

## Control Flow Analysis (<https://goo.gl/0QIqeI>)

```
module ControlFlowAnalysis
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

/**
Returns the node(s) in the CFG that has an edge pointing to node 'n'.

@param n the CFG target
@return the CFG source(s)
*/
def cfg(n : ICFGNode) : ICFGNode = {
s := cfg0(n)
assert s not instanceof ReturnStatement
return s
}

/**
Helper function for function cfg. This function does not
filter out return statements as illegal CFG sources.

@param n the CFG target
@return the CFG source(s)
*/
private def cfg0(n : ICFGNode) : ICFGNode = {
assert undef cannotActAsEdgeTarget(n)
return cfg_before(n)
} alt {
assert undef cannotActAsEdgeTarget(n)
return cfg_testFirstControl(n)
} alt {
return cfg_testAfterControl(n)
} alt {
return cfg_special(n)
} alt {
return cfg_continue(n)
} alt {
return cfg_break(n)
} alt {
return cfg_section(n)
}

/**
CFG edges related to continue statements
(1) node 'n' is the first loop ancestor of a continue statement 's'

@param n the CFG target loop
@return the CFG source continue statement
*/
private def cfg_continue(n : ICFGNode) : ContinueStatement = {
assert s instanceof ContinueStatement
assert n == firstLoopAncestor(s)
return s
}

/**
CFG edges related to break statements
(1) there exists a break statement 's', its first loop ancestor is 'l'
and the first CFG node after 'l' is 'n': 's' -> 'n'

@param n the CFG target
@return the CFG source break statement
*/
private def cfg_break(n : ICFGNode) : BreakStatement = {
assert s instanceof BreakStatement
l := firstLoopAncestor(s)
a := directlyAfter(l)
assert n == firstNodes(a)
}
```

```

    return s
}
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133

/**
CFG edges related to else if parts and switch branches.

@param n the CFG target
@return the CFG source
*/
private def cfg_special(n : ICFGNode) : ICFGNode = {
    return cfg_elseIf(n)
} alt {
    return cfg_switchCase(n)
}

/**
CFG edges related to test first control statements.

@param n the CFG target
@return the CFG source
*/
private def cfg_testFirstControl(n : ICFGNode) : ICFGNode = {
    s := cfg_while(n)
    assert s not instanceof ContinueStatement
    return s
} alt {
    s := cfg_for(n)
    assert s not instanceof ContinueStatement
    return s
} alt {
    s := cfg_if(n)
    assert s not instanceof ContinueStatement
    return s
} alt {
    s := cfg_switch(n)
    assert s not instanceof ContinueStatement
    return s
}

/**
CFG edges related to test after control statements.

@param n the CFG target
@return the CFG source
*/
private def cfg_testAfterControl(n : ICFGNode) : ICFGNode = {
    s := cfg_doWhile(n)
    assert s not instanceof ContinueStatement
    return s
}

/**
CFG edges produced through the syntactic precedence relationship.
(1) node 'b' is directly before node 's' and node 'l' is a last CFG node of 'b': 'l' -> 's'

@param s the CFG target
@return the CFG source
*/
private def cfg_before(s : ICFGNode) : ICFGNode = {
    b := directlyBefore(s)
    l := lastNodes(b)
    assert l not instanceof ContinueStatement
    return l
}

/**
We call a statement headless if the very first CFG node in the context of the statement
is not the statement itself, rather some other contained CFG node.

```

```

The function returns the headless parent of a statement 's', if any.
134
135
@param s the contained statement
136
@return the container headless statement
137
*/
138
private def cfg_headless_parent(s : Statement) : ICFGNode = {
139
  assert undef directlyBefore(s)
140
  p := parent(s)
141
  assert p instanceof StatementList
142
  d := parent(p)
143
  assert d instanceof DoWhileStatement
144
  return d
145
} alt {
146
  assert undef directlyBefore(s)
147
  l := parent(s)
148
  assert l instanceof StatementList
149
  p := parent(l)
150
  assert eval(!CFGHelper.hasStatementList(p))
151
  return l
152
}
153
154
/**
155
  Returns the CFG source(s) of statement 's' after skipping all
156
  headless ancestors.
157
158
  @param s the CFG target
159
  @return the CFG source
160
  */
161
private def cfg_headless_traversal(s : Statement) : ICFGNode = {
162
  p := cfg_headless_parent(s)
163
  return cfg_before(p)
164
} alt {
165
  p := cfg_headless_parent(s)
166
  return cfg_testFirstControl(p)
167
} alt {
168
  p := cfg_headless_parent(s)
169
  return cfg_headless_traversal(p)
170
}
171
172
/**
173
  CFG edges related to sections.
174
175
  @param s the CFG target
176
  @return the CFG source
177
  */
178
private def cfg_section(s : Statement) : ICFGNode = {
179
  assert undef cannotActAsEdgeTarget(s)
180
  return cfg_headless_traversal(s)
181
}
182
183
/**
184
  CFG edges related to do while statements
185
  (1) there is a do while statement 'd' whose first CFG node is 's': 'd' -> 's'
186
  (2) CFG source(s) obtained after skipping all headless ancestors.
187
  (3) 's' is a do while statement and 'l' is a last node in 's': 'l' -> 's'
188
189
  @param s the statement in the context of a do while statement
190
  @return the CFG source
191
  */
192
private def cfg_doWhile(s : Statement) : ICFGNode = {
193
  assert undef cannotActAsEdgeTarget(s)
194
  assert d instanceof DoWhileStatement
195
  assert s == firstNodes(d)
196
  return d
197
} alt {
198
  assert undef cannotActAsEdgeTarget(s)
199
  return cfg_headless_traversal(s)
200
} alt {
201

