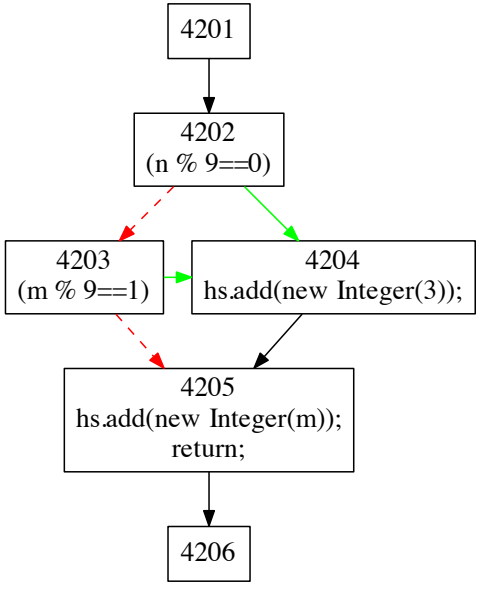
Decompiling Boolean Expressions from Java™ Bytecode

Mangala Gowri Nanda (IBM-IRL)
and S. Arun-Kumar (IIT Delhi)

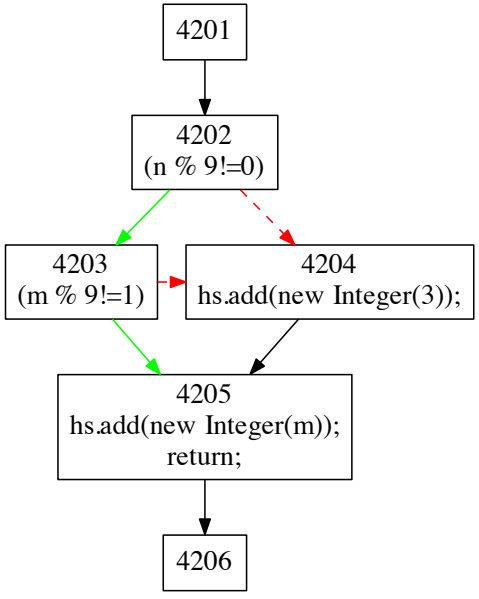
Motivation

- Generating executable code from program slices.
- Java bytecode does not preserve program structure, especially for complex boolean expressions.
- Try to reconstruct boolean expression (equivalent to the original) in terms of `&&`, `||` and the ternary `if-then-else`.
- `goto` is a four-letter word (so is `break`).

Equivalent CFGs for a simple OR clause

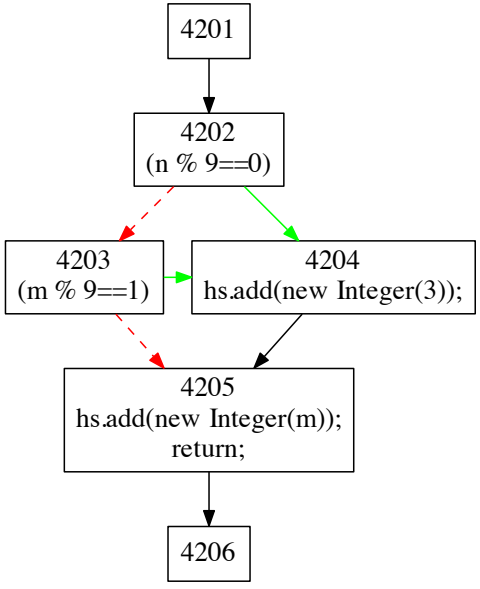


(a) The Classic OR



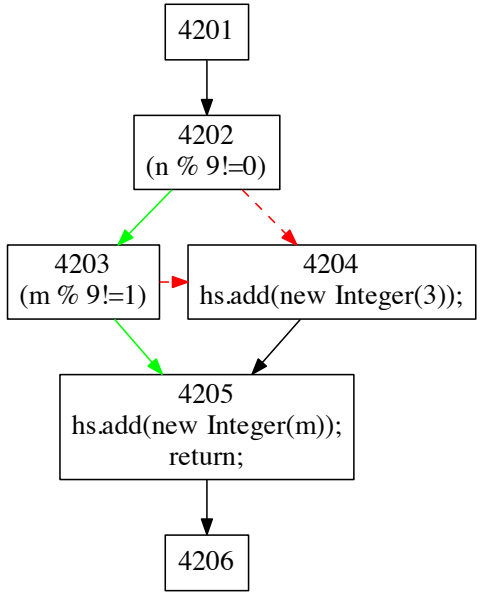

```
if(n%9==0) goto 4204
  else goto 4203
4203: if(m%9==1) goto 4204
  else goto 4205

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```

Equivalent CFGs for a simple OR clause



(a) The Classic OR

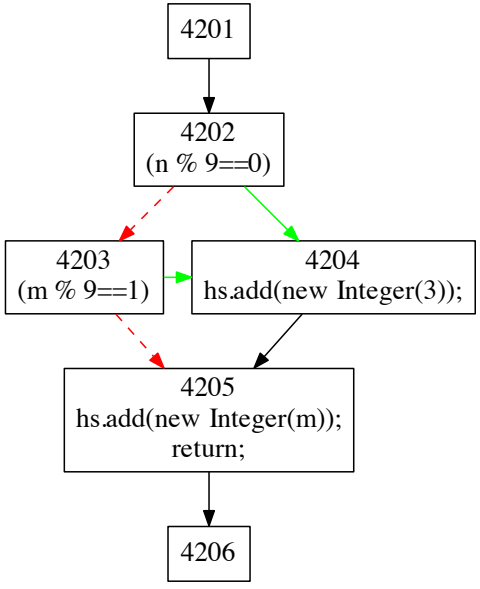
```
if(n%9==0) goto 4204
  else goto 4203
4203: if(m%9==1) goto 4204
  else goto 4205

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```



(b) The AND equivalent

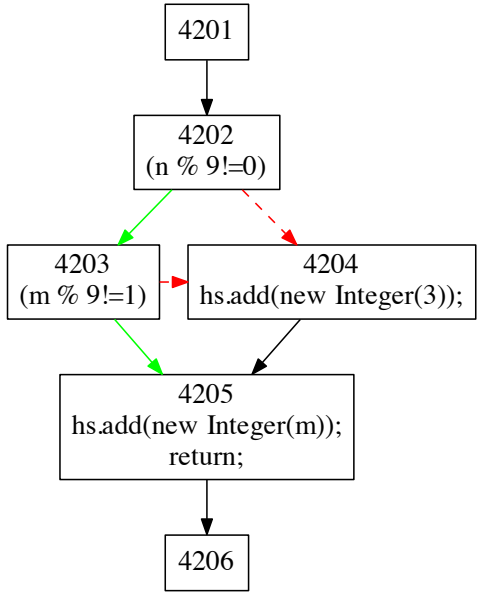
```
if(n%9!=0 && m%9!=1){}
 else {
  hs.add(new Int(3))
 }
hs.add(new Int(m));
return;
```

