
Indus - Kaveri

Ganeshan Jayaraman, Kansas State University
<ganeshan@cis.ksu.edu>

Table of Contents

Introduction	1
Requirement	1
Name	1
Install / Uninstall	2
Installation	2
Uninstall	2
Concepts	2
Setup	3
Setting up Kaveri	4
Running Kaveri	6
Steps	6
Features	8
Toolbar buttons	8
Slice label decoration	9
Slice View	9
Appendix - How to pick criteria	10
Examples of picking Jimple criteria	10
Appendix - Performance	11
Bibliography	11

Introduction

Kaveri is an eclipse plug-in front-end for the *Indus Java slicer*. It utilizes the Indus program slicer to calculate slices of Java programs and then displays the results visually in the editor. The purpose of this project is to create an effective tool for simplifying program understanding, program analysis, program debugging and testing.

Requirement

1. **Eclipse** : Eclipse version 3.0 or higher is needed.
2. **Indus Plug-in**: Indus Plug-in version 0.3 or higher.

Name

Kaveri is one of the major rivers in India, my native land. A river has many branches through which it flows. In a similar way a program branches in its execution. It is hoped that *Kaveri* makes it easier to decipher this flow and increases your understanding of the program.

Install / Uninstall

Installation

Install: Using zip files

Kaveri can be downloaded from the SAnToS sourceforge site at: <http://projects.cis.ksu.edu/projects/indus/>. To install, unzip the downloaded file `edu.ksu.cis.indus.Kaveri-$VERSION.zip` in `ECLIPSE_HOME`. Make sure that the folder names are preserved while decompressing. Note that Kaveri requires the **Indus plug-in** in order to run. The Indus plug-in can be downloaded from <http://projects.cis.ksu.edu/projects/indus/>.

Uninstall

To uninstall Kaveri, remove the `edu.ksu.cis.indus.kaveri_$VERSION` folder from your `ECLIPSE_HOME/plugins/` directory.

Concepts

Configuration

The Indus Java Slicer requires a configuration to specify details such as the type of slice to be performed (backward, forward or complete), the types of dependencies to track in the slice (ready, divergence, interference) [HatcliffSAS99] and the parameters for each dependency. In short, it is the configuration of the slicer. To learn more about the configuration please look at the Indus Java Slicer Documentation [<http://projects.cis.ksu.edu/docman/view.php/12/71/slicer-ug.pdf>].

Jimple

Jimple is a typed 3-address intermediate representation of Java [VH98] and Indus slicer operates on Jimple. Hence, all the Java code is converted into Jimple and then fed to the Slicer. The slicer calculates the slice of the given system in terms of Jimple.

Example 1. Example conversion of Java into Jimple

Consider `nw++`, a Java expression. Following is a Jimple representation of this expression:

- `$i2 = r0.<myPackage.Monitor: int nw>`
- `$i3 = $i2 + 1`
- `r0.<myPackage.Monitor: int nw> = $i3`

In the first statement, the reference from `monitor` field is assigned to a local variable `i2`. In the next statement, the local variable `i3` is assigned the incremented value of `i2` and finally the original field is updated.

As the Indus Java Slicer works on Jimple, the criteria for slicing is specified as Jimple chunks. From the above illustration, a Java statement can map to many Jimple statements. Therefore, when a Java statement/expression needs to be used as a slice criteria in *Kaveri*, the user has to pick one or more of the Jimple statements to which the Java statement/expression is mapped to. For more information about Jimple please refer to [VH98]. For more exposition about how to pick criteria closest to your requirements, please refer to **Appendix - How to pick criteria**.

Value of the expression

When an expression / statement is specified as a criteria, does this mean that:

- The value of the expression should be preserved (the expression is executed) or
- The control reaching the expression should be preserved (the expression is not executed)

In the former we consider all those statements which might affect the computed value of the expression while in the latter we consider only those statements through which control can reach the selected expression or statement.

Example 2. Example to show the influence of control or value while choosing a criteria

Consider the following fragment of code.

```
public static void main(final String[] args) {
    int a, b;
    a = b = 1 + randInt();
    if (a == 2) {
        a = 10;
    }
    else {
        b = 20;
    }
    a = a + b;
}
```

If the statement `a = a + b` is chosen as the criteria, picking control or value causes different slices to be generated. If *control* is picked, the statements corresponding to `if` conditional will be ignored in the slice as they don't affect the reachability of the chosen statement. However if the *value of the expression* is picked then the conditional statements have to be included as they affect the computed value of the expression.

