























定值到达分析用户手册











撰写人：张书博

关于定值到达分析的详细操作步骤。

1. 打开 Eclipse，然后选择对应项目。对应项目的结构如下：

- >  JRE System Library [jdk1.8.0_45]
- >  (default package) [JA-ver2 master]
- >  analyzer.dataTable
- >  analyzer.nullCheck
- >  analyzer.qualitas
- >  graph.basic
- >  graph.cfg
- >  graph.cfg.analyzer
- >  graph.cfg.creator
- >  gui.astViewer
- >  gui.toolkit
- >  nameTable
- >  nameTable.creator
- >  nameTable.filter
- >  nameTable.nameDefinition
- >  nameTable.nameReference
- >  nameTable.nameReference.referenceGroup
- >  nameTable.nameScope
- >  nameTable.visitor
- >  sourceCodeAST
- >  util
- >  Referenced Libraries

2. 点击 gui.astViewer

- >  graph.cfg.creator
- ▼  gui.astViewer
 - >  ConciseASTViewer.java
 - >  ConciseASTVisitor.java
 - >  ControlFlowGraphViewer.java
 - >  ReachNameViewer.java
 - >  SimpleASTViewer.java
 - >  SimpleASTVisitor.java
 - >  SimpleProgressMonitor.java
 - >  TestASTViewer.java

右键 TestASTViewer.java，然后选定运行。
出现如下 GUI。

3. 点击文件，选择定值到达分析，然后选定一个.java 文件进行定值到达分析。（选择 JAnalyzer 项目本身的代码就可以）会有如下的效果图。

右边配备了详细的文字说明，从字面意思去理解分析的结果就可以。（分析是以可执行点为单位，每个可执行点的名字定义都会被分析（显示定义名字的定值表达式），还涉及到名字定义的根源定值的分析等）

The screenshot displays the JAnalyzer tool interface. On the left, the source code for `SourceCodeLocation.java` is shown, with line numbers 97 to 142. The code includes methods for comparing locations and calculating hash codes. On the right, the 'Abstract Syntax Tree' (AST) view is active, showing the control flow graph (CFG) for the `isBetween` method. The analysis results are presented in a table-like format, detailing the state of variables and expressions at various points in the code.