Equivalent CFGs for a simple OR clause



(a) The Classic OR

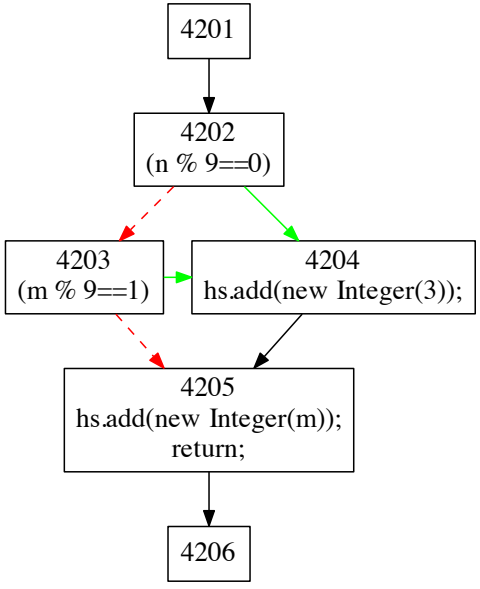
```
if(n%9==0) goto 4204
  else goto 4203
4203: if(m%9==1) goto 4204
  else goto 4205

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```



(b) The AND equivalent

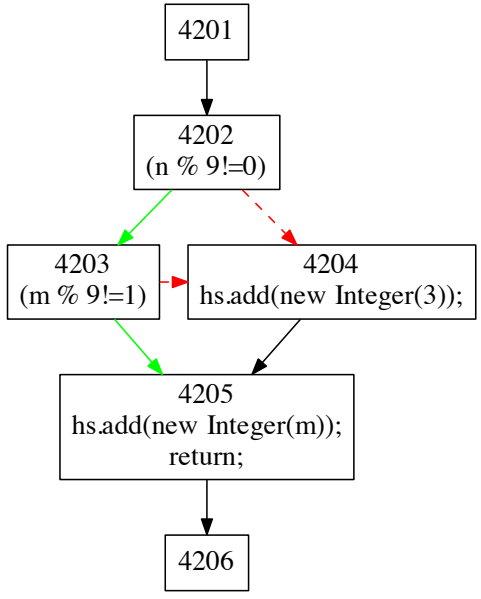
```
if(n%9!=0 && m%9!=1){}
 else {
  hs.add(new Int(3))
 }
hs.add(new Int(m));
return;
```

Equivalent CFGs for a simple OR clause



(a) The Classic OR

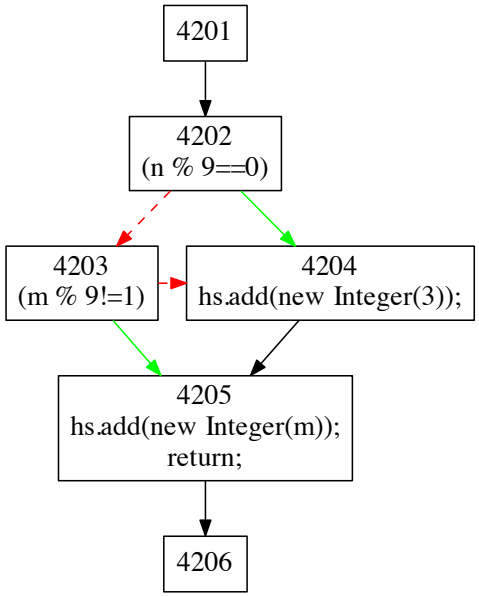
```
if(n%9==0) goto 4204
  else goto 4203
4203: if(m%9==1) goto 4204
  else goto 4205

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```



(b) The AND equivalent

```
if(n%9!=0 && m%9!=1){}
 else {
  hs.add(new Int(3))
 }
hs.add(new Int(m));
return;
```