```

```

assert s instanceOf DoWhileStatement
i := lastStatementInDoWhile(s)
l := lastNodes(i)
assert l not instanceOf BreakStatement
return l
}

/**
CFG edges related to if statements
(1) statement 's' is the first statement in the body of an if statement 'e' or
    in the body of an else if part 'e': 'e' -> 's'
(2) statement 's' is the first statement in the body of an else part 'e' and the container
    if statement's last else if part is 'f': 'f' -> 's'
(3) statement 's' is the first statement in the body of an else part 'e' and the container
    if statement 'i' has no else if part: 'i' -> 's'

@param s the statement in the context of a switch statement
@return the CFG source
*/
private def cfg_if(s : Statement) : ICFGNode = {
assert undef directlyBefore(s)
p := parent(s)
assert p instanceOf StatementList
e := parent(p)
assert eval(e.isInstanceOf(IfStatement) || e.isInstanceOf(ElseIfPart))
return e
} alt {
assert undef directlyBefore(s)
p := parent(s)
assert p instanceOf StatementList
e := parent(p)
assert e instanceOf ElsePart
i := parent(e)
assert i instanceOf IfStatement
return lastElseIfInIfStatement(i)
} alt {
assert undef directlyBefore(s)
p := parent(s)
assert p instanceOf StatementList
e := parent(p)
assert e instanceOf ElsePart
i := parent(e)
assert i instanceOf IfStatement
assert undef lastElseIfInIfStatement(i)
return i
}

/**
CFG edges related to switch statements
(1) statement 's' is the first statement in the body of a case branch 'c': 'f' -> 's'
(2) statement 's' is the first statement in the body of a default branch 'd' and the container
    switch statement is 'c': 'c' -> 's'
(3) statement 's' is the first statement in the body of a branch and the control
    falls through from the last statement(s) 'l' of the previous branch: 'l' -> 's'

@param s the statement in the context of a switch statement
@return the CFG source
*/
private def cfg_switch(s : Statement) : ICFGNode = {
assert undef directlyBefore(s)
p := parent(s)
assert p instanceOf StatementList
c := parent(p)
assert c instanceOf SwitchCase
return c
} alt {
assert undef directlyBefore(s)
p := parent(s)

```

```

    assert p instanceof StatementList
    d := parent(p)
    assert d instanceof SwitchDefault
    c := parent(d)
    assert c instanceof SwitchStatement
    return c
} alt {
    assert undef directlyBefore(s)
    p := parent(s)
    assert p instanceof StatementList
    m := parent(p)
    assert m instanceof SwitchMember
    pm := directlyBefore(m)
    i := lastStatementInSwitchCase(pm)
    l := lastNodes(i)
    assert l not instanceof BreakStatement
    return l
}

/**
CFG edges related to while statements
(1) statement 's' is the first statement in the body of a while statement 'f': 'f' -> 's'
(2) 's' is a while statement and 'l' is a last statement in its body: 'l' -> 's'

@param s the statement in the context of a while statement
@return the CFG source
*/
private def cfg_while(s : Statement) : ICFGNode = {
    assert undef directlyBefore(s)
    p := parent(s)
    assert p instanceof StatementList
    f := parent(p)
    assert f instanceof WhileStatement
    return f
} alt {
    assert s instanceof WhileStatement
    i := lastStatementInFor(s)
    l := lastNodes(i)
    assert l not instanceof BreakStatement
    return l
}

/**
CFG edges related to for statements.
(1) statement 's' is the first statement in the body of a for statement 'f': 'f' -> 's'
(2) 's' is a for statement and 'l' is a last statement in its body: 'l' -> 's'

@param s the statement in the context of a for statement
@return the CFG source
*/
private def cfg_for(s : Statement) : ICFGNode = {
    assert undef directlyBefore(s)
    p := parent(s)
    assert p instanceof StatementList
    f := parent(p)
    assert f instanceof ForStatement
    return f
} alt {
    assert s instanceof ForStatement
    i := lastStatementInFor(s)
    l := lastNodes(i)
    assert l not instanceof BreakStatement
    return l
}

/**
Returns the CFG source of an else if part.
(1) the direct predecessor else if part

