Wisconsin Program-Integration System 2.0
Reference Manual

University of Wisconsin
Madison, WI

WPIS 2.0 Reference Manual

This work was supported in part by a David and Lucile Packard Fellowship for Science and Engineering, by the National Science Foundation under grants DCR-8552602, DCR-8603356, CCR-8958530, and CCR-9100424, by the Defense Advanced Research Projects Agency under ARPA Order No. 6378 and ARPA Order No. 8856 (monitored by the Officeof Naval Research under contracts N00014-88-K-0590 and N00014-92-J-1937, respectively), as well as by grants from Cray Research Foundation, DEC, Eastman Kodak, HP, IBM, 3M, and Xerox.

Contents

CHAPTER 1
Introduction and Background **1**
1.1 Introduction 1
1.2 Background 2
1.2.1 Language 2
1.2.2 System Dependence Graph 3
1.2.3 Systems and Compilation-units 4
1.2.4 Editor Tags 4
1.3 Invoking the Wisconsin Program-Integration System 5
1.3.1 Environment 5
1.3.2 The WPIS Command 6
1.4 Changes Since Release 1.0 6

CHAPTER 2
Integration Tool Commands **7**
2.1 Program Slicing 7
2.2 Language-Based Differencing 9
2.3 Integration 11
2.3.1 Integration Records 11
2.3.2 Illustrating interferences 12
2.3.3 Affected Points 14
2.4 Cross Referencing 16
2.5 Displaying Graph Edges 16
2.6 Debugging Features 17

CHAPTER 3
Non-standard Features of the WPIS Editor **21**
3.1 Compilation-Unit Buffers 21
3.2 Files 22

Index **27**

CHAPTER 1

Introduction and Background

1.1 Introduction

The Wisconsin Program-Integration System is a prototype language-based system that provides support for several important software-engineering tasks, such as

- understanding what an existing program does and how it works,
- understanding the differences between several versions of a program,
- creating new programs by combining pieces of old programs.

At the heart of the system is a package for manipulating *program dependence graphs* [2, 5, 9] and an extension of program dependence graphs, called *system dependence graphs* [8].

The primary operations that the system provides fall into three different classes, depending on whether one program, two programs, or three (or more) programs are involved.

Slicing ***(one program)***
: Slicing operations help the user to gain an understanding of how elements in a given program interact.

Differencing ***(two programs)***
: Differencing operations help the user to gain an understanding of how one program differs from another [3].

Integration ***(three or more programs)***
: The integration operation lets the user merge different variants of a base program into a version that exhibits the changes that were made

in each of the variants [5]. If integration fails, the system incorporates a facility to help interactively diagnose the cause of interference [14].

This document describes the basic command set of the Wisconsin Program-Integration System. An overview of the system's theoretical underpinnings can be found in [7]. A tutorial introduction to an earlier version of the system can be found in [13].

The user interface for the integration system incorporates a language-specific editor created using the Synthesizer Generator, a meta-system for creating interactive, language-based program-development systems [15, 16] As with all editors created with the Synthesizer Generator, the integration system's editor exhibits characteristics that are specific to the language being edited, while at the same time sharing the generic user interface described in Chapter 3 of The Synthesizer Generator Reference Manual [16].

This document primarily describes the commands that are specific to the integration system—for example, for invoking the system's operations for **slicing**, **differencing**, and **integration**—as opposed to commands that are part of the standard user interface of editors created with the Synthesizer Generator. A few of the commands that are part of the standard user interface of Synthesizer-Generator-created editors are also described because these commands have been altered to function in non-standard ways.

1.2 Background

1.2.1 Language

The language supported by the integration system is a declaration-free imperative language with the following constructs:

- Assignment statements
- Conditional statements
- While loops
- For loops
- Output statements
- Procedure declarations and call statements.

Parameters are passed by reference; recursion (and mutual recursion) is permitted; nested procedure declarations are not supported (i.e., as in C, only external procedure declarations are permitted). The language supports both scalar variables and array variables. Any variable that is not a formal parameter and has a name starting with the letter g is treated as a global variable. (Extensions to handle a full-fledged programming language are on-going.)

For historical reasons, we use the term *system* where most others would use *program*. A *system* is made up of *compilation-units*, which are in turn made up of procedures. Each editing buffer contains one compilation-unit.

1.2.2 System Dependence Graph

Control-flow analysis of procedures is carried out using the Synthesizer Generator's facility for evaluating and maintaining the values of attributes. The Synthesizer Generator's external store facility [16] is used to pass this information to an external package written in C that builds and maintains control flow graphs and call graphs. A control flow graph is maintained for each procedure. Array data dependences are determined with the Omega Test [11, 12]. The *system dependence graph* for a program is constructed on demand, at the time a command needing the dependence graph is invoked: the program's procedure dependence graphs are first constructed from their corresponding control flow graphs; the system dependence graph proper is then constructed by linking together all of the program's procedure dependence graphs with *call-graph* information and introducing some additional information that summarizes transitive dependences due to procedure calls [8].

Commands added to the editor make use of *system dependence graphs* to perform various actions, as documented in the other sections of this manual. For example, the integration command invokes the integration algorithm on three system dependence graphs, reports whether the variant programs interfere, and, if there is no interference, builds the integrated program. The Synthesizer Generator's facility for displaying program components in different display styles is used to report information back to the user of the integration system. The various commands use different colors and/or different type-faces (depending on how the display styles are defined) to indicate various collections of program components, for example, all of the components in a slice of a program, etc.