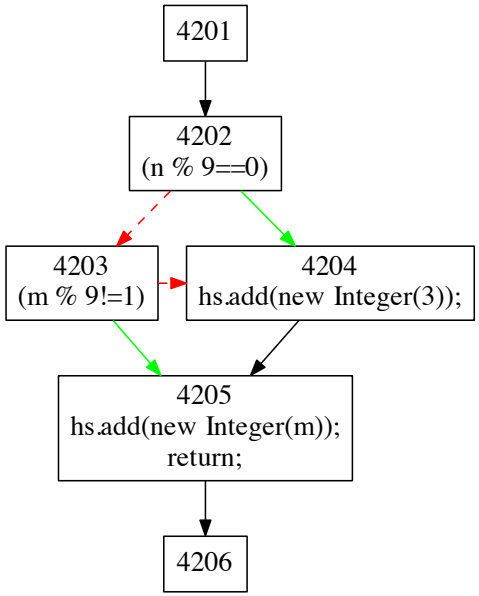
Equivalent CFGs for a simple OR clause

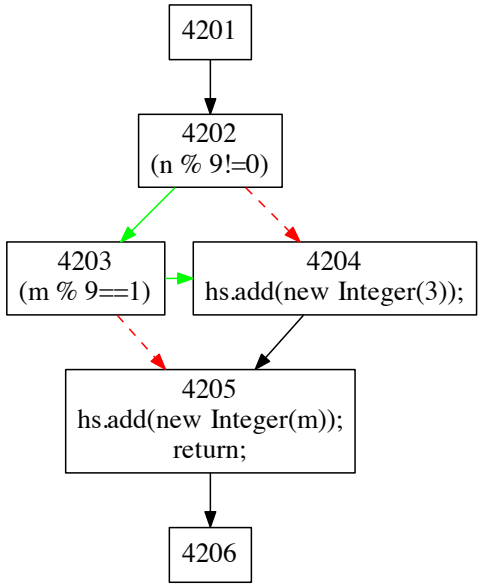


(c) An alternative

```
if(n%9==0) goto 4204
  else goto 4203
4203: if(m%9!=1) goto 4205
  else goto 4204

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```

Equivalent CFGs for a simple OR clause

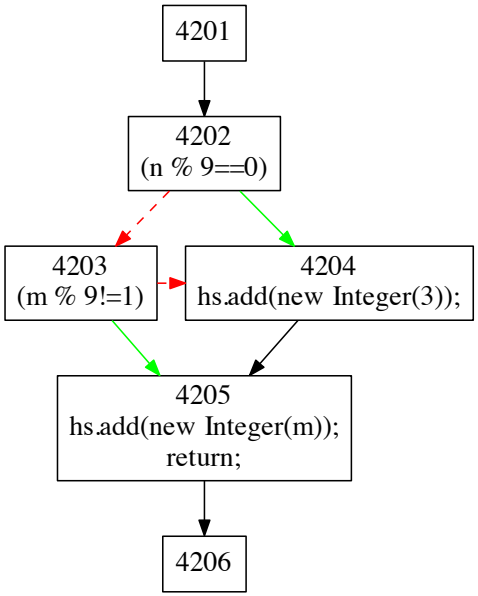




(c) An alternative

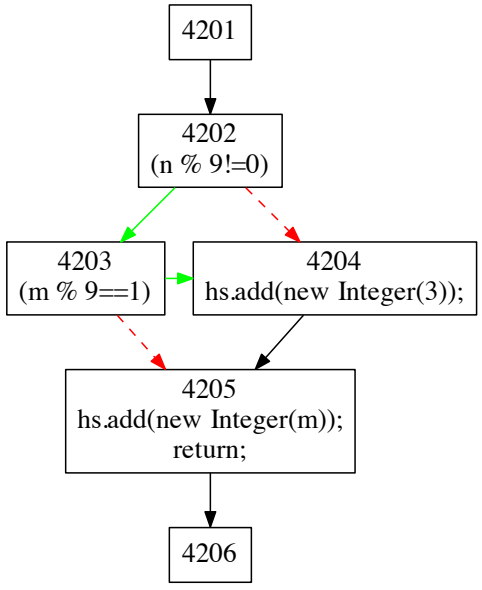
```
if(n%9==0) goto 4204
  else goto 4203
4203: if(m%9!=1) goto 4205
  else goto 4204

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```

Equivalent CFGs for a simple OR clause



(c) An alternative

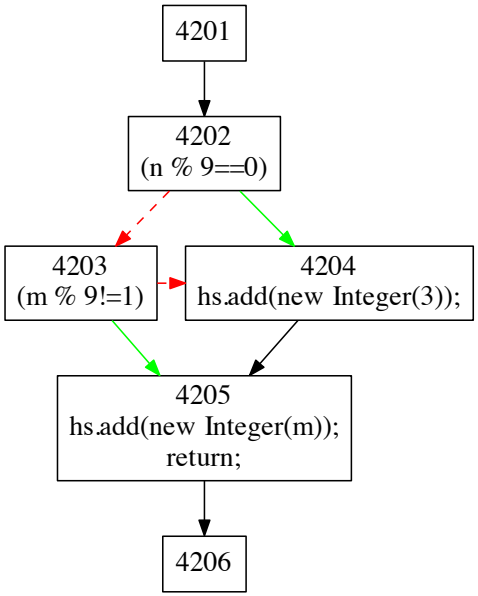
```
if(n%9==0) goto 4204
  else goto 4203
4203: if(m%9!=1) goto 4205
  else goto 4204

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```



(d) Yet Another Alternative

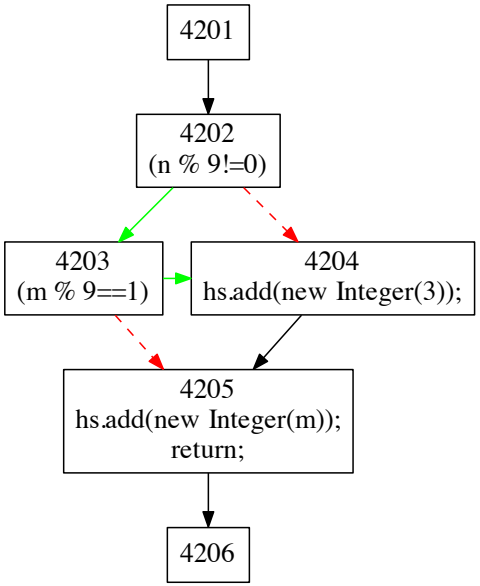
```
if(n%9!=0) goto 4203
  else goto 4204
4203: if(m%9==1) goto 4204
  else goto 4205

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```

Equivalent CFGs for a simple OR clause



(c) An alternative

```
if(n%9==0) goto 4204
  else goto 4203
4203: if(m%9!=1) goto 4205
  else goto 4204

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```



(d) Yet Another Alternative

```
if(n%9!=0) goto 4203
  else goto 4204
4203: if(m%9==1) goto 4204
  else goto 4205

4204: hs.add(new Int(3));
goto 4205
4205: hs.add(new Int(m));