Line	Code Snippet	Analysis Result
97	<code>if (lineNumber != other.lineNumber) return (lineNumber < other.lineNumber);</code>	基于该可执行点 (End: 161: 2) 的定值到达分析结果
98	<code>return column != other.column;</code>	在节点ID为[164: 2]的CFG节点 对column名字定义 使用root.getColumnNumber(position) + 1表达式求定值 名字定义位置[158: 6] 对应到达定值已经是根源到达定值
100	<code>return column != other.column;</code>	在节点ID为[164: 2]的CFG节点 对lineNumber名字定义 使用root.getLineNumber(position)表达式求定值 名字定义位置[158: 6] 对应到达定值已经是根源到达定值
101	<code>return column != other.column;</code>	在节点ID为[164: 2]的CFG节点 对position名字定义 使用node.getStartPosition() + node.getLength() - 1表达式求定值 名字定义 对应到达定值已经是根源到达定值
102	<code>/**</code>	基于该可执行点 (164: 2) 的定值到达分析结果
103	<code>* Check the current location is between start and end or not</code>	
104	<code>*/</code>	
105	<code>public boolean isBetween(SourceCodeLocation start, SourceCodeLocation end) {</code>	
106	<code>if (start == null end == null) return false;</code>	
107	<code>if (!fileInitName.equals(start.fileInitName) !fileInitName.equals(end.fileInitName)) return false;</code>	
108	<code>if (lineNumber < start.lineNumber) return false;</code>	
109	<code>if (lineNumber > end.lineNumber) return false;</code>	
110	<code>if (lineNumber == start.lineNumber && column < start.column) return false;</code>	
111	<code>if (lineNumber == end.lineNumber && column > end.column) return false;</code>	
112	<code>return true;</code>	
113	<code>}</code>	
114	<code>@Override</code>	
115	<code>public boolean equals(Object other) {</code>	
116	<code>if (this == other) return true;</code>	
117	<code>if (other == null) return false;</code>	
118	<code>if (!(other instanceof SourceCodeLocation)) return false;</code>	
119	<code>SourceCodeLocation otherLocation = (SourceCodeLocation) other;</code>	
120	<code>if (fileInitName == null) {</code>	
121	<code>if (otherLocation.fileInitName != null) return false;</code>	
122	<code>if (otherLocation.fileInitName == null) return false;</code>	
123	<code>if (!fileInitName.equals(otherLocation.fileInitName)) return false;</code>	
124	<code>if (lineNumber != otherLocation.lineNumber column != otherLocation.column) return false;</code>	
125	<code>return true;</code>	
126	<code>}</code>	
127	<code>@Override</code>	
128	<code>public int hashCode() {</code>	
129	<code>int result = 37;</code>	
130	<code>if (fileInitName != null) result = fileInitName.hashCode();</code>	
131	<code>result = 31 * result + lineNumber;</code>	
132	<code>result = 31 * result + column;</code>	
133	<code>return result;</code>	
134	<code>}</code>	
135	<code>public static SourceCodeLocation getStartLocation(ASMNode node, CompilationUnit root, String compilationUnitFileName) {</code>	
136	<code>int position = node.getStartPosition();</code>	
137	<code>}</code>	
138	<code>}</code>	
139	<code>}</code>	
140	<code>}</code>	
141	<code>}</code>	
142	<code>}</code>	

4.关于定值到达分析还有一个选项是最大定值到达分析，注意，最大定值到达分析的含义就是，选定一个.java 文件，分析的是这个.java 文件的父目录下的所有.java 文件中的最长（大）的一个方法定义（从父目录的所有的.java 文件中选择一个最大的方法定义，最大的方法定义属于一个.java 文件，一个.java 文件可能有很多方法定义）。所以选定的.java 文件有可能不会被分析!!!! 要和老师讲清楚这点，以免产生误会。

个人建议展示完定值到达分析后，直接关掉 GUI，重新运行 TestASTViewer.java。然后再选择最大定值到达分析，当然，接着上一步选最大定值到达分析也是可以的。
最大定值到达分析的效果图如下：



5. 接下来是可视化部分

注意：运行一次定值到达分析，就会产生一个.dot 文件在电脑中。至于产生的位置在哪里，取决于 TestASTViewer.java 中的 OutputStream os 的配置，如下图所示：
目前这么设置，点击运行一次定值到达分析产生的.dot 就是在 C:\Java\test2.dot 位置。