When a program's *system dependence graph* is constructed, the program is checked to make sure it meets certain consistency requirements. In particular, an error is reported (and the *system dependence graph* is not built) if any of the following conditions hold:

- A procedure is called but not declared.
- Two or more procedures are declared with the same name.
- Two or more formal parameters of some procedure are declared with the same name.
- There is a mismatch between the number of formal parameters in a procedure declaration and the number of actuals at some call site.

If none of these conditions hold, a system is said to be *consistent*.

Due to calls of the form p(g) and q(g, g) , where g is a global variable, it is possible for a formal parameter to be an alias of a global variable and for two formal parameters to be *aliases* of each other. The *system dependence graph* constructed for a program that contains such calls contains dependence edges that safely account for dependences due to aliasing. The **slicing** and **differencing** operations correctly account for aliasing effects; however, in the current implementation, the result of an **integration** operation is not guaranteed to be correct if aliasing occurs in one of the programs that is an argument of the integration operation.

1.2.3 Systems and Compilation-units

Systems are organized in compilation-units, which are groupings of the system's procedures. During an editing session, a system being edited has its compilation-units associated with buffers (see Section titled Compilation Units Buffers). Between editing sessions, systems can be saved and restored. The system is written to a directory of the same name; each compilation-unit is written to a file in that directory. A system can always be reread into the editor that wrote it. The file commands of the Synthesizer Generator have been modified to preserve the integrity and consistency of systems (see Section titled Files for the description of these changes).

1.2.4 Editor Tags

So that corresponding components can be identified in different versions of a program, the program-integration system's editor automatically supplies tags on program components (i.e., assignment statements, call statements, predicates, formal parameters, and actual parameters). These tags are normally hidden from the user, but can be displayed using the alternate-unparsing-toggle command. The tags allow the editor to keep track of the migration of components as a new program version is edited. The tags are manipulated according to the following principles:

- When a copy of a program is made, each component in the copy is given the same tag as the corresponding component in the original program.
- A newly inserted component is given a previously unused tag; the tag of a component that is deleted is never re-used; a component that is moved from one position to another retains its tag.
- The tags on components persist across different editing sessions and machines.
- Tags are allocated by a single server, so that two different editors cannot allocate the same new tag.

The tags establish a component-correspondence relation on the components of different versions of a program. This furnishes the means for establishing how vertices of different *system dependence graphs* correspond.

1.3 Invoking the Wisconsin Program-Integration System

1.3.1 Environment

The value of variable INTEGRATOR_LOC of the UNIX environment is used to specify the directory in which the integration system resides. Thus, the setenv command should first be used to set the value of environment variable INTEGRATOR_LOC to the name of the appropriate directory:

```
setenv INTEGRATOR_LOC integration-system-directory
```

To carry out the rest of the necessary initialization actions, if the display is a color display the following command should be performed:

```
source $INTEGRATOR_LOC/bin/setenv.color
```

else if the display is a monochrome display the following command should be performed:

```
source $INTEGRATOR_LOC/bin/setenv.bw
```

else if the display is a ncd display the following command should be performed:

```
source $INTEGRATOR_LOC/bin/setenv.ncd
```

(These two commands would ordinarily be placed in a user's .cshrc file.) This command will have the following effect:

- The environment variable TAG_FILE_NAME is set to
  $INTEGRATOR_LOC/etc/UniqueTAG
  which contains the counter that the integration system uses for supplying tags on program components.
- The X11 resources for the editor are defined (see The Synthesizer Generator Reference Manual [16] for more information in setting X resources and how to customize an editor).

By using the InterNet and X11 to connect to a remote host, it is possible to run the integration system remotely. For example, suppose that the remote host is x.cs.remote.edu and that one is using display device v.w (If y.cs.local.edu is a workstation, the display device will ordinarily be 0.0) .

To complete the setup on y.cs.local.edu, execute the command

```
xhost +x.cs.remote.edu
```

The final steps are to login to x.cs.remote.edu from some xterm window on y.cs.local.edu:v.w, execute the command

```
setenv DISPLAY y.cs.local.edu:v.w
```

and then invoke the copy of the integration system that exists on x.cs.remote.edu.

1.3.2 The WPIS Command

The integration system is known as wpis. Command-line arguments are a list of one or more system names (directory names) together with various command-line flags. When wpis is invoked, a system named *tmp* is created with a compilation-unit named main. If wpis is invoked wihtout any system names as arguments, editing begins in the buffer for compilation-unit main of *tmp*.

If wpis is supplied with one or more system names as arguments, then a single window is created and **open** is executed in that window on each argument in turn. In this case editing begins in one of the compilation-units of the last system whose name was a command line argument. If this system has a compilation-unit named main, this will be the initial editing buffer presented to the user. (Note that the compilation-unit named main is not necessarily one containing a procedure named *main*.)

Wpis can also be used as a batch tool by invoking it in batch mode with the -b command-line option. When invoked in this fashion, no windows appear. Batch mode is only useful when the user supplies an editor script.

The editor automatically loads the command script from the file .synrc in the users $HOME directory. Other scripts may be loaded and interpreted by giving the command-line option -l *script1* ... -l *scriptn*, where files *script1,...,scriptn* contain the scripts to be interpreted. Scripts may also be used in this fashion when wpis is invoked in interactive mode.

1.4 Changes Since Release 1.0

Release 2.0 of the Wisconsin Program-Integration System incorporates the new features and user-interface of the Synthesizer Generator release 4.0. Some of these new features are:

- Use of dialog boxes to enter command arguments.
- Invocation of the editor with scripts or in batch mode.
- On-line help documentation.