return;
```

Outline

Outline

Outline

Outline

The Monochromatic Theorem: A Lemma

The language of non-negative conditional expressions is defined by the BNF

$$c ::= a \mid c_1 \;\&\&\; c_2 \mid c_1 \parallel c_2 \mid c_0?c_1 : c_2 \mid (c_0?i1 : i2) == val$$

$$\mid (c_0?o1 : o2).\text{boolfunc}()$$

Lemma

For this language of non-negative conditional expressions, every CFG generated for the program segment $\texttt{if } c \texttt{ then } S_{true} \texttt{ else } S_{false}$ *may be transformed into one that preserves the property that all incoming edges to any node in the CFG are of the same color.*

Pushing Negation Inwards

Lemma

The Boolean Identities

$$!!c' = c' \quad , \quad !(c_0?c_1 : c_2) = !c_0?!c_2 :!c_1$$
$$!(c_1 \&\& c_2) = !c_1 \| !c_2 \quad , \quad !(c_1 \| c_2) = !c_1 \&\& !c_2$$

- Negations of all comparison operators are also available in non-negative form
- Negation "twists" the subgraphs
- Use negation (recursively) to ensure that incoming arcs are of the same color.

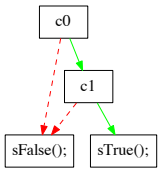
The Monochromatic Theorem

Theorem

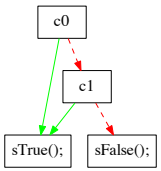
For each `basic block` that participates in a boolean expression, all the incoming edges must be the same color. That is, all incoming edges are either `true edges` or they are `false edges`, or in the case of certain ternary clauses, they may be unconditional edges (`black edges`).

If they are not, simply twist them using the boolean identities.

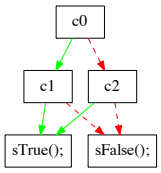
Constructing the boolean expressions