```

```

(2) the container if statement if (1) is rejected
338
@param e the else if part
339
@return the CFG source
340
*/
341
private def cfg_elif(e : ElseIfPart) : ICFGNode = {
342
  b := directlyBefore(e)
343
  assert b instanceof ElseIfPart
344
  return b
345
} alt {
346
  assert undef directlyBefore(e)
347
  s := parent(e)
348
  assert s instanceof IfStatement
349
  return s
350
}
351
352
/**
353
  Returns the CFG source of a switch case branch.
354
  The CFG source if the container switch statement.
355
356
@param c the switch case branch
357
@return the CFG source
358
*/
359
private def cfg_switchCase(c : SwitchCase) : ICFGNode = {
360
  s := parent(c)
361
  assert s instanceof SwitchStatement
362
  return s
363
}
364
365
/**
366
  Returns the CFG node which can act as the first node in a given CFG node.
367
  First means that in the context of a given node this node represents
368
  the first control flow point during execution.
369
370
371
@param n the CFG node
372
@return the first node
373
*/
374
private def firstNodes(n : ICFGNode) : ICFGNode = {
375
  assert eval(!n.isInstanceOf(DoWhileStatement))
376
  return n
377
} alt {
378
  assert n instanceof DoWhileStatement
379
  f := firstStatementInDoWhile(n)
380
  return firstNodes(f)
381
}
382
383
/**
384
  Returns the CFG node(s) which can act as the last node(s) in a given CFG node.
385
  Last means that in the context of a given node these nodes represent
386
  the final control flow points during execution.
387
388
389
@param n the CFG node
390
@return the last node(s)
391
*/
392
private def lastNodes(n : ICFGNode) : ICFGNode = {
393
  // if statement is a last statement itself if it has neither else if part(s) nor else part
394
  assert n instanceof IfStatement
395
  assert undef lastElseIfInIfStatement(n)
396
  assert undef elsePartOfIf(n)
397
  return n
398
} alt {
399
  // last elseif part of if statement, if it has no else part
400
  assert n instanceof IfStatement
401
  assert undef elsePartOfIf(n)
402
  elseif := lastElseIfInIfStatement(n)
403
  return elseif
404
} alt {
405
  // last statements of if statement

```

```

assert n instanceOf IfStatement
l := lastStatementInIf(n)
return lastNodes(l)
} alt {
  // last statements of section
assert n instanceOf StatementList
l := lastStatementInStatementList(n)
return lastNodes(l)
} alt {
  // while statement itself is a last statement
assert n instanceOf WhileStatement
return n
} alt {
  // for statement itself is a last statement
assert n instanceOf ForStatement
return n
} alt {
  // do while statement itself is a last statement
assert n instanceOf DoWhileStatement
return n
} alt {
  // primitive statement itself is a last statement
assert eval(CFGHelper.isPrimitiveStatement(n))
return n
} alt {
  // last statements of switch statement
assert n instanceOf SwitchStatement
l := lastStatementInSwitch(n)
return lastNodes(l)
}

// HELPER FUNCTIONS

/**
Enumerates CFG nodes which cannot act as CFG edge targets
from syntactically before the node.

@param n the node
*/
def cannotActAsEdgeTarget(n : ICFGNode) : << ... >> = {
assert n instanceOf DoWhileStatement
} alt {
assert n instanceOf StatementList
}

/**
Returns the first loop-like ancestor of a node.

@param child the child node
@return the loop ancestor
*/
def firstLoopAncestor(child : IWithParentPointer) : IParentPointerTarget = {
parent := parent(child)
assert eval(CFGHelper.isLoopStatement(parent))
return parent
} alt {
parent := parent(child)
assert eval(!CFGHelper.isLoopStatement(parent))
return firstLoopAncestor(parent)
}

/**
Returns the last statement(s) of an if statement.

@param s the if statement
@return the last statement(s)
*/
private def lastStatementInIf(s : IfStatement) : Statement = {