Some editor commands of the Synthesizer Generator have been modified to preserve the integrity and consistency of systems (see Chapter 3 for the description of these changes).

In addition, some improvements to the tool includes:

- Array data dependence analysis performed with the Omega test [11, 12].
- Extension of the language to comments.
- The result of an integration operation can now consist of several compilation-units.

CHAPTER 2
Integration Tool Commands

All commands in this chapter operate on *consistent* systems and will return an error if one of the system used to perform the command is not consistent.

2.1 Program Slicing

The integration system includes commands to display the components of program slices [10, 5, 3]. A special display style (with a distinctive color and/or type-face) is used to distinguish the components of a slice from other components of the program. Components of backward slices are displayed in BACKWARD_SLICE style; components of forward slices are displayed in FORWARD_SLICE style.

To perform a slicing operation, first change the current selection to the program point or points with respect to which the slicing operation is to be performed; then invoke the appropriate slicing command. Each system has associated *forward* and *backward slice sets*—sets of components with respect to which forward and backward slices are taken, respectively. Initially, both of these sets are empty; each slicing operation first changes the forward or backward slice set of the current system; the display is then updated so that program components in the backward slice with respect to the backward slice set are displayed in the BACKWARD_SLICE display style, and components of the program in the forward slice with respect to the forward slice set are displayed in the FORWARD_SLICE display style.

Every formal parameter in a procedure header and every actual parameter in a procedure call is represented in the system dependence graph using two vertices: a *parameter-in* vertex and a *parameter-out* vertex. When either of these vertices is included in a slice, the corresponding text is displayed in the appropriate slice style. To permit the user to tell which of the two vertices is in the slice, the textual display is annotated with *:in, :out, or :in,out.*

slice
: Set the current system's backward slice set to the collection of program components contained within the current selection. Change the display to indicate the components of the backward slice of the program with respect to the backward slice set. This command also brings up a new window that displays the current compilation-unit in the view SliceView, which hides all program components that are not part of the slice.

slice-augment
: Augment the current system's backward slice set with the collection of program components contained within the current selection. Change the display to indicate the components of the backward slice of the program with respect to the augmented backward slice set.

unslice
: Discard all program components contained within the current selection of the current system from the system's backward slice set. Change the display to indicate the components of the backward slice of the program with respect to the remaining elements in the backward slice set.

forward-slice
: Set the current system's forward slice set to the collection of program components contained within the current selection. Change the display to indicate the components of the forward slice of the program with respect to the forward slice set.

forward-slice-augment
: Augment the current system's forward slice set with the collection of program components contained within the current selection. Change the display to indicate the components of the forward slice of the program with respect to the augmented forward slice set.

forward-unslice
: Discard all program components contained within the current selection of the current system from the system's forward slice set. Change the display to indicate the components of the forward slice of the program with respect to the remaining elements in the forward slice set.

unslice-system
: Discard the forward and backward slice sets of the current system and remove all slicing information from the display of all windows containing parts of the system.

unslice-all-systems
: Discard the forward and backward slice sets of all systems and remove all slicing information from the display of all windows.

set-slice-parameters

Set-slice-parameters pops up a dialog box containing radio buttons for each slice parameter, with the current value of each parameter highlighted. Clicking SELECT on radio buttons changes proposed values for parameters; these values do not take effect until SELECT is clicked on Start. Clicking SELECT on Cancel pops down the dialog box without resetting the slice parameters. The slices parameters are as follows:

- At parameter vertices
  Specifies what it means to select a parameter at a procedure declaration or call site. Because each formal or actual parameter can correspond to two vertices in the system dependence graph: the in vertex which represents the actions taken to pass information to the call, or the out vertex, which represents the actions taken to return information to the caller. Consequently, it is ambiguous what the selection of a parameter indicates. By clicking on the appropriate button the user indicates that when a slice operation is invoked and the current selection is a parameter, the selection represents either the in vertex, the out vertex, or both.
- Slice phase
  Choice of whether the next slicing operation invoked is to perform a phase I slice, a phase II slice, or both (i.e., a phase I slice followed by a phase II slice). (See [8] for a description of the different phases of the interprocedural slicing algorithm.) The Phase I button indicates that either a *b1* slice (backward phase I slice) or an *f1* slice (forward phase I slice) should be performed; the Phase II button indicates that either a *b2* slice (backward phase II slice) or an *f2* slice (forward phase II slice) should be performed.

2.2 Language-Based Differencing

The system includes commands to compute textual and semantic differences of two variants of a program. Differencing operations are not symmetric and hence the results are displayed in only one of the variants. The commands for language-based differencing are included in the *LDiff* menu .

A special display style with a distinctive color and/or type-face is used to distinguish the textual and semantic differences of one program with respect to another. Semantically different components are displayed using the SEMANTIC_CHANGE style. In addition, for the differencing commands that use comparison techniques based on isomorphism with renaming [6], semantically identical but textually different components are displayed using the TEXTUAL_CHANGE style.

difference-by-vertex ***old-sys proc new-sys proc***

Display a safe approximation to the set of semantically changed components of the new procedure (*new-sys, proc*) with respect to the old one (*old-sys, proc*). The correspondence between the two procedures, based on b2-slice isomorphism, minimizes the number of semantically changed components.

The dialog box for **difference-by-vertex** includes a system browser that can be used to obtain the *old-sys* and *new-sys* parameters. Default values from the current selection are supplied for *new-sys* and *proc*.