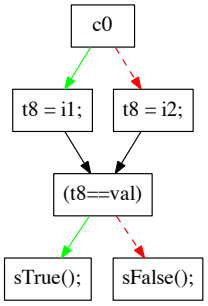
(a) c0 && c1



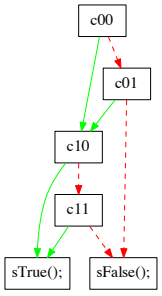
(b) c0 || c1



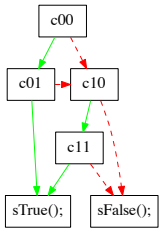
(c) c0 ? c1 : c2



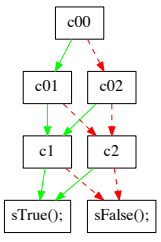
(d) (c0 ? i1 : i2) == val



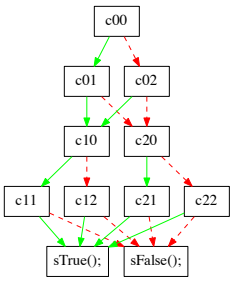
(e) (c00 || c01) && (c10 || c11)



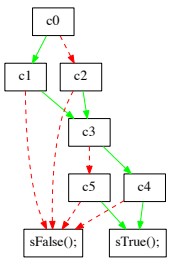
(f) (c00 && c01) || (c10 && c11)



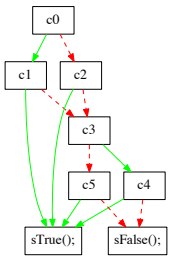
(g) (c00 ? c01 : c02) ? c1 : c2



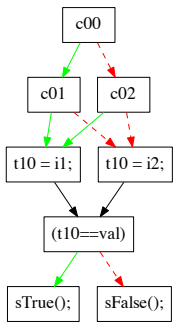
(h) (c00 ? c01 : c02) ? (c10 ? c11 : c12): (c20 ? c21 : c22)



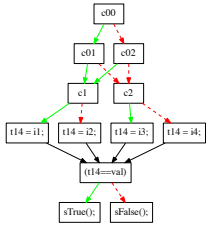
(i) (c0 ? c1 : c2) && (c3 ? c4 : c5)



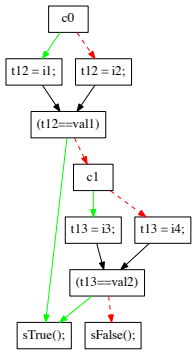
(j) (c0 ? c1 : c2) || (c3 ? c4 : c5)



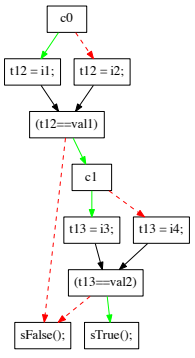
(k) ((c00 ? c01 : c02) ? i1 : i2) == val



(l) ((c00 ? c01 : c02) ? (c1 ? i1 : i2) : (c2 ? i3 : i4)) == val



(m) (c0 ? i1 : i2) == val1 && (c1 ? i3 : i4) == val2

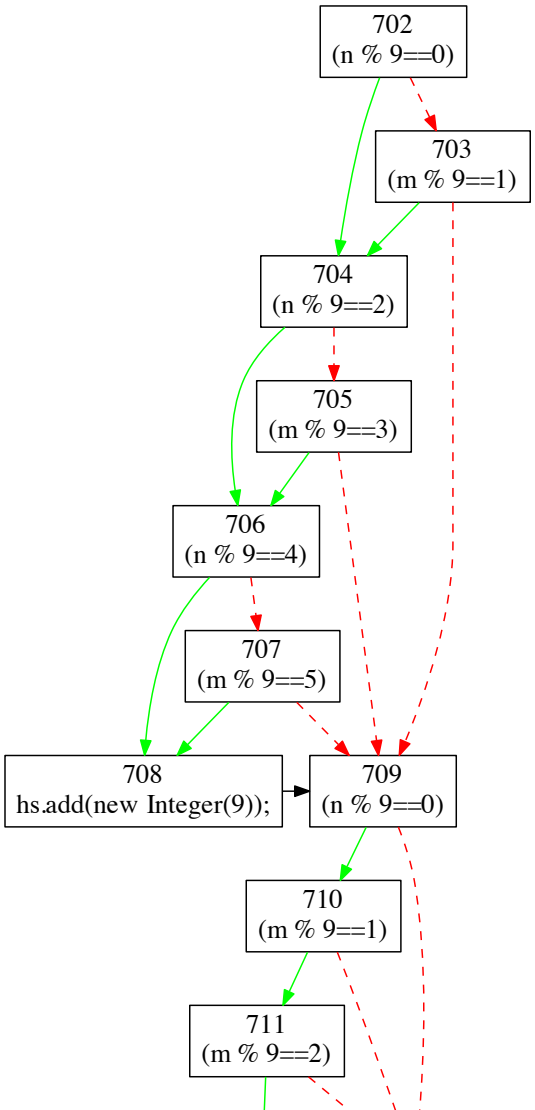


(n) (c0 ? i1 :i2) == val1 || (c1? i3 :i4) == val2

An example with only ANDs and ORs

Example code

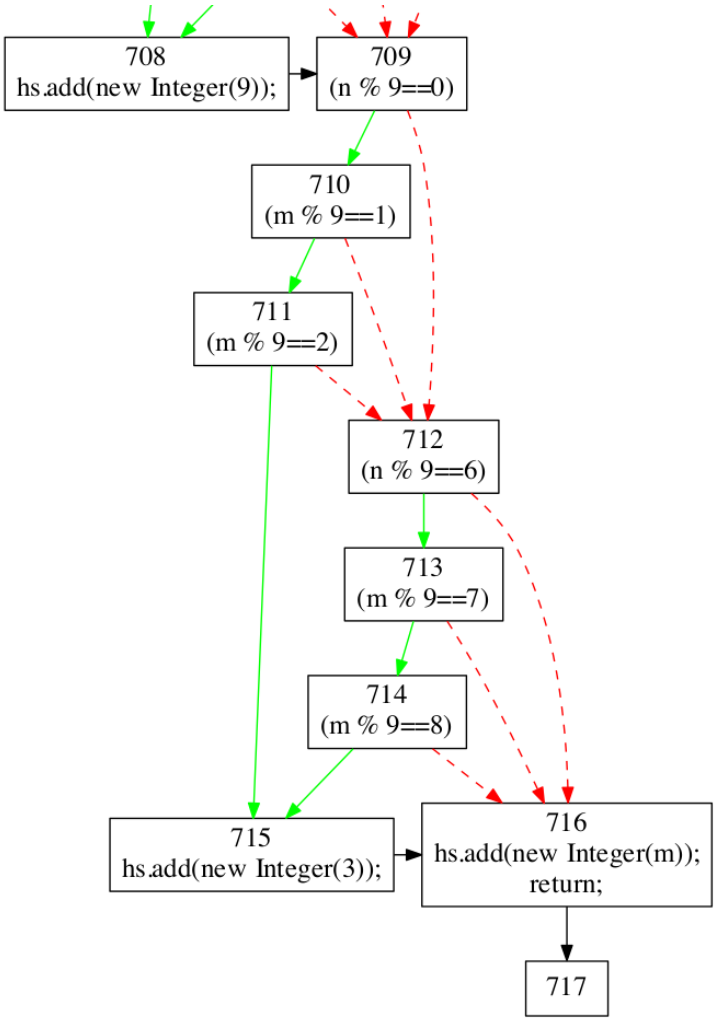
```
void andor(Vector hs, int m, int n) {
  if ( (n % 9 == 0 || m % 9 == 1) &&
    (n % 9 == 2 || m % 9 == 3) &&
    (n % 9 == 4 || m % 9 == 5) ) {
  hs.add(new Integer(9));
}
  if ( (n % 9 == 0 && m % 9 == 1 && m % 9 == 2)
    || (n % 9 == 6 && m % 9 == 7 && m % 9 == 8) ) {
    hs.add(new Integer(3));
  }
  hs.add(new Integer(m));
}
```



An example – continued

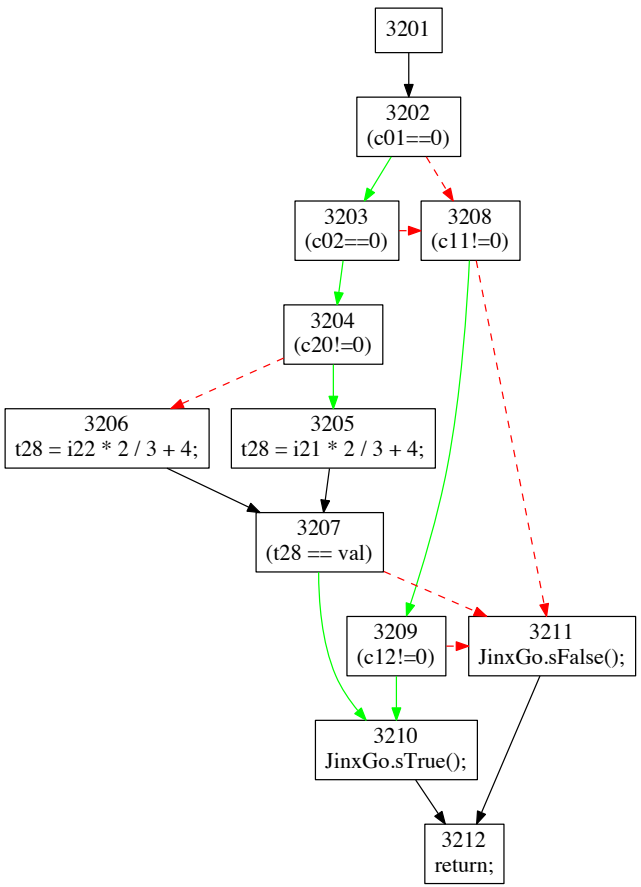
Example code