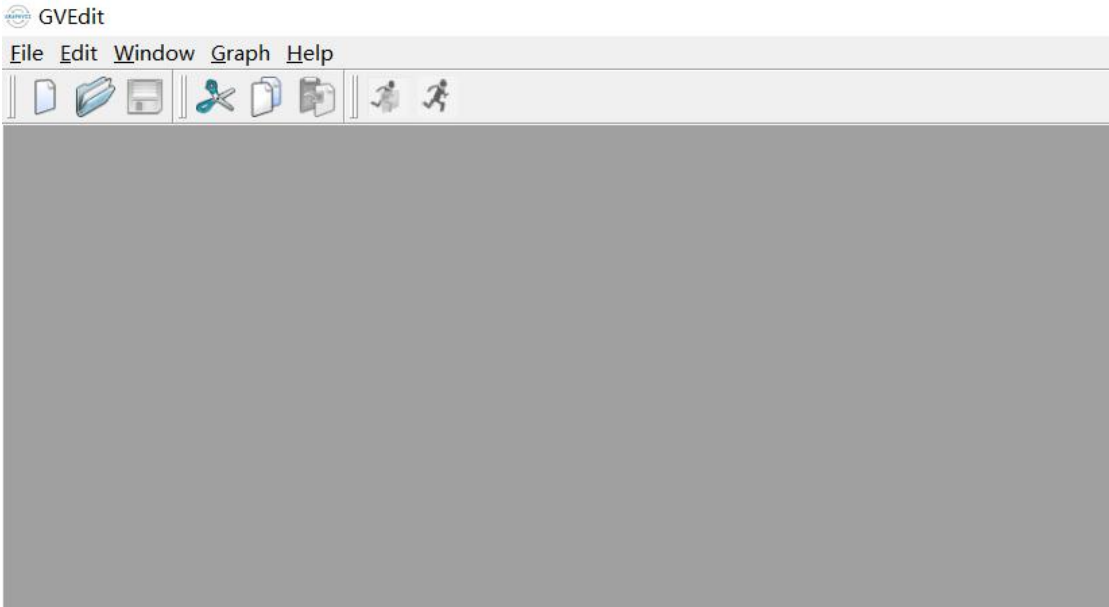
（接着定值到达分析再点击最大定值到达分析不会生成.dot，.dot 还是原来定值到达分析的.dot）