The programmatic interface to this command is:
Difference_by_vertex(STR *old-sys*, STR *proc*, STR *new-sys*, STR *proc*)

difference-by-vertex-and-edge ***old-sys proc new-sys proc***

Display a safe approximation to the set of semantically changed components of the new procedure (*new-sys, proc*) with respect to the old one (*old-sys, proc*). The correspondence between the two procedures, based on b2-slice isomorphism, minimizes the number of semantically changed components plus the number of new flow or control edges.

The dialog box for **difference-by-vertex-and-edge** includes a system browser that can be used to obtain the *old-sys* and *new-sys* parameters. Default values from the current selection are supplied for *new-sys* and *proc*.

The programmatic interface to this command is:
Difference_by_vertex_and_edge(STR *old-sys*, STR *proc*, STR *new-sys*, STR *proc*)

clear-difference

Remove all information obtained from any of language-based-differencing operations (as well as any other special annotations on program components, such as slicing information) from the procedure containing the current selection.

slice-isomorphism ***old-sys proc vertex new-sys proc vertex***

Tests if the two component's backward-phase-2 slices are isomorphic, with the selected criterion for slice isomorphism. This command requires a system-name, procedure-name, and vertex-number for each of the two components that are being compared.

The dialog box for **slice-isomorphism** includes a system browser that can be used to obtain the *old-sys* and *new-sys* parameters. Vertex numbers can be determined by changing the view of the appropriate buffer to TagView. Default values from the current selection are supplied for *new-sys, proc, vertex*.

The programmatic interface to this command is:
Slice_isomorphism(STR *old-sys*, STR *proc*, STR *vertex*, STR *new-sys*, STR *proc*, STR *vertex*)

set-slice-isomorphism

Set-slice-isomorphism pops up a dialog box containing radio buttons for various slice-isomorphism criteria with the current value highlighted. Clicking SELECT on a radio button changes the proposed isomorphism criterion; this value does not take effect unless SELECT is clicked on Start. Clicking SELECT on Cancel pops down the dialog box box without resetting the slice-isomorphism criterion.

The programmatic interface to this command is:
Set_slice_isomorphism(BOOL *renam*) where *renam* is *true* or *false* whether the isomophism is with renaming of variables or not.

Note: For related operations that also display the differences between two programs, see the command **display-affected** , which is described in Section 2.3.3.

2.3 Integration

Integration allows the user to merge different variants of a base system into a single version that captures the changes that were made in each of the variants. When integration is successful, the result will be placed in a new system (the name of which is supplied by the user as a parameter to the integration command) which becomes the current system.

Integration can fail for several reasons, and several facilities are available to help the user diagnose the cause. For example, a trace of the results of various intermediate stages of the integration process is provided in a buffer named *status*, which is available for perusal by the user. For several types of failure, a set of directly affected components is displayed in one of the arguments. For other types of failure, there is an interactive facility to help diagnose the cause (see Subsection 2.3.2.)

The integration command is included in the *Integrate* menu.

integrate *Base VariantA VariantB Result*

Integrate *VariantA* and *VariantB* with respect to *Base* and place the result in *Result*. The dialog box for **integrate** includes a system browser that can be used to obtain the *Base, VariantA*, and *VariantB* parameters. The default value supplied for the result system is *INTEGRATED*.

The programmatic interface to this command is:
Integrate(STR *Base*, STR *VariantA*, STR *VariantB*, STR *Result*)

2.3.1 Integration Records

For each integrate command initiated by the user, a record is created containing information about the systems that were the arguments to the command as well as the results of the operation. This record is discarded as soon as any editing operations are performed on any of the systems that were the arguments to the integration operation. **Show-integration-record** allows the user to sequentially step through the list of these records. This allows the user to inspect the results of previous integrations performed during a session with the integration system. (Integration records are not saved between sessions.) If a particular invocation of the integration operation did not succeed, it is possible to use these facilities for finding information about the cause of interference.

show-integration-record

Pops up a dialog box showing the names of the base system, the variant systems, and the result system of the current integration record. (The current integration record is either that of the most recent invocation of the integration operation, or the integration record set by the most recent invocation of this command.) The dialog box contains the following action buttons:

- newest
  Show the names for the base system, the variant systems, and the result system (if any) from the most recent invocation of the integration operation.
- older
  Sequentially step through older integration records (showing the names for the base system, variant systems, and result system) until the oldest integration record is reached.
- newer
  Sequentially step through newer integration records (showing the names for the base system, variant systems, and result system) until the most recent integration record is reached.
- Done
  Pops down the dialog box, leaving the current integration-record set for next invocation of the command.

2.3.2 Illustrating interferences

The command **show-interference** helps the user diagnose the cause of interference when integration fails due to Type I interference [1, 4]. A description of the analysis performed in conjunction with this command can be found in [14]. If integration is unsuccessful because of Type I interference, the action buttons in the *show-interference* dialog box permit the user to navigate among the list of *interference vertices*. Suppose that the integration command that failed attempted to integrate system A and system B with respect to system Base. The buttons First, Next, and Previous allow you to step through the list of all *interference vertices*. When these buttons are pressed, the selections in the buffers for system A and system B are both repositioned at the current interference vertex. These commands also cause a second dialog box to pop up, the *edge-selection* dialog box.