Setup

Setting up Kaveri

Preferences

Indus Preferences

Before using Kaveri the user needs to define a set of configurations to use with the slicer. To do so open **Window -> Preferences -> Indus Preferences** from the Eclipse menu. A default configuration is provided. (Refer to Figure 1). From the Configuration tab you can manage the set of slicer configurations. Be sure to press the **Apply** button after any changes to preserve the settings immediately. Please ignore the tabs marked *Colors* and *View* as their usage is reserved for the next version. Using them produces no effect currently.

Figure 1. Slicer Configuration Preferences

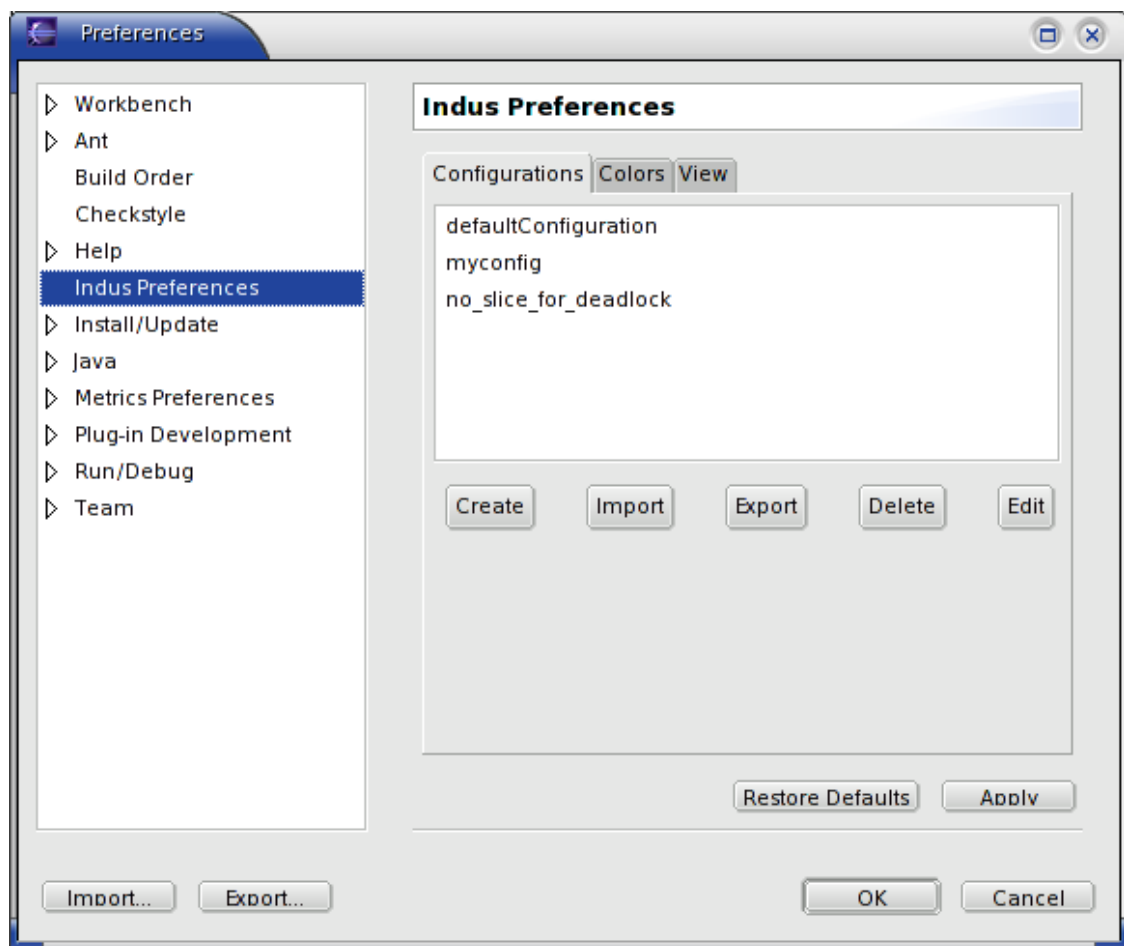
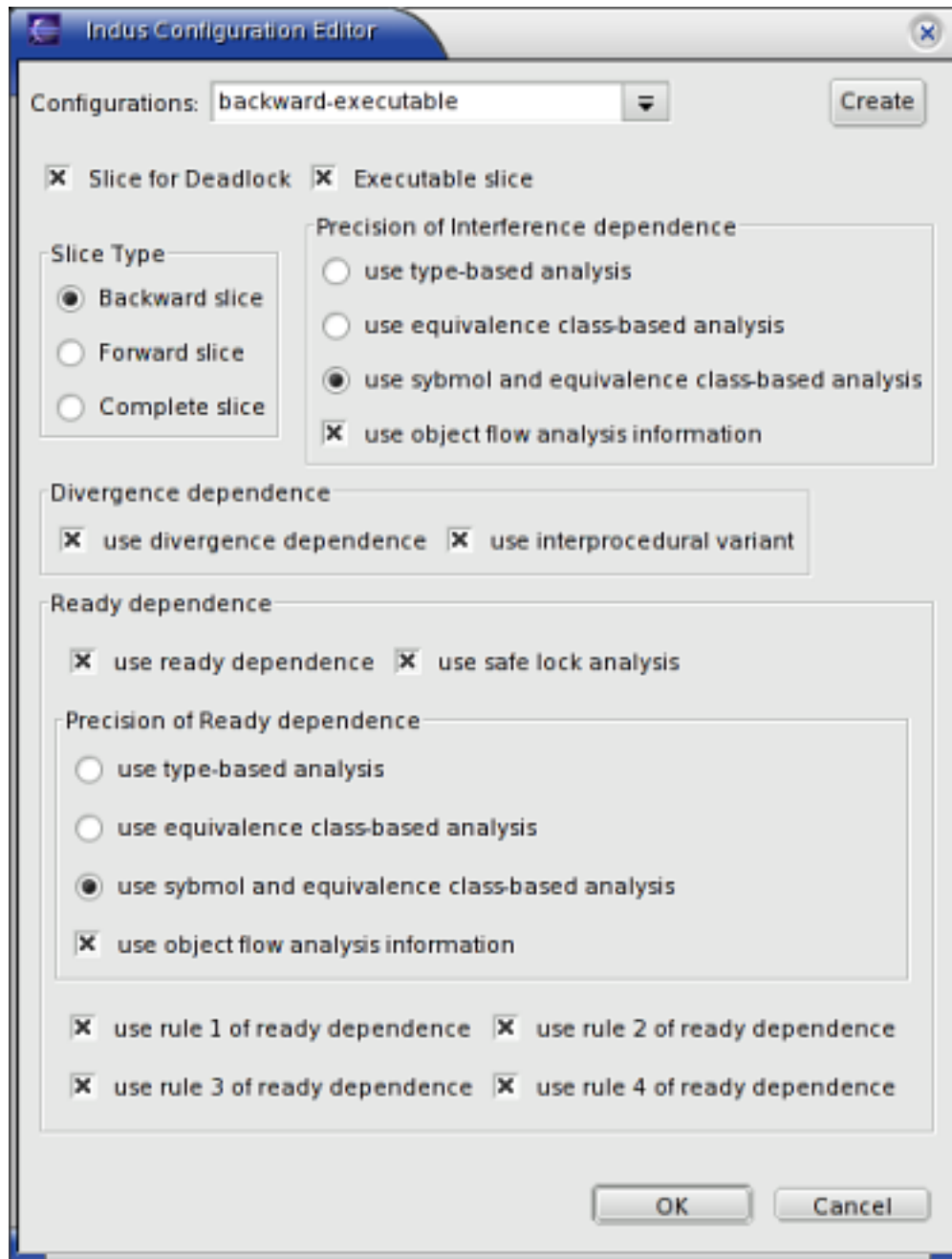