```
void andor(Vector hs, int m, int n) {
  if ( (n % 9 == 0 || m % 9 == 1) &&
    (n % 9 == 2 || m % 9 == 3) &&
    (n % 9 == 4 || m % 9 == 5) ) {
  hs.add(new Integer(9));
}
  if ( (n % 9 == 0 && m % 9 == 1 && m % 9 == 2)
    || (n % 9 == 6 && m % 9 == 7 && m % 9 == 8) ) {
    hs.add(new Integer(3));
  }
  hs.add(new Integer(m));
}
```



An example with ternary expressions

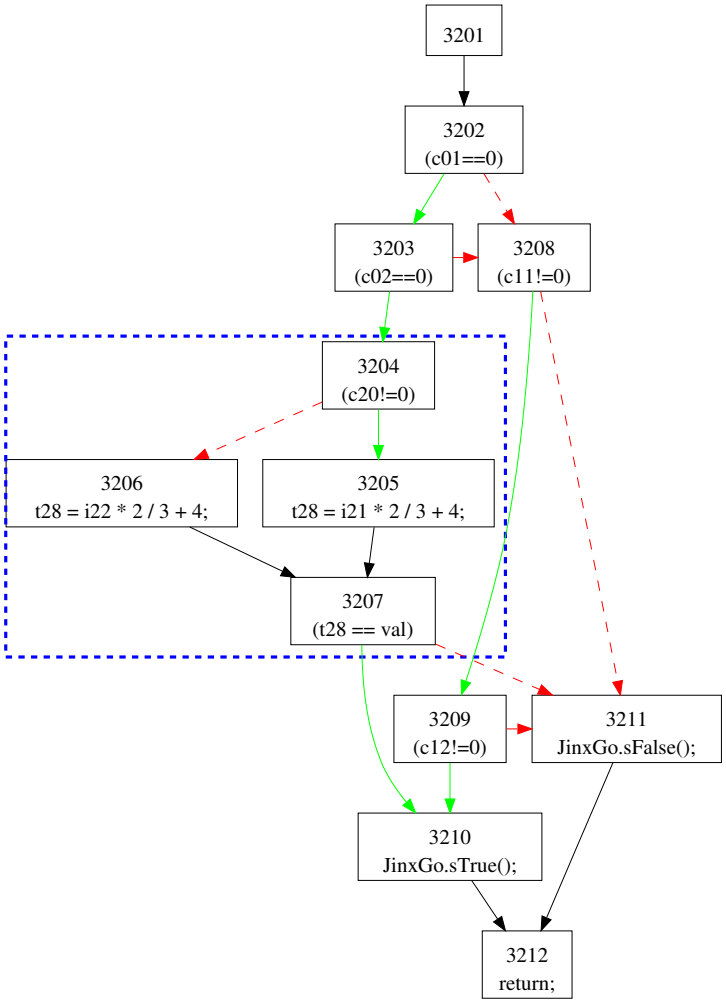
The input program: if $((c01||c02)?(c11\&\&c12) : (c20?i21*2/3+4 : i22*2/3+4) == val)$ sTrue(); else sFalse();



(a) The CFG generated by WALA for the given program.

An example with ternary expressions

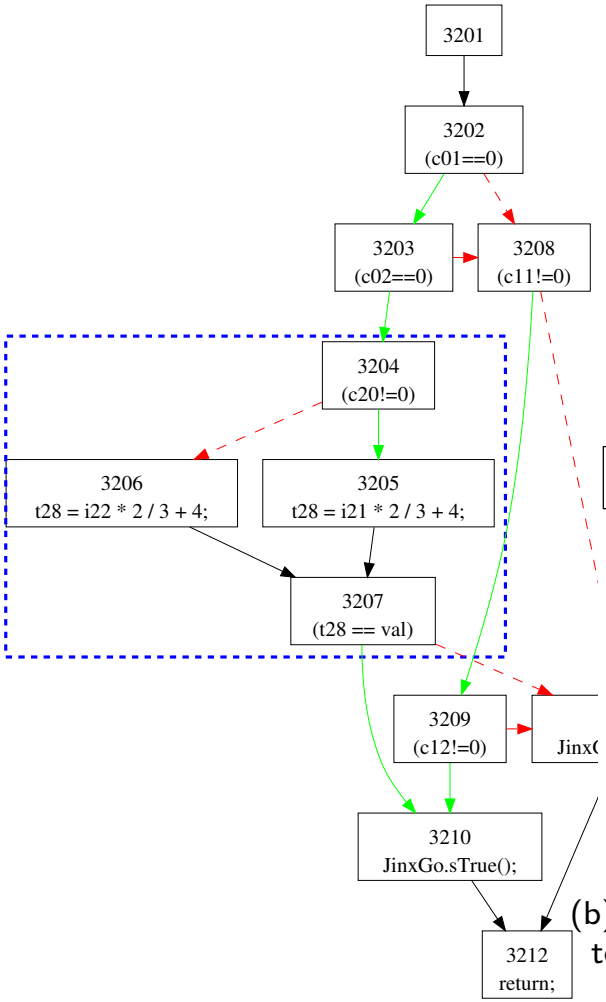
The input program: if $((c01||c02)?(c11\&\&c12) : (c20?i21*2/3+4 : i22*2/3+4) == val)$ sTrue(); else sFalse();



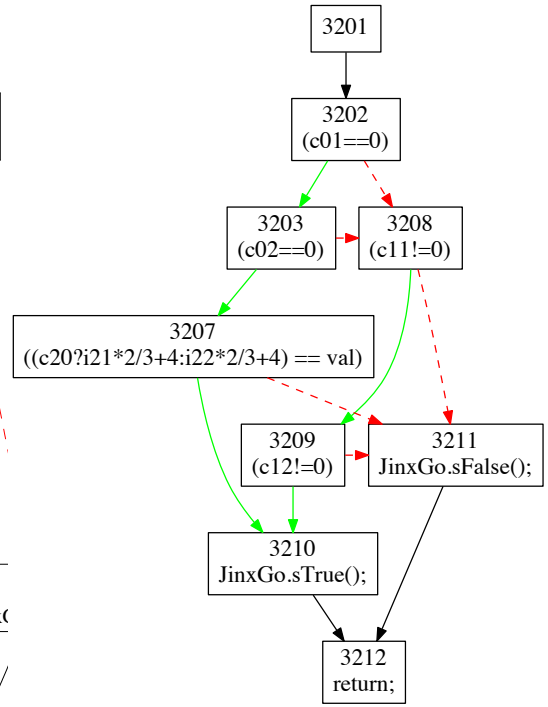
(a) The CFG generated by WALA for the given program.

An example with ternary expressions

The input program: if $((c01||c02)?(c11\&\&c12) : (c20?i21*2/3+4 : i22*2/3+4) == val)$ sTrue(); else sFalse();