The *edge-selection* dialog box permits the user to display the symmetric difference of the edges incident on the current interference vertex in the *system dependence graphs* associated with system A and system B. Two special display styles with distinctive colors and/or type-faces are used to display these edges: to distinguish edge sources from edge targets, sources are displayed in CULPRIT_TAIL style whereas targets are displayed in CULPRIT_HEAD style.

show-interference

Pops up a dialog box that allows the user to navigate through the list of *interference vertices* [14]. The dialog box contains the following action buttons (in what follows, suppose that the command **integrate** that failed attempted to integrate system A and system B with respect to system Base):

- first
  Move the selections of the buffers associated with system A and system B to the first interference vertex in the list of interference vertices and pop up an *edge-selection* dialog box.
- next
  Move the selections of the buffers associated with system A and system B to the next interference vertex in the list of interference vertices and pop up an *edge-selection* dialog box.
- previous
  Move the selections of the buffers associated with system A and system B to the previous interference vertex in the list of interference vertices and pop up an *edge-selection* dialog box.
- all
  Use the CULPRIT_TAIL style to display simultaneously all interference vertices in the buffers associated with system A and system B .
- clear
  Remove from the display all information about the symmetric difference of the edges that are incident on the current interference vertex. (That is, the components that were displayed in the changed color and/or type-face are changed back to the normal color and type-face.)
- Done
  Pop down the *show-interference* and *edge-selection* dialog boxes and change all the components that were displayed in the changed color and/or type-face back to the normal color and type-face.

The *edge-selection* dialog box contains radio buttons for the user to choose which of the different classes of dependence edges to display. The eight radio buttons refer to loop-independent flow edges (Lif), loop-carried flow edges (Lcf), control edges (Con), def-order edges (Do), summary edges (Sum), call edges (Call), parameter-in edges (Pin), and parameter-out edges (Pout), respectively. (See [8] for the definitions of the kinds of dependences that these edges represent.) An edge is only displayed if it is in the symmetric difference of the edges incident on the current interference vertex and if it is in one of the chosen edge classes. A * appears next to the edge class names for which there are such edges. The *edge-selection* dialog box contains the following action buttons:

- All
  Set the radio button for every edge class to Yes, and show all edges in the symmetric difference of the current interference vertex.
- None
  Set the radio button for every edge class to No,
- Show
  Display edges that are in the symmetric difference of the edges incident on the current interference vertex and also in one of the chosen edge classes.
- Done
  Pop down the *edge-selection* dialog box and clear the display of information about the edges incident on *interference vertices.*

2.3.3 Affected Points

The *Integrate* menu also includes commands to display the results of various computations that are intermediate steps of program integration. When these commands are invoked, a special display style with a distinctive color and/or type-face is used to distinguish the relevant components from other components of the program. The type of components are: directly affected, strongly affected, affected, preserved, delta, preserved+affected, and preserved+delta (See [1] for the definitions of these concepts).

display-affected *Base Variant Display*

If *Display* is DAP, use DIRECTLY_AFFECTED_POINTS style to display the components of the *Variant* system that are directly affected with respect to the *Base* system.

If *Display* is SAP, use STRONGLY_AFFECTED_POINTS style to display the components of the *Variant* system that are strongly affected with respect to the *Base* system.

If *Display* is AP, use AFFECTED_POINTS style to display the components of the the *Variant* system that are affected with respect to the *Base* system.

If *Display* is Delta, use DELTA_POINTS style to display the components of the *Variant* system that are in the delta with respect to the *Base* system.

The dialog box for **display-affected** includes a system browser that can be used to obtain the *Base* and *Variant* parameters. The default value supplied for the *Variant* system is the current system.

The programmatic interface to this command is:
Display_affected(STR *Base*, STR *Variant*, INT *Display*) where Display is 1 for DAP, 2 for SAP, 3 for AP, and 4 for Delta.

display-preserved *Base VariantA VariantB Display*

If *Display* is PP, use PRESERVED_POINTS style to display the components of the the *Base* system that are preserved with respect to the *VariantA* and *VariantB* systems.

If *Display* is PP_AP, display the preserved components of the three systems in the *Base* system; display the components of *VariantA* system that are affected with respect to *Base* system; display the components of *VariantB* system that are affected with respect to *Base* system.

If *Display* is PP_Delta, display the preserved components of the three systems in the *Base* system; display the components of *VariantA* system that are in delta with respect to *Base* system; display the components of *VariantB* system that are in delta with respect to *Base* system.

The dialog box for **display-preserved** includes a system browser that can be used to obtain the *Base, VariantA*, and *VariantB* parameters. The default value supplied for the *VariantB* system is the current system.

The programmatic interface to this command is:
Display_preserved(STR *Base*, STR *VariantA*, STR *VariantB*, INT *Display*) where Display is 4 for PP, 5 for PP_AP and 6 for PP_Delta.

clear-system

Remove all information obtained from any of the **display-affected** or **display-preserved** computations (as well as any other special annotations on program components, such as slicing information) from all buffers of the current system.

clear-all-systems

Remove all information obtained from any of the **display-affected** or **display-preserved** computations (as well as any other special annotations on program components, such as slicing information) from all buffers of all systems.

Note that the results displayed by the commands **display-preserved** and **display-affected** are similar to the results displayed by some of the commands for language-based program differencing that are discussed in Section 2.2.

2.4 Cross Referencing

A limited form of navigation via cross referencing is provided. The editor of the integration system allows the user to search for uses of procedure names via the command **search-cross-references** in the *Structure* menu.

search-cross-references

Pop up a dialog box in which the user can enter a procedure name. A procedure browser on the procedures of the current system can be used to select the procedure name. The dialog box contains action buttons that cause the selection to be changed to an occurence of this name.