所以在上面我建议关掉程序重新运行，点击最大定值到达分析，这样会产生一个新的.dot，从而覆盖掉上次产生的.dot。

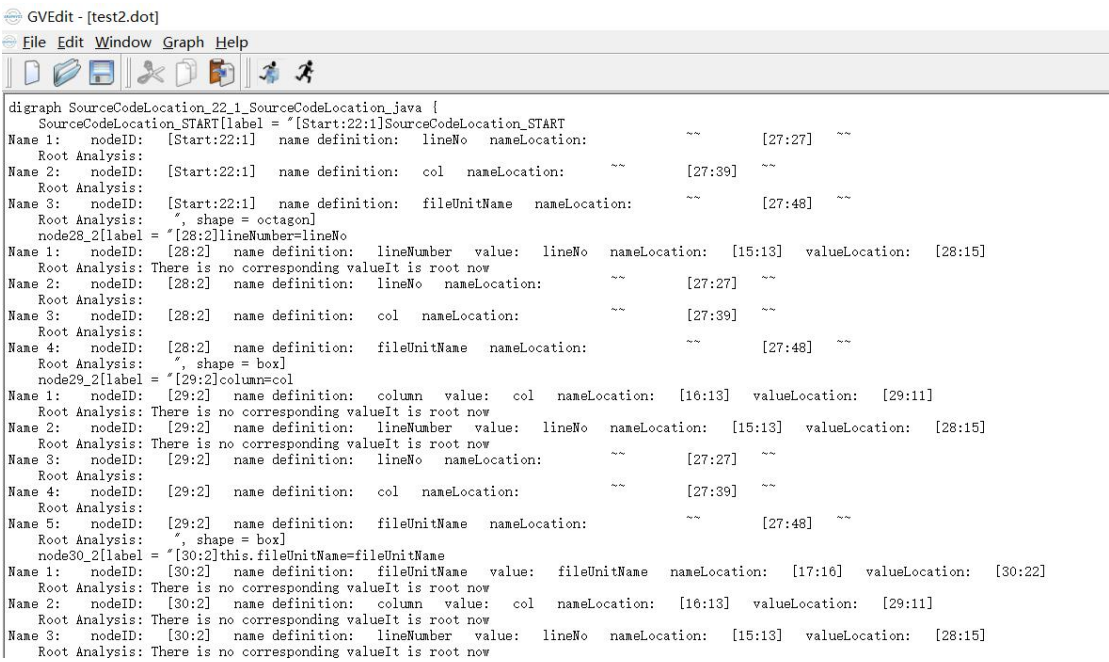
```
public DemoMenuCreator(Container place, JFrame topLevelFrame) {
    this.place = place;
    this.topLevelFrame = topLevelFrame;
    fileOpener = new FileChooserAndOpener(topLevelFrame);
    try {
        OutputStream os = new FileOutputStream("C:\\Java\\test2.dot");
        output = new PrintWriter(os);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

当然，仅仅有.dot 是不够用的，根据老师的文档，还要安装 GVEdit 并配置相关的环境。（这里请自行百度）

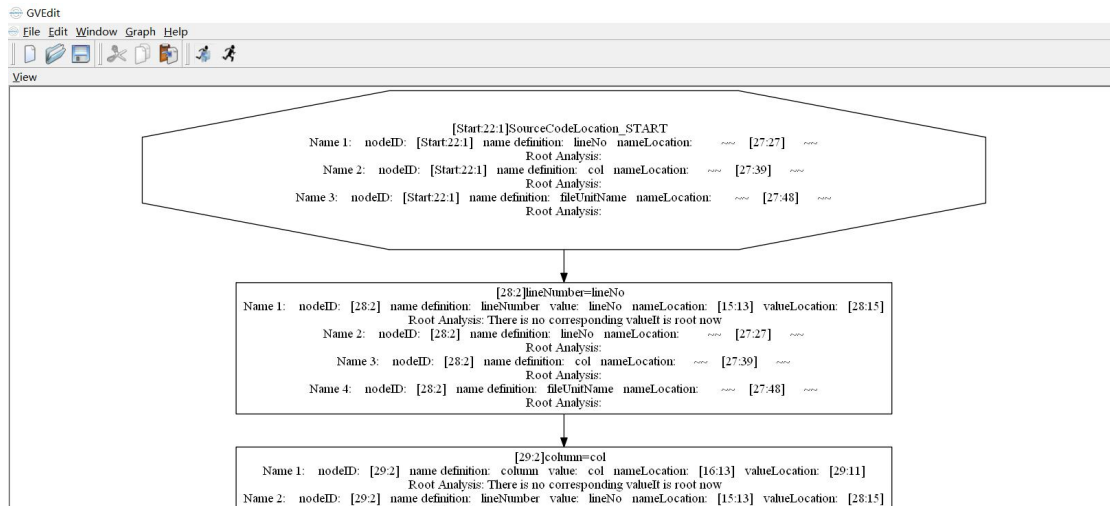
下面讲解如何运行：




点击 File，点击 open，选择之前定值到达分析生成的.dot。



这是一般会自动弹出.dot 对应的 view。



如果没有 **view**，手动点击  这里即可。注意，如果点击第一次没反应，报错了，那就再点一次，就会弹出 **view**。（我也不知道为什么第一次报错，可能与.dot 包括了多个可执行点的定值到达分析有关）

要注意的是！定值到达分析生成的.dot 包括了那个.java 文件中的所有可执行点，生成的 **view** 只是第一个可执行点的 **view**。如果想看接下来的，要手动删除第一个 **digraph**，然后再点击



，这时可以看到第二个可执行点的 **view**。（每个可执行点以 `digraph{};` 为单位）


```
digraph getColumn_37_1_SourceCodeLocation_java {
    getColumn_START[label = "[Start:37:1]getColumn_START ", shape = octagon]
    node38_2[label = "[38:2]return column ", shape = box]
    getColumn_END[label = "[End:37:1]getColumn_END ", shape = octagon]
    getColumn_START->node38_2
    node38_2->getColumn_END
};

digraph getFileUnitName_41_1_SourceCodeLocation_java {
    getFileUnitName_START[label = "[Start:41:1]getFileUnitName_START ", shape = octagon]
    node42_2[label = "[42:2]return fileUnitName ", shape = box]
    getFileUnitName_END[label = "[End:41:1]getFileUnitName_END ", shape = octagon]
    getFileUnitName_START->node42_2
    node42_2->getFileUnitName_END
};