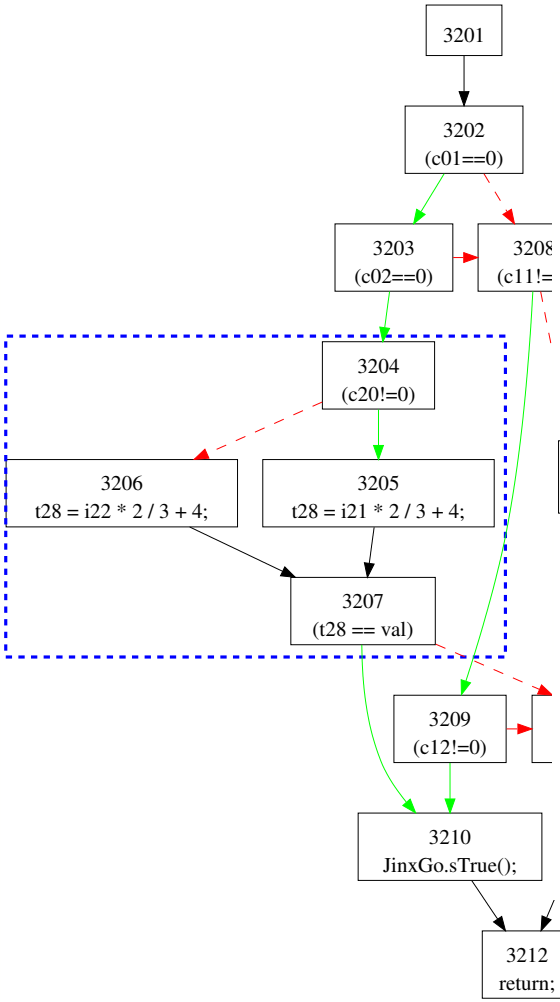
(a) The CFG generated by WALA for the given program.



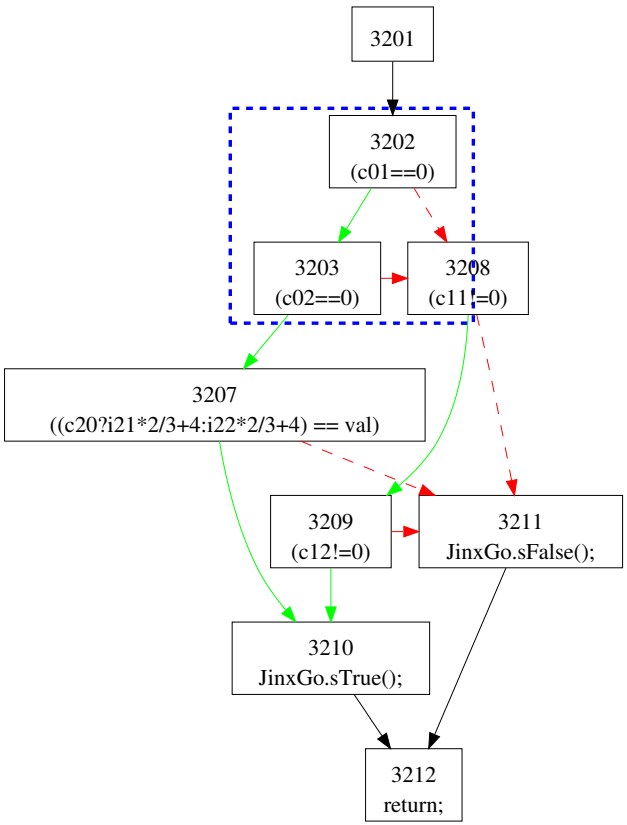
(b) The CFG after replacing the ternary expression by a single node.

An example with ternary expressions

The input program: if $((c01||c02)?(c11\&\&c12) : (c20?i21*2/3+4 : i22*2/3+4) == val)$ sTrue(); else sFalse();



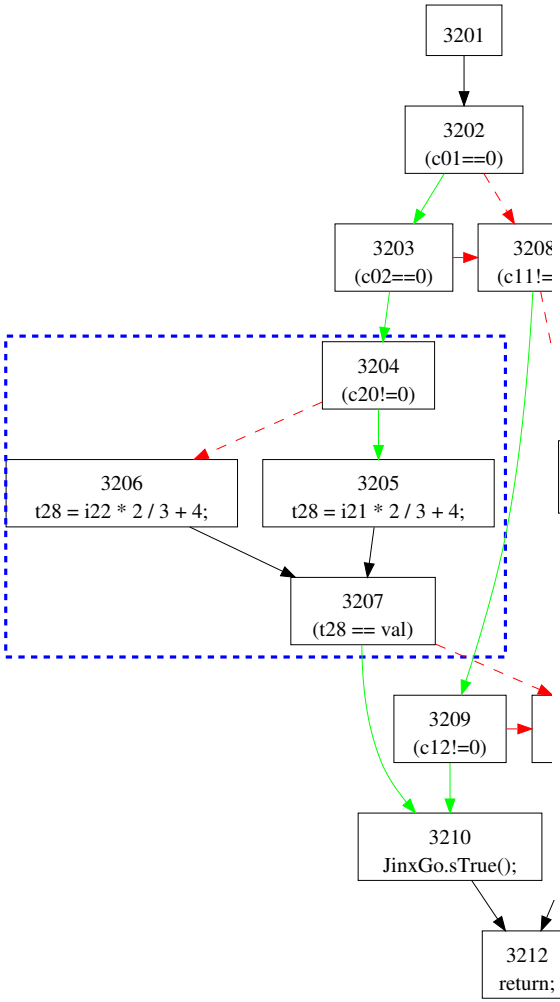
(a) The CFG generated by WALA for the given program.



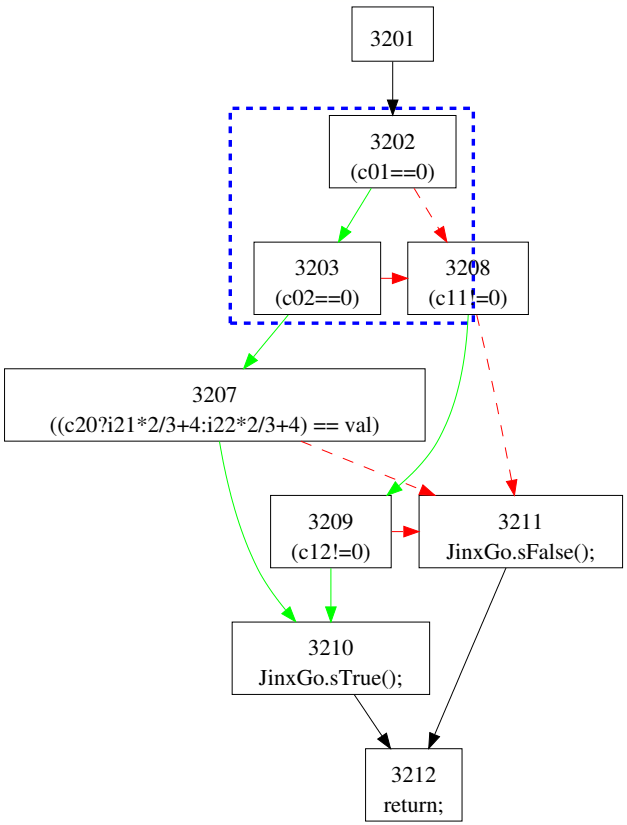
(b) The CFG after replacing the ternary expression by a single node.

An example with ternary expressions

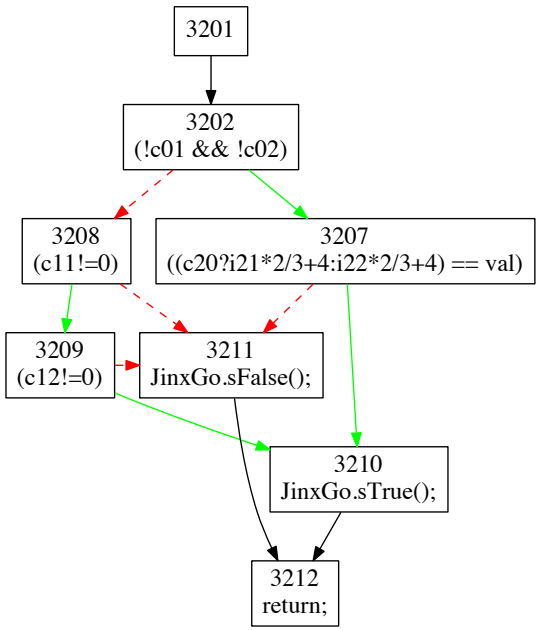
The input program: if $((c01||c02)?(c11\&\&c12) : (c20?i21*2/3+4 : i22*2/3+4) == val)$ sTrue(); else sFalse();



(a) The CFG generated by WALA for the given program.



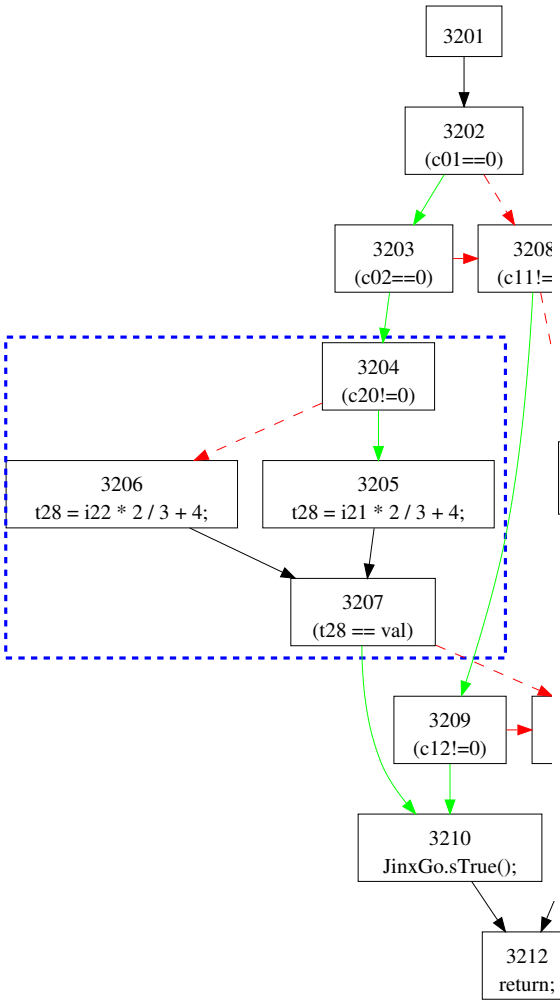
(b) The CFG after replacing the ternary expression by a single node.



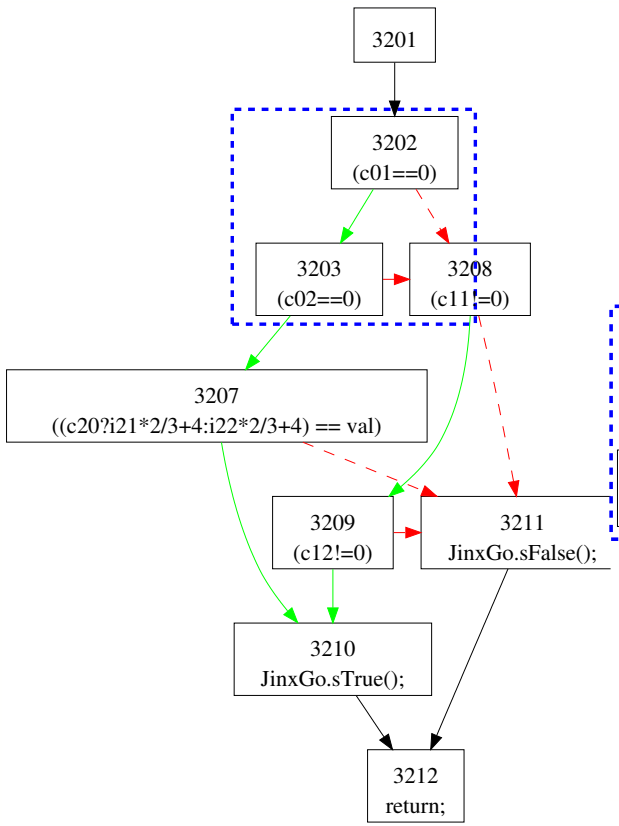