The dialog box includes the following action buttons:

- First
  Change the current selection to be the first occurrence of the procedure name listed in the dialog box.
- Next
  Change the current selection to be the next occurrence of the procedure name listed in the dialog box.
- Previous
  Change the current selection to be the previous occurrence of the procedure name listed in the dialog box.
- Done
  Pop down the dialog box, leaving the procedure name set for the next invocation of the command.

2.5 Displaying Graph Edges

The integration system includes the command **show-edges** for displaying edges of a program's system dependence graph, and commands **forward-control-true-edge**, **forward-control-false-edge** for traversing the control edges of a program's control flow graph. These commands are included in the *show* menu. As with the slicing operations, the display of edges in a system dependence graph is initiated by first selecting the program component or components whose graph edges are to be displayed, and then invoking the command. It is possible to display either incoming edges (i.e., edges whose targets are the selected components) or outgoing edges (i.e., edges whose sources are the selected components).

Special display styles with distinctive colors and/or type-faces are used to distinguish the head and tail of each edge that belongs to the set of edges requested. The styles SOURCE_HEAD, SOURCE_TAIL, TARGET_HEAD, and TARGET_TAIL are used to distinguish edge components from other program text.

show-edges

Show-edges presents a dialog box containing radio buttons for parameter selection and action buttons to display edges. Current values of parameters are highlighted. The eight radio buttons for edge classes refer to loop-independent flow edges (Lif), loop-carried flow edges (Lcf), control edges (Con), def-order edges (Do), summary edges (Sum), call edges (Call), parameter-in edges (Pin), and parameter-out edges (Pout), respectively. (See [8] for the definitions of the kinds of dependences that these edges represent.) The edge class choices are the same as those provided in the *edge-selection* dialog box for the **show-interference** command. An edge is only displayed if it is in one of the chosen edge classes. A * appears next to the edge class names for which there are such edges. Parameter vertices are selected with the In, Out, and In Out radio buttons in the same way as in the dialog box for the **set-slice-parameters** command. Display of incoming or outgoing edges is selected with the Source and Target radio button. The dialog box contains the following action buttons:

- All
  Set the radio button for every edge class to Yes, and display all edges of the current selection.
- None
  Set the radio button for every edge class to No, and clear the display of edge-display information.
- Show
  Display edges of the chosen edge classes for the current selection.
- Done
  Pops down the dialog box, leaving the parameters set for the next invocation of the command.

forward-control-true-edge
Move the selection to the control-true program point.

forward-control-false-edge
Move the selection to the control-false program point.

2.6 Debugging Features

The integration system incorporates several debugging commands that can be used to provide information about internal data structures as well as to perform some test computations. These commands are only available when the integration system is built with the appropriate compile-time debug flags set. See the document titled WPIS Installation Instructions for instructions on how to build a version of the integration system with these features.

Unless otherwise noted, the debugging commands take their arguments from the most recent selection made in an editing buffer and send their output to an appropriately named text buffer. When displaying a control flow graph (*cfg*) or a procedure dependence graph (*pdg*), there are numerous fields that can be displayed at the request of the user. The user can choose the fields of interest with the commands **set-cfg-fields** and **set-pdg-fields**.

The commands for debugging are included in the *Show* menu.

global-cfg-vertex-store
: Display a list of all control-flow-graph vertices currently in the global external store.

show-cfg
: Display the vertices of the control flow graph for the selected procedure. This command does not force data flow analysis to be performed, nor will the results of data flow analysis be displayed if it has already been performed. The name of the output text buffer is constructed from the system-id and the procedure-id of the control flow graph.

show-augmented-cfg
: Display the vertices of the control flow graph of the selected procedure, including the results from data flow analysis.

show-system-cfgs
: Display the control flow graphs for all the procedures of the selected system.

show-call-graph
: List the vertices and edges of the call graph for the selected system. Output is directed to the tty from which wpis was invoked.

show-system-ag
: List information about the *linkage grammar* for the selected system; the grammar is used to compute the interprocedural summary edges of the program's system dependence graph [8]. Output is directed to the tty from which wpis was invoked.

show-pdg
: Display the procedure dependence graph for the selected procedure.

show-system-pdgs
: Display the procedure dependence graphs for all the procedures of the selected system.

test-reconstitute *New*
: Reconstitute the system dependence graph associated with the current system and place the resulting system in the *New* system. The *New* system defaults to *test* .

show-merged-system
: Display the procedure dependence graphs of the system dependence graph that results from the merge step of the most recent invocation of the integration algorithm (if such a system dependence graph exists).

show-merged-ordered-system
: Display the procedure dependence graphs of the system dependence

graph that results from the merge step of the most recent invocation of the integration algorithm (if such a system dependence graph exists). The vertices that correspond to program statements within a region are ordered as they will appear in the reconstituted system.

set-cfg-fields

Set-cfg-fields pops up a dialog box containing radio buttons for each *cfg* fields, with the current value of each field highlighted. Clicking SELECT on radio buttons changes proposed values for fields; these values do not take effect until SELECT is clicked on Start. Clicking SELECT on Cancel pops down the dialog box without resetting the *cfg* fields.

set-pdg-fields

Set-pdg-fields pops up a dialog box containing radio buttons for each *pdg* fields, with the current value of each field highlighted. Clicking SELECT on radio buttons changes proposed values for fields; these values do not take effect until SELECT is clicked on Start. Clicking SELECT on Cancel pops down the dialog box without resetting the *pdg* fields.

read-sdg *System*

Read in *System* stored in system dependence graph format. The dialog box for **read-sdg** includes a list of systems that can be used to obtain the *System* parameter.

write-sdg

Write the current system, in system dependence graph format, to its associated directory. An error is returned if the system doesn't have an associated directory or if it has been modified since the last **save** command has been performed.