Figure 2. Editing the configuration

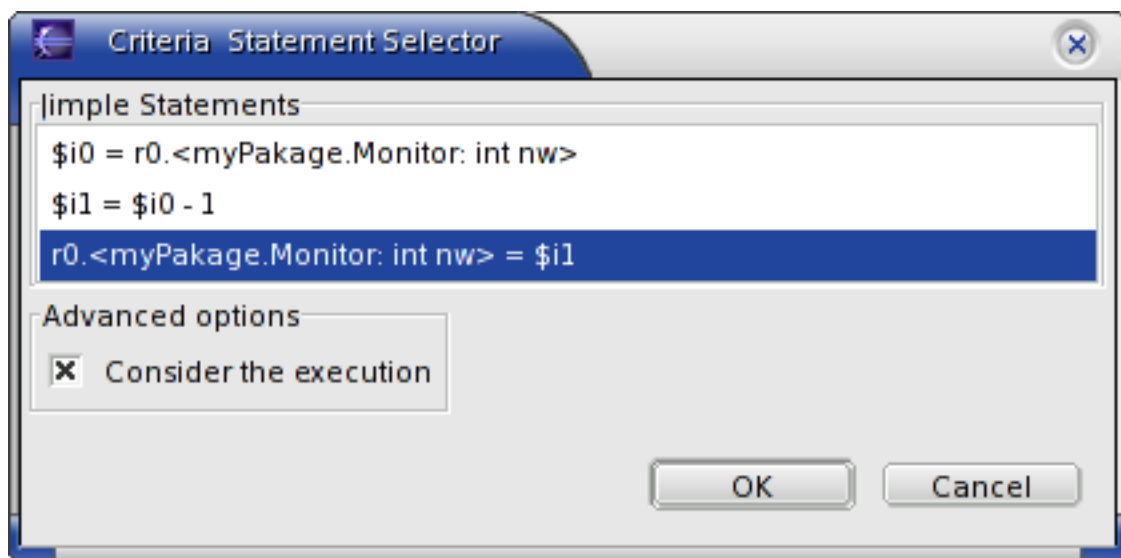
**Buttons:**

- Create** Enables you to create a new slicer configuration. Be sure to enter a descriptive name for the configuration in the **Indus configuration editor** dialog.
- Import** Allows you to import a slicer configuration from an existing file.
- Export** Allows you to export a configuration to a file for later use.
- Delete** Deletes the chosen configuration.
- Edit** Allows you to edit an existing configuration.

Setting up the criteria

A Java project can be associated with a set of *slicing criteria* pertaining to the Java files included in the project. Kaveri currently allows you to add Java statements as criteria for slicing. To add a Java statement as a criteria, select the statement in the editor and choose **Indus -> Add to Criteria** from the context menu. If the equivalent Jimple statements for the statement are found, the **Criteria Statement Selector** dialog pops up allowing you to pick the Jimple statement closest in meaning to the required criteria. Enable or disable the checkbox marked "**Consider the execution**" to slice using *value of the expression* or with the *control* reaching the statement respectively. Click **OK** to add the selected Jimple statement to the set of available criteria. Note that Kaveri checks the criteria for duplicates, so only unique criteria can be added.

