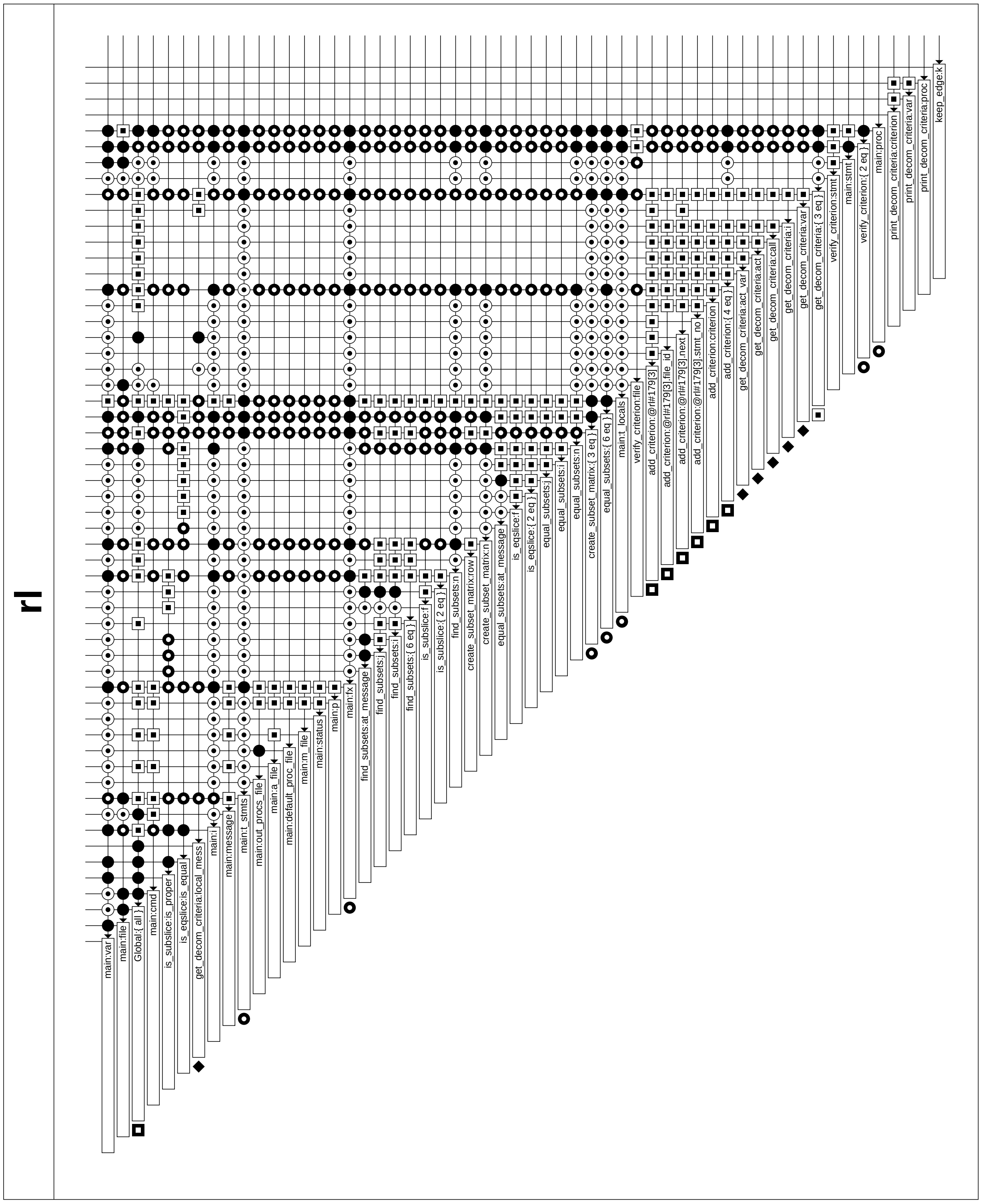


Figure 10: Display for “rl” program.