CHAPTER 3

Non-standard Features of the WPIS Editor

This chapter describes the modifications that were made to certain Synthesizer Generator editor commands to support wpis. Editor commands that have their standard functionality are described in The Synthesizer Generator Reference Manual [16]. Only commands whose functionality has changed significantly are mentionned here.

3.1 Compilation-Unit Buffers

Most editing buffers are associated with compilation-units of some system; such compilation-unit buffers have names of the form *system-name/compilation-unit-name*. Compilation unit buffers are created by **new** and are deleted by **delete-buffer.** The syntactic category of a compilation-unit buffer is program.

new *name cu-name new-window*

Creates a new compilation-unit *cu-name* for the system *name* and displays it in a window. If such a compilation-unit already exists, an error is returned. The compilation-unit is initialized with a placeholder for the syntactic category *program,* the write permission is set to read-write, and the compilation-unit is not associated with any file.

If *new-window* is yes, the new compilation-unit is displayed in a new window and if no, the new compilation-unit is displayed in the window from which the command was invoked.

The programmatic interface to this command is:
New(STR *name*, STR *cu-name*, BOOL *b*),

where *b* is *true* or *false* whether the *new-window* parameter is yes or no.

3.2 Files

Systems are stored in directories of the file system, with a file for each compilation-unit of the system. Many commands in wpis operate on entire systems (i.e., on all the compilation-unit files in a directory). For example, **open** reads a directory into a system of the same name; **save** writes a system to its associated directory; **save-as** associates a system with a new directory name, renames the system accordingly, and writes the system to that directory; **list-systems** lists all systems and their properties; **close** closes the current system.

The write-permission of compilation-unit buffers created by opening a system is derived from the write-protection of the file in the system directory: the buffer is read-only if the file is write-protected and is read-write otherwise. Attempts to save a system that would overwrite an existing write-protected file in the file system are illegal.

close F8,^Xc

Closes a system, deleting its associated buffers and windows.

open ***name cu-name new-window*** L7,F7,^X^R,^X^F,^X^V

Displays the compilation-unit *cu-name* of system *name* in a window. If the compilation-unit is already open, then the window is changed to display that compilation-unit. If system *name* is not being edited, all files of directory *name* are read into their associated compilation-units and the system becomes associated with the directory for subsequent output commands.

The dialog box for open includes a system browser and a compilation-unit browser that can be used to obtain the *name* and *cu-name* parameters. When a system has been selected, the compilation-unit browser (initially empty) contains the list of the system's compilation-units. If the system *name* doesn't contain a compilation-unit named *cu-name*, the compilation-unit displayed will either be main, if there is a compilation-unit named main; otherwise an arbitrary compilation-unit is displayed.

If *new-window* is yes, the new compilation-unit is displayed in a new window and if no, the new compilation-unit is displayed in the window from which the command was invoked.

The programmatic interface to this command is:
Open(STR *name*, STR *cu-name*, BOOL *b*),
where *b* is *true* or *false* whether the *new-window* parameter is yes or no.

save ^Xs

Writes the contents of the current system to its associated directory.

save-as ***directory*** ^X^W

Writes the contents of the current system to the given *directory*.

The dialog box for **save-as** includes a system browser that can be used to obtain the *directory* parameter.

The programmatic interface for this command is:
Save_as(STR *directory*).

save-all ^X^M

Every modified system with an associated directory is written to that directory.

save-exit ^Xf

Save-all followed by **exit.**

copy-system *name* ^X^B

Create a new system *name*—a copy of the current system—and display it in a new window. Each compilation-unit of the current system is copied into a compilation-unit of the system *name*.

The programmatic interface for this command is:
Copy_system(STR *name*).

list-systems ^X^B

Lists all system and their compilation-units in buffer *system-list* and displays it in a window. Systems are listed in alphabetical order with their directory, system-id, and flags. The flag M indicates that the system has been modified since it was last saved to a directory; the flag C indicates that the system is *consistent* and that the *system dependence graph* has been successfully built. For each system, its compilation-units are listed with their associated file and flagged with M if the compilation-unit has been modified since it was last saved to a file.

The programmatic interface to this command is:
List_systems().

Acknowledgements

Contributors to the implementation were Thomas Reps, Thomas Bricker,Genevieve Rosay, Victor Barger, and Samuel Bates. Others who contributed to the development of algorithms used in the system were Susan Horwitz, Jan Prins, David Binkley, Thomas Ball, G. Ramalingam, Philip Pfeiffer IV, and Wuu Yang.

Bibliography

[1] D Binkley. Multi-procedure program integration. Technical report, Computer Sciences Department, University of Wisconsin, Madison, WI, August 1991. Ph.D TR-1038.

[2] J. Ferrante, K. Ottenstein, and J. Warren. The program dependence graph and its use in optimization. In *ACM Trans. Program. Lang. Syst.*, pages 319–349, July 1987. TOPLAS.

[3] S. Horwitz. Identifying the semantic and textual differences between two versions of a program. *ACM SIGPLAN 90 Conference on Programming Language Design a nd Implementation*, 25(6):234–245, June 1990. PLDI.