Figure 3. Criteria Picker

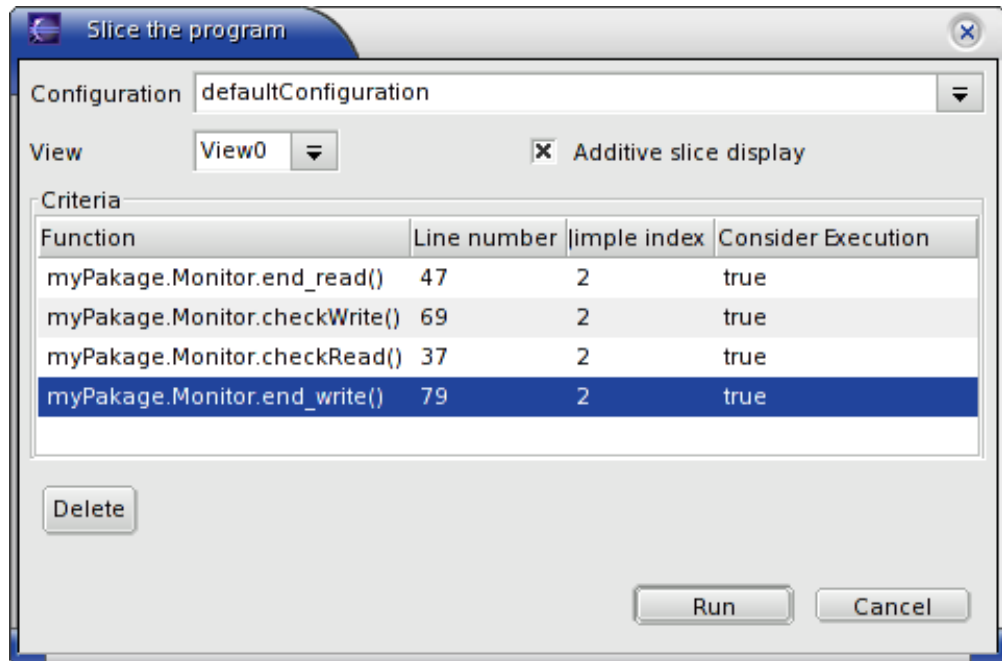


Running Kaveri

Steps

- Step 1 Right click the Java file or Java project in the *navigator* or *package explorer* window in Eclipse and choose either **Indus -> Slice Java file** or **Indus -> Slice project** from the context menu. The difference is that in the former only the specified file is sliced while in the latter all the files present in the Java project are sliced.
- Step 2 In the slice configuration dialog that appears, pick the *slice configuration* and the *criteria* for slicing. Note that you can add multiple criteria by selection them from the list of criteria. The **Additive slice display** check box indicates that the new slice should be added onto the display of the previous slice. This displays the slice as the union of the previous and the new slice. Press the **Run** button to start the slicing. A progress dialog indicates the progress of the slicing.

Figure 4. Running the slicer



- Step 3 After the slice is completed, open the Java file if it is not currently open. If the *Kaveri decorator* is enabled (Refer to Figure 7) the slice is automatically highlighted. Else press the *Slice Toggle* button (Refer to Figure 6) to highlight the slice in the editor. There are two types of highlighting that are displayed. Java statements for which all the corresponding Jimple statements have the slice tag are highlighted in once color while those in which only a part of the Jimple statements have the slice tag are highlighted in a different color. To change the color for the slice highlighting go to **Window -> Preferences -> Workbench -> Editors -> Annotations** from the Eclipse menu and change the color for the '**IndusSliceAnnotation**' for the complete slice element highlight annotation and **IndusPartialSliceAnnotation** for the partial slice element highlight annotation. You may have to reopen the file in the editor and toggle the slice highlight for the change in color to be updated.

Figure 5. Viewing the slice

```

63     }
64     }
65     private synchronized boolean checkWrite()
66     {
67         if(nw==0 && nr==0)
68         {
69             nw++; ← Complete Slice Element
70             return true;
71         }
72         else
73             return false;
74     }
75     void end_write()
76     {
77         synchronized(this)
78         {
79             nw--; ↓ Partial Slice Element
80         }
81         synchronized(objectR) { objectR.notifyAll();}
82         synchronized(objectW) { objectW.notify();}
83     }
84 };

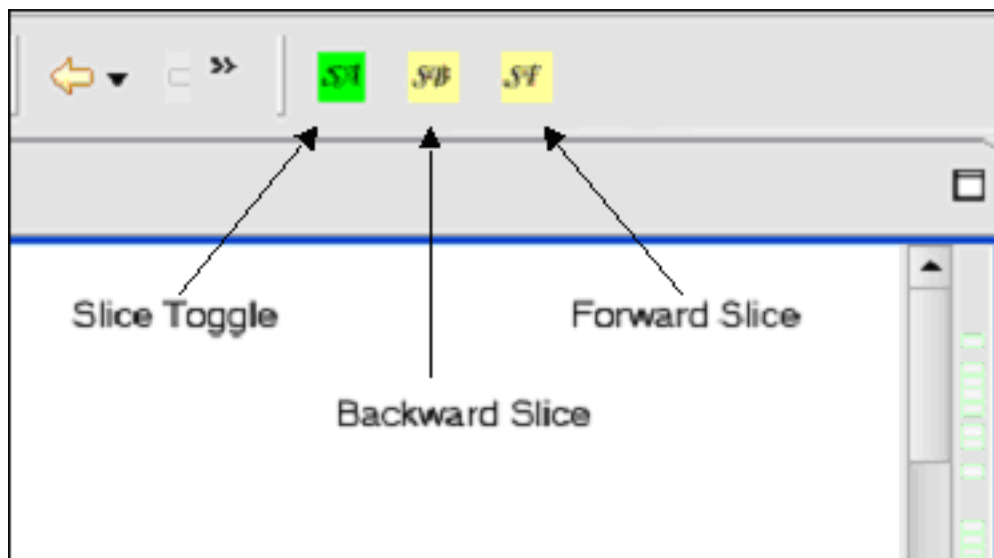
```