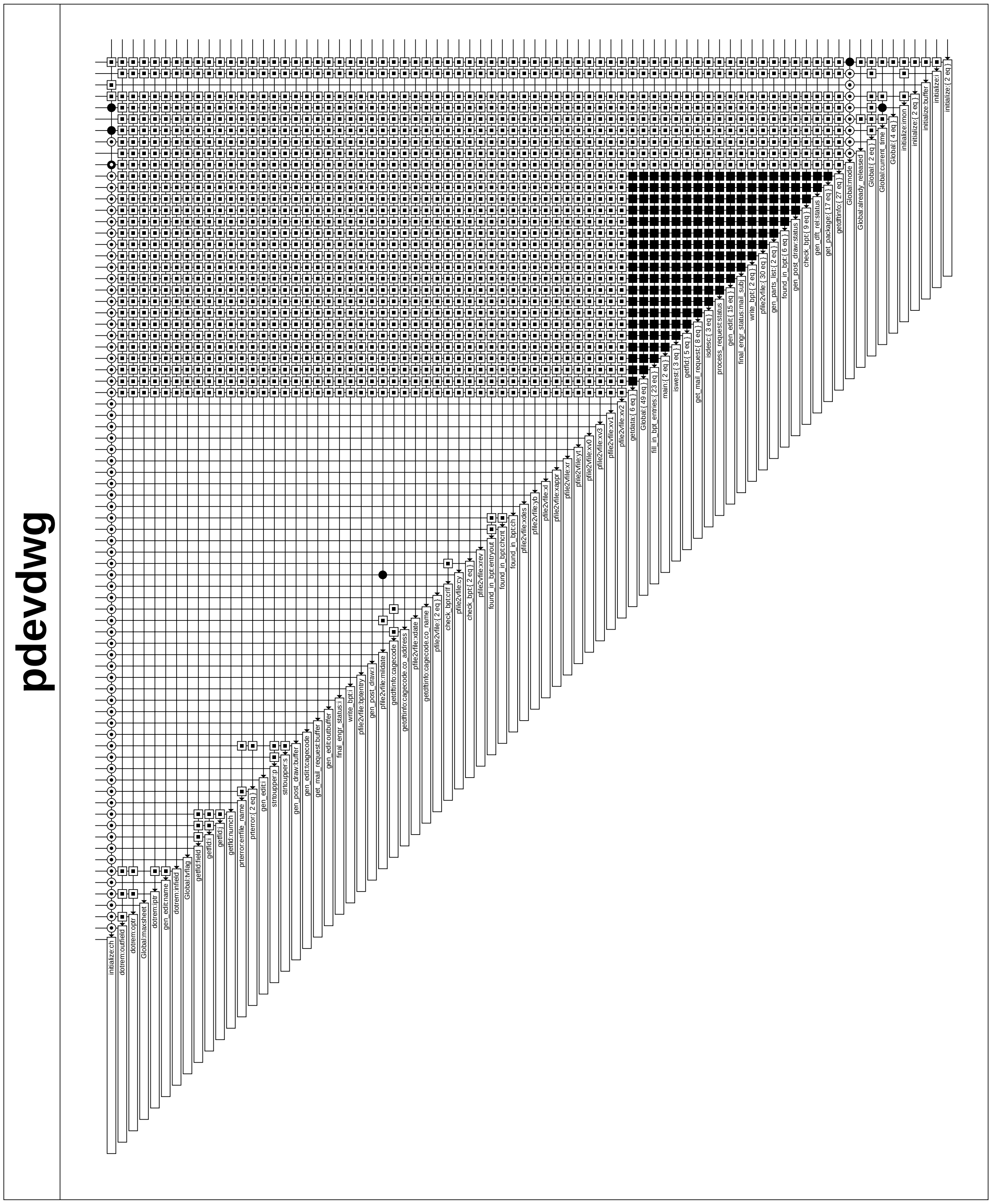


Figure 11: Display for “pdevdwg” program.

6. Present status, future work, conclusions

The interference display was designed as a case study in an investigation into the application of formal methods to display design. It is currently only implemented as a rough prototype with a command file interface that outputs Postscript files directly. The next step is to reimplement the display for full inclusion in the Surgeon's Assistant CASE tool. We have not yet performed any trials to establish the effectiveness of the display when used in a software maintenance environment. Real use would suggest changes to make the display more effective. The display has provided further insight into the decomposition slice approach to software maintenance, and several issues that have arisen as a result will be investigated further.

In conclusion, we have designed a new display that we hope will improve the effectiveness of visual impact analysis. The new display was designed in response to several weaknesses that were identified in the existing strong dependence graph display by a systematic and formal analysis of the display semantics. Requirements for the new display were derived by concentrating directly on the maintenance task and the most important question that the display needed to answer to assist with that task. As a result, the new display provides a model of interpretation based on a simple "look right, look up" rule. Several examples have shown that the new display captures more information than the strong dependence graph, and can scale up to reasonably large problems.

References

[1] D. BINKLEY AND K. GALLAGHER. A survey of program slicing. In M. Zelkowitz, editor, *Advances in Computers*. Academic Press, 1996.

[2] KEITH B. GALLAGHER. Visual impact analysis. In proceedings of the Conference on Software Maintenance, 1996.

[3] K. B. GALLAGHER AND J. R. LYLE. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, vol. 17, no. 8, August 1991. Pages 751–761.

[4] KEITH GALLAGHER AND LIAM O'BRIEN. Reducing visualization complexity using decomposition slices. In proceedings of the 1997 Software Visualization Workshop, SoftVis97. December 1997.

[5] I. LEMKE AND G. SANDER. *Visualization of Compiler Graphs: Design Report and Documentation*. Universitat des Saarlandes, Saarbrucken, Germany, May 1994.

[6] F. TIP. A survey of programming slicing techniques. *Journal Of Programming Languages*, vol. 13, no. 3, 1995. Pages 121–189.

[7] M. WEISER. Program slicing. *IEEE Transactions on Software Engineering*, vol. 10, July 1984. Pages 352–357.