[4] S. Horwitz, J. Prins, and T. Reps. Integrating non-interfering versions of programs. In *Conference Record of the Fifteenth ACM Symposium on Principles of Programming Languages*, pages 133–145, New York, NY, January 1988. ACM. POPL San Diego, CA, January 13-15.

[5] S. Horwitz, J. Prins, and T. Reps. Integrating non-interfering versions of programs. *ACM Trans. Program. Lang. Syst.*, 11(3):345–387, July 1989. TOPLAS.

[6] S. Horwitz and T. Reps. Efficient comparison of program slices. *Acta Informatica*, 28:713–732, 1991.

[7] S. Horwitz and T. Reps. The use of program dependence graphs in software engineering. In *Proceedings of the Fourteenth International Conference on Software Engineerin g*, pages 392–411, New York, NY, 1992. ACM. ICSE, May 11-15, 1992, Melbourne, Australia.

[8] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.*, 12(1), January 1990.

[9] D.J. Kuck, R.H. Kuhn, B. Leasure, D.A. Padua, and M. Wolfe. Dependence graphs and compiler optimizations. In *Conference Record of the Eighth ACM Symposium on Principles of Programming La nguages*, pages 207–218, New York, NY, 1981. ACM. POPL, Williamsburg, VA, January 26-28, 1981.

[10] K.J. Ottenstein and L.M. Ottenstein. The program dependence graph in a software development environment. *Proceedings of the ACM SIG-*

SOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, 19(5):177–184, May 1984. Pittsburgh, PA, Apr. 23-25, 1984) ACM SIGPLAN Notices.

[11] William Pugh. The omega test: a fast and pratical integer programming algorithm for dependence analysis. *Supercomputing*, 1991.

[12] William Pugh. Eliminating false data dependences using the omega test. *ACM SIGPLAN*, 1992.

[13] T. Reps. Demonstration of a prototype tool for program integration. Technical report, Computer Sciences Department, University of Wisconsin, Madison, WI, January 1989. TR-819.

[14] T. Reps and T. Bricker. Illustrating interference in interfering versions of programs. In *Proceedings of the Second International Workshop on Software Configuration Management*, volume 17, pages 46–55, Princeton, NJ, November 1989. ACM SIGSOFT Software Engineering Notes.

[15] T. Reps and T. Teitelbaum. *The Synthesizer Generator: A System for Constructing Language-Based Editors*. Springer-Verlag, New York, NY, 1988.

[16] T. Reps and T. Teitelbaum. *The Synthesizer Generator Reference Manual: Third Edition*. New York, NY, 1988.

Index

display-affectedcommand, 11

affected, 11,14
alias, 4
At parameter vertices, 9

batch mode, 6
buffer, 2,6,21,22
    status 11
    system-list 23
    read-write permission 22

call-graph, 3
clear-all-systems command, 15
clear-difference command, 10
clear-system command, 15
close command, 22
compilation-unit, 2,4,21,22
Compilation-Unit Buffers, 21
consistent, 3
control flow graph, 18
control-flow, 3
copy-system command, 23
cref, 16
Cross Reference, 16

Debugging, 18
debugging, 17
dialog box, 11,14,15
difference-by-vertex command, 9
difference-by-vertex-and-edge command, 10
differencing, 1,9,15
directly affected, 14
directory, 22
display-affected command, 14
display-preserved command, 15

edge, 13
    class 17
edges, 16
Edges, 16
editor, 2,4,21
Editor Tags, 4
external store, 3,18

file, 22
    protection 22
Files, 22
forward-control-false-edge command, 17
forward-control-true-edge command, 17
forward-slice command, 8
forward-slice-augment command, 8
forward-unslice command, 8

global variable, 2
global-cfg-vertex-store command, 18

integrate command, 11
integration, 1,11
    record 11
Integration, 11
Integration Tool Commands, 19
INTEGRATOR_LOC, 4
interference, 12
interpreter, 6
Introduction, 1
isomorphism, 9

language, 2
Language, 2
Language-Based Differencing, 9
linkage grammar, 18
list-systems command, 23

menu
  Integrate 11,14
  Ldiff 9
  Show 18
  Structure 16

new command, 21

open command, 6,22

parameter, 2,7,9,17
preserved, 15
procedure dependence graph, 18

read-sdg command, 19

save command, 22
save-all command, 23
save-as command, 23
save-exit command, 23
scripts
  .synrc 6
  invoking from command line 6
search-cross-references command, 16
selection, 7,12
semantic, 9
set-cfg-fields command, 19
set-pdg-fields command, 19
set-slice-isomorphism command, 10
set-slice-parameters command, 9
show-augmented-cfg command, 18
show-call-graph command, 18
show-cfg command, 18
show-edges command, 17
show-integration-record command, 12
show-interference command, 13
show-merged-ordered-system command, 18
show-merged-system command, 18
show-pdg command, 18
show-system-ag command, 18
show-system-cfgs command, 18
show-system-pdgs command, 18
slice, 7
  sets 7
slice command, 8
Slice phase, 9
slice-augment command, 8
slice-isomorphism command, 10
SliceView, 8
Slicing, 8
slicing, 1
strongly affected, 14
style, 7,9,14
styles, 16
syntactic category of buffer, 21
system, 2,3,22
  tmp 6
  browser 11,14,15
System Consistency, 3
System Dependence Graph, 3
system dependence graph, 3
Systems and Compilation-units, 4

tag, 4
test-reconstitute command, 18

unslice command, 8
unslice-all-systems command, 8
unslice-system command, 8

window, 6,8
wpis
  script 6
WPIS - Getting In and Out of The Editor, 6
write-sdg command, 19