(c) The CFG after replacing the first boolean expression also by a single node.

An example with ternary expressions

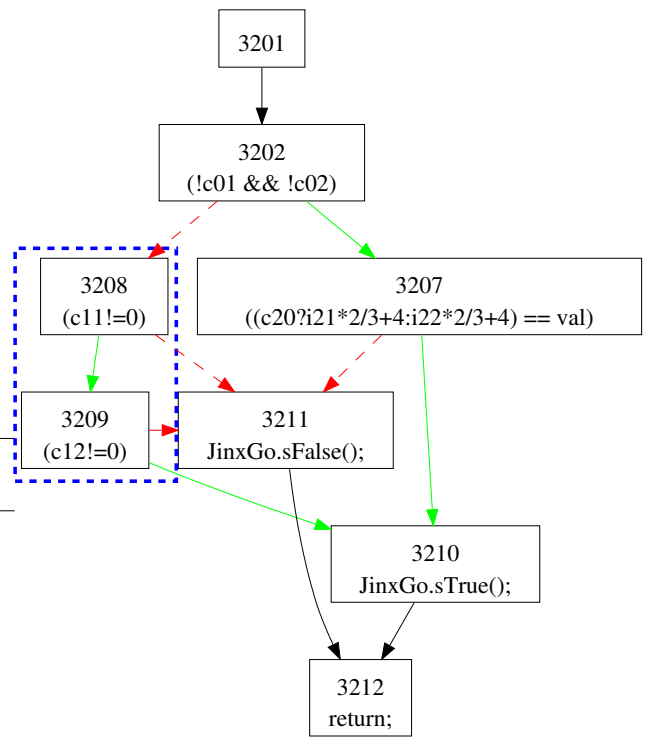
The input program: if $((c01||c02)?(c11\&\&c12) : (c20?i21*2/3+4 : i22*2/3+4) == val)$ sTrue(); else sFalse();



(a) The CFG generated by WALA for the given program.



(b) The CFG after replacing the ternary expression by a single node.

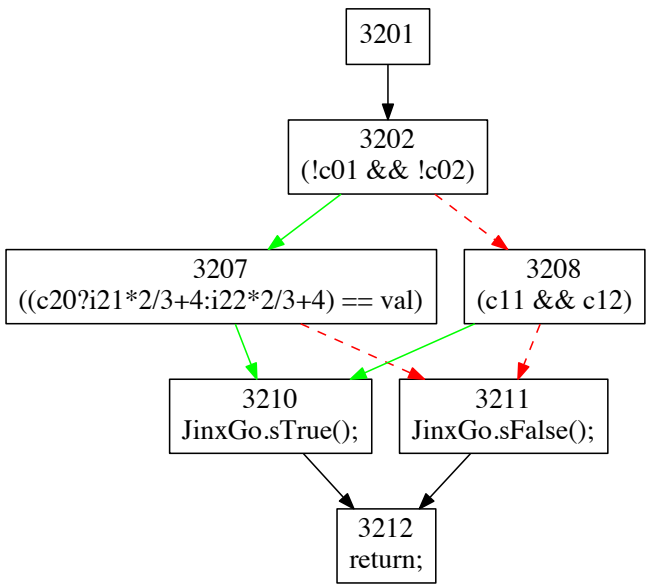


(c) The CFG after replacing the first boolean expression also by a single node.

An example with ternary expressions – continued

The input program

if $((c01||c02)?(c11\&\&c12) : (c20?i21 * 2/3 + 4 : i22 * 2/3 + 4) == val)$ sTrue(); else sFalse();

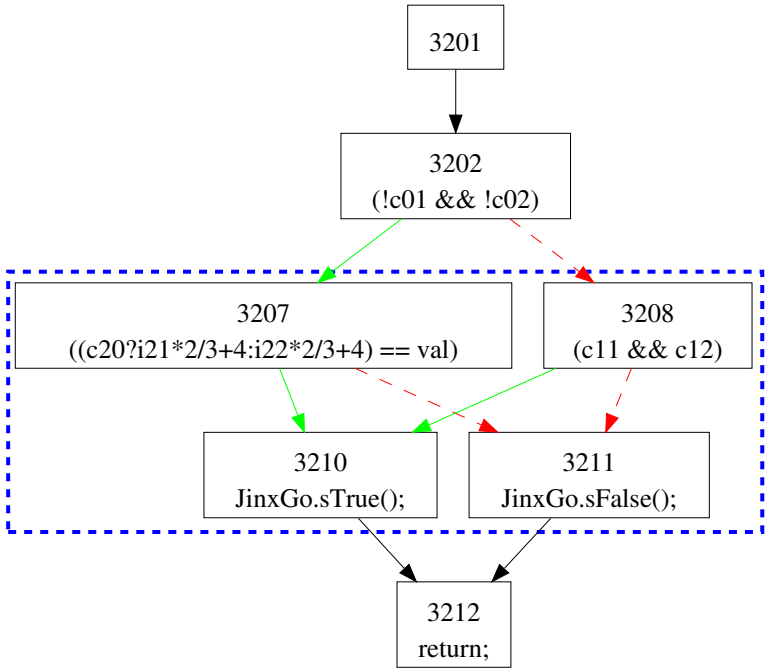


(d) The CFG after replacing all the top level

An example with ternary expressions – continued

The input program

if $((c01||c02)?(c11\&\&c12) : (c20?i21 * 2/3 + 4 : i22 * 2/3 + 4) == val)$ sTrue(); else sFalse();



(d) The CFG after replacing all the top level boolean expressions. The second layer ternary