Features

Toolbar buttons

The following toolbar buttons are activated when a Java file is open in the editor.

Figure 6. Toolbars



Slice highlight toggle

The slice toggle menu button is used to toggle the slice highlighting on / off in the Java editor. After a file is sliced, open it and press this button to show the slice in the editor.

Backward Slice

To run a backward slice without going through the criteria and configuration selection phases, you can select a Java statement and press the *backward slice menu button* to run a backward slice with the selected statement as the criteria. After the slice is performed the relevant statements are highlighted automatically.

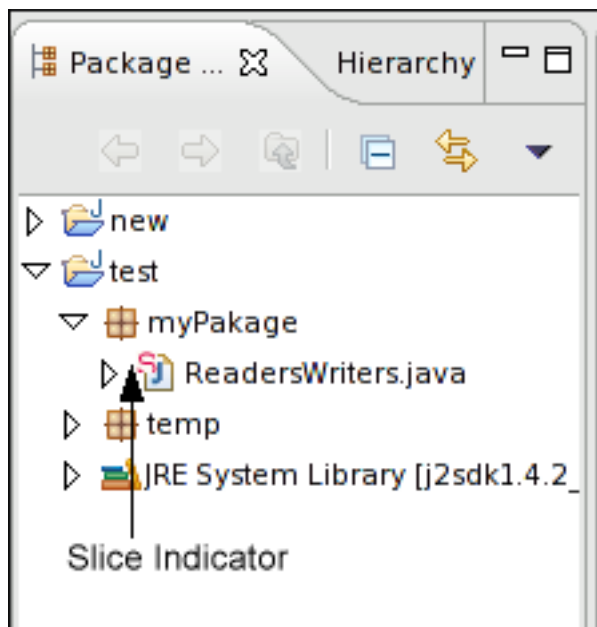
Forward Slice

To run the forward slice on a given Java statement select it and press the *Forward Slice menu button*. Note that Indus currently supports only non-executable forward slicing.

Slice label decoration

Kaveri has a *label decorator* that annotates the list of Java files shown in the package or navigator window with a marker to indicate that the file has a slice associated with it. To enable or disable it go to **Window -> Preferences -> Workbench -> Label Decorations** from the Eclipse menu and toggle the checkbox with the *Kaveri.Decorator* label.

Figure 7. Slice label decorator

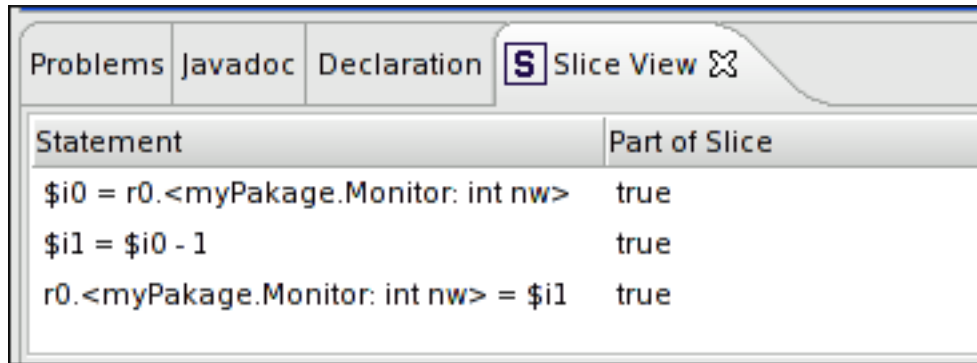


Slice View

As mentioned earlier, Kaveri highlights Java statements with complete and partial slices in different colors. To know which of the Jimple statements have the slice and which don't requires the **Slice View** to be enabled. To do so, select **Window -> Show View -> Other** from the Eclipse menu and then select

Kaveri -> **Slice View** from the dialog shown. The *Slice view* can only be used after a slice has been performed. To view the Jimple statements for a sliced Java statement, select the Java statement in the editor and select **Indus** -> **View Slice** from the context menu. The view then displays the equivalent Jimple statements for the selected Java statement and also indicates whether they are part of the slice.

