

## Points-to Analysis (<https://goo.gl/0QIqeI>)

```
module PointsToAnalysis import ControlFlowAnalysis

/**
  Returns the points-to tuple targets for a variable before the execution of CFG node 'n'.

  @param n the CFG node
  @param u the pointer variable
  @return the pointed-to variable
 */
def pointsToBefore(n : ICFGNode, u : IVariableDeclaration) : IVariableDeclaration = {
  b := cfg(n)
  return pointsToAfter(b, u)
}

/**
  Returns the points-to tuple targets for a variable after the execution of CFG node 'n'.

  @param n the CFG node
  @param u the pointer variable
  @return the pointed-to variable
 */
def pointsToAfter(n : ICFGNode, u : IVariableDeclaration) : IVariableDeclaration = {
  // no new binding at the current node 'n'
  assert undef pointsToAt(n)
  return pointsToBefore(n, u)
} alt {
  // there is a binding at the current node but it does not affect 'u'
  (x, y) := pointsToAt(n)
  assert x != u
  return pointsToBefore(n, u)
} alt {
  // there is a binding at the current node and it affects 'u'
  // and the binding does not point to the null literal
  (x, y) := pointsToAt(n)
  assert x == u
  assert y not instanceof NullExpression
  return variableInAssignmentSide_right(y)
}

/**
  Returns the points-to binding at a CFG node.
  The pointed-to element of the tuple has type IAssignmentSide, because
  pointsToAfter needs to handle the null assignment as well.

  @param n the CFG node
  @return the current binding
 */
private def pointsToAt(n : ICFGNode) : (IVariableDeclaration, IAssignmentSide) = {
  (l, r) := extractSides(n)
  u := variableInAssignmentSide_left(l)
  return (u, r)
}

/**
  Returns the potential sides of assignments at a CFG node.

  @param n the CFG node
  @return the sides of assignment(s)
 */
private def extractSides(n : ICFGNode) : (IAssignmentSide, IAssignmentSide) = {
  assert n instanceof LocalVariableDeclaration
  return (n, n.init)
} alt {
  e := extractExpression(n)
  c := extractAssignment(e)
}
```

<b>return</b> (c.left, c.right)	66
}	67
/**	68
Returns the expressions at a CFG node n.	69
	70
@param n the CFG node	71
@return the expressions at the node	72
*/	73
<b>private def</b> extractExpression(n : ICFGNode) : Expression = {	74
<b>assert</b> n <b>instanceOf</b> ExpressionStatement	75
<b>return</b> n.expr	76
} <b>alt</b> {	77
<b>assert</b> n <b>instanceOf</b> ForStatement	78
<b>return</b> n.incr	79
} <b>alt</b> {	80
<b>assert</b> n <b>instanceOf</b> WhileStatement	81
<b>return</b> n.condition	82
} <b>alt</b> {	83
<b>assert</b> n <b>instanceOf</b> DoWhileStatement	84
<b>return</b> n.condition	85
} <b>alt</b> {	86
<b>assert</b> n <b>instanceOf</b> IfStatement	87
<b>return</b> n.condition	88
} <b>alt</b> {	89
<b>assert</b> n <b>instanceOf</b> ElseIfPart	90
<b>return</b> n.condition	91
} <b>alt</b> {	92
<b>assert</b> n <b>instanceOf</b> SwitchStatement	93
<b>return</b> n.expression	94
} <b>alt</b> {	95
<b>assert</b> n <b>instanceOf</b> LocalVariableDeclaration	96
<b>return</b> n.init	97
}	98
	99
/**	100
Returns the assignment expression(s) in an expression node.	101
	102
@param e the expression	103
@return the assignment expression(s)	104
*/	105
<b>private def</b> extractAssignment(e : Expression) : AssignmentExpr = {	106
<b>assert</b> e <b>instanceOf</b> AssignmentExpr	107
<b>return</b> e	108
} <b>alt</b> {	109
<b>assert</b> e <b>instanceOf</b> ParensExpression	110
<b>return</b> extractAssignment(e.expression)	111
} <b>alt</b> {	112
<b>assert</b> e <b>instanceOf</b> ExpressionList	113
<b>return</b> extractAssignment(e.expressions)	114
} <b>alt</b> {	115
<b>assert</b> e <b>instanceOf</b> BinaryExpression	116
<b>return</b> extractAssignment(e.left)	117
} <b>alt</b> {	118
<b>assert</b> e <b>instanceOf</b> BinaryExpression	119
<b>return</b> extractAssignment(e.right)	120
} <b>alt</b> {	121
<b>assert</b> e <b>instanceOf</b> TernaryExpression	122
<b>return</b> extractAssignment(e.condition)	123
} <b>alt</b> {	124
<b>assert</b> e <b>instanceOf</b> TernaryExpression	125
<b>return</b> extractAssignment(e.thenExpr)	126
} <b>alt</b> {	127
<b>assert</b> e <b>instanceOf</b> TernaryExpression	128
<b>return</b> extractAssignment(e.elseExpr)	129
}	130
	131
/**	132
	133

```

Returns the pointer variable from an assignment left hand side
based on Andersen's rules.
134
135
136
@param s the left hand side of an assignment
137
@return the pointer variable
138
*/
139
private def variableInAssignmentSide_left(s : IAssignmentSide) : IVariableDeclaration = {
140
  assert s instanceOf DerefExpr
141
  u := variableInAssignmentSide_left(s.expression)
142
  n := eval(s.ancestor<concept = ICFGNode>)
143
  v := pointsToBefore(n, u)
144
  return v
145
} alt {
146
  assert s instanceOf ParensExpression
147
  return variableInAssignmentSide_left(s.expression)
148
} alt {
149
  return variableInAssignmentSide_primitive(s)
150
}
151
152
/**
153
Returns the pointed-to variable from an assignment right hand side
154
based on Andersen's rules.
155
156
@param s the right hand side of an assignment
157
@return the pointed-to variable
158
*/
159
private def variableInAssignmentSide_right(s : IAssignmentSide) : IVariableDeclaration = {
160
  assert s instanceOf ReferenceExpr
161
  return variableInAssignmentSide_primitive(s.expression)
162
} alt {
163
  assert s instanceOf DerefExpr
164
  u := variableInAssignmentSide_right(s.expression)
165
  n := eval(s.ancestor<concept = ICFGNode>)
166
  v := pointsToBefore(n, u)
167
  return v
168
} alt {
169
  assert s instanceOf ParensExpression
170
  return variableInAssignmentSide_right(s.expression)
171
} alt {
172
  u := variableInAssignmentSide_primitive(s)
173
  n := eval(s.ancestor<concept = ICFGNode>)
174
  v := pointsToBefore(n, u)
175
  return v
176
}
177
178
/**
179
Returns the variable in an assignment side.
180
181
@param s the assignment side (left or right)
182
@return the variable
183
*/
184
private def variableInAssignmentSide_primitive(s : IAssignmentSide) : IVariableDeclaration = {
185
  assert s instanceOf GlobalVarRef
186
  return s.var
187
} alt {
188
  assert s instanceOf LocalVarRef
189
  return s.var
190
} alt {
191
  assert s instanceOf ArgumentRef
192
  return s.arg
193
} alt {
194
  assert s instanceOf LocalVariableDeclaration
195
  return s
196
}
197

```