digraph toString_45_1_SourceCodeLocation_java {
    toString_START[label = "[Start:45:1]toString_START ", shape = octagon]
    node47_2[label = "[47:2]return '' + lineNumber + LINE_COLUMN_SPLITTER+ column ", shape = box]
    toString_END[label = "[End:45:1]toString_END ", shape = octagon]
    toString_START->node47_2
    node47_2->toString_END
};
```

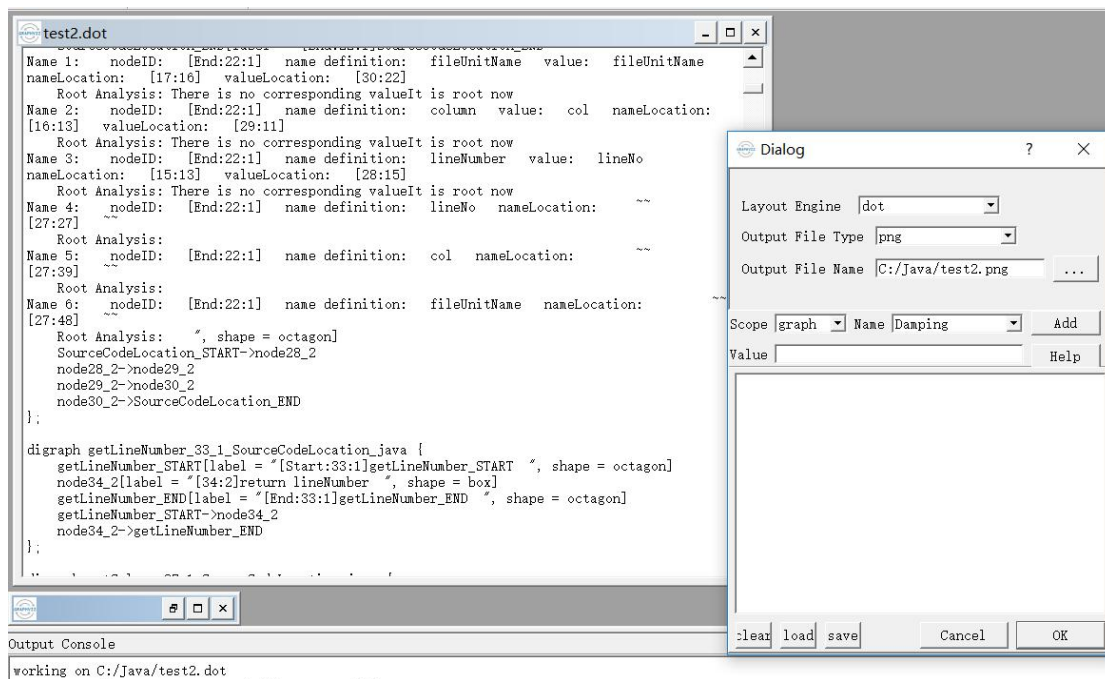
此外，GVedit 还有一个强大的功能就是根据.dot 导出图片。即 **view** 作为图片被保存到电脑



中。点击这个即可。点击以后会弹出 **Dialog**，导出什么格式的图片以及导出的图片存放在哪里由自己设定。设定好了点击 **OK** 即可。



注意!!! 和之前一样，第一次点击 **OK** 如果报错了，那就再选择再点击一次 **OK**。第二次一定会生成对应的图片。



最大定值到达分析的图会比较丰富，相比定值到达分析的话。（因为最大定值到达分析的可执行点规模很大，比普通的定值到达分析的第一个可执行点往往要大）

展示的时候建议着重展示最大定值到达分析的可视化。

原理是一样的，运行完最大定值到达分析以后，关闭 **GVEdit** 再重新打开，选择新的 **.dot**。然后导出图片以及在 **GVEdit** 软件中看 **view** 的操作都是和之前一样的。最大定值到达分析的图片显然更加复杂。