Figure 8. Slice View



Statement	Part of Slice
<code>\$i0 = r0.<myPackage.Monitor: int nw></code>	true
<code>\$i1 = \$i0 - 1</code>	true
<code>r0.<myPackage.Monitor: int nw> = \$i1</code>	true

Appendix - How to pick criteria

As mentioned before, the criteria for slicing is a Jimple statement. When a Java statement is picked, the user is allowed to select one of the corresponding Jimple statements as the criteria. The following examples show how to choose the correct Jimple statement depending on the requirement.

Examples of picking Jimple criteria

Example 3. Setting up criteria for an assignment statement

Consider the simple statement : `nw++` , where `nw` is a integer. The equivalent Jimple for this statement consists of the following:

- `$i2 = r0.<myPackage.Monitor: int nw>`
- `$i3 = $i2 + 1`
- `r0.<myPackage.Monitor: int nw> = $i3`

Here `r0.<myPackage.Monitor: int nw>` corresponds to the variable `nw`. So to pick `nw = nw + 1` as a criteria you would pick the last statement `r0.<myPackage.Monitor: int nw> = $i3` as in that statement the final value of `nw` is assigned.

Example 4. Setting up criteria for a function call

Consider the expression `monitor.start_write()` . The Jimple for this consists of two statements:

- `r1 = r0.<myPackage.Writers: myPackage.Monitor mon>`

- `virtualinvoke r1.<myPackage.Monitor: void start_write()>()`

To pick the function call, the second statement `virtualinvoke r1.<myPackage.Monitor: void start_write()>()` is to be selected as it involves the call to the function `start_write()` in the `Monitor` class.

Example 5. Setting up criteria for a conditional

Consider the expression: `if (!empty(Clist))`. The equivalent Jimple for this statement is:

- `$r4 = r0.<temp.DiskScheduler: java.util.LinkedList Clist>`
- `$z0 = virtualinvoke r0.<temp.DiskScheduler: boolean empty(java.util.LinkedList)>($r4)`
- `if $z0 != 0 goto $r7 = r0.<temp.DiskScheduler: java.util.LinkedList NList>`

To pick the function call `empty(Clist)` you would choose the second statement `$z0 = virtualinvoke r0.<temp.DiskScheduler: boolean empty(java.util.LinkedList)>` as it invokes the `empty()` method. On the other hand if you wanted to choose the `if` conditional you would choose the last line `if $z0 != 0 goto $r7 = r0.<temp.DiskScheduler: java.util.LinkedList NList>`

Appendix - Performance

Kaveri has a JVM memory requirement for a smooth operation. Typically a memory specification of 200 MB for the Java virtual machine suffices for small projects. For large projects it is advisable to allocate more memory. To supply more memory to Eclipse pass the string `"-vmargs -Xmx200m"` as a parameter to the Eclipse launch application.

Bibliography

Venkatesh Prasad Ranganath. *Indus project Documentation*.
[http://projects.cis.ksu.edu/docman/index.php?group_id=12].

[VH98] Raja Vallee-Rai, Laurie J. Hendren. *Jimple: Simplifying Java Bytecode for Analyses and Transformations*.

David Binkley, Keith Brian Gallagher. *Program Slicing*.

Frank Tip. *A survey of Program Slicing Techniques*.

[HatcliffSAS99] John Hatcliff, James Corbett, Matthew Dwyer, Stefan Sokolowski, Hongjun Zheng. *A Formal Study of Slicing for Multi-threaded Programs with JVM Concurrency Primitives*.

John Hatcliff, Venkatesh Prasad Ranganath. *Pruning Interference and Ready Dependence for Slicing Concurrent Java Programs*.