```

b := s.thenPart	474
return lastStatementInStatementList(b)	475
} alt {	476
b := s.elseIfs.body	477
return lastStatementInStatementList(b)	478
} alt {	479
b := s.elsePart.body	480
return lastStatementInStatementList(b)	481
}	482
	483
/**	484
Returns the last statement of a do while statement.	485
	486
@param s the do while statement	487
@return the last statement	488
*/	489
private def lastStatementInDoWhile(s : DoWhileStatement) : Statement = {	490
b := s.body	491
return lastStatementInStatementList(b)	492
}	493
	494
/**	495
Returns the first statement of a do while statement.	496
	497
@param s the do while statement	498
@return the first statement	499
*/	500
private def firstStatementInDoWhile(s : DoWhileStatement) : Statement = {	501
b := s.body	502
return firstStatementInStatementList(b)	503
}	504
	505
/**	506
Returns the last statement of a while statement.	507
	508
@param s the while statement	509
@return the last statement	510
*/	511
private def lastStatementInWhile(s : WhileStatement) : Statement = {	512
b := s.body	513
return lastStatementInStatementList(b)	514
}	515
	516
/**	517
Returns the last statement of a for statement.	518
	519
@param s the for statement	520
@return the last statement	521
*/	522
private def lastStatementInFor(s : ForStatement) : Statement = {	523
b := s.body	524
return lastStatementInStatementList(b)	525
}	526
	527
/**	528
Returns the last statement(s) of a switch statement.	529
	530
@param s the switch statement	531
@return the last statement(s)	532
*/	533
private def lastStatementInSwitch(s : SwitchStatement) : Statement = {	534
c := s.cases	535
assert c instanceof SwitchCase	536
return lastStatementInSwitchCase(c)	537
} alt {	538
d := s.cases	539
assert d instanceof SwitchDefault	540
return lastStatementInSwitchDefault(d)	541



}	542
	543
/**	544
Returns the last statement of a switch case branch.	545
	546
@param c the case branch	547
@return the last statement	548
*/	549
<b>private def</b> lastStatementInSwitchCase(c : SwitchCase) : Statement = {	550
b := c.body	551
<b>return</b> lastStatementInStatementList(b)	552
}	553
	554
/**	555
Returns the last statement of a switch default branch.	556
	557
@param d the default branch	558
@return the last statement	559
*/	560
<b>private def</b> lastStatementInSwitchDefault(d : SwitchDefault) : Statement = {	561
b := d.body	562
<b>return</b> lastStatementInStatementList(b)	563
}	564
	565
/**	566
Returns the else part of an if statement.	567
	568
@param s the if statement	569
@return the else part	570
*/	571
<b>private def</b> elsePartOfIf(s : IfStatement) : ElsePart = {	572
<b>return</b> s.elsePart	573
}	574
	575
/**	576
Returns the last statement in a statement list.	577
	578
@param l the statement list	579
@return the last statement	580
*/	581
<b>private def</b> lastStatementInStatementList(l : StatementList) : Statement = {	582
s := l.statements	583
<b>assert undef</b> directlyAfter(s)	584
<b>return</b> s	585
}	586
	587
/**	588
Returns the first statement in a statement list.	589
	590
@param l the statement list	591
@return the first statement	592
*/	593
<b>private def</b> firstStatementInStatementList(l : StatementList) : Statement = {	594
s := l.statements	595
<b>assert undef</b> directlyBefore(s)	596
<b>return</b> s	597
}	598
	599
/**	600
Returns the last else if part of an if statement.	601
	602
@param s the if statement	603
@return the last else if part	604
*/	605
<b>private def</b> lastElseIfInIfStatement(s : IfStatement) : ElseIfPart = {	606
elseif := s.elseIfs	607
<b>assert undef</b> directlyAfter(elseif)	608
<b>return</b> elseif	609

}	610
/**	611
Returns the parent node of a child node.	612
	613
@param child the child node	614
@return the parent node	615
*/	616
<b>private def</b> parent(child : IWithParentPointer) : IParentPointerTarget = {	617
<b>return</b> child.parent_wb	618
}	619
	620
/**	621
Returns the node which is syntactically before a given node.	622
	623
@param trg the successor node	624
@return the predecessor node	625
*/	626
<b>private def</b> directlyBefore(trg : INextPointerTarget) : IWithNextPointer = {	627
<b>assert</b> src <b>instanceOf</b> IWithNextPointer	628
<b>assert</b> trg == src.next_wb	629
<b>return</b> src	630
}	631
	632
/**	633
Returns the node which is syntactically after a given node.	634
	635
@param src the predecessor node	636
@return the successor node	637
*/	638
<b>def</b> directlyAfter(src : IWithNextPointer) : INextPointerTarget = {	639
<b>return</b> src.next_wb	640
}	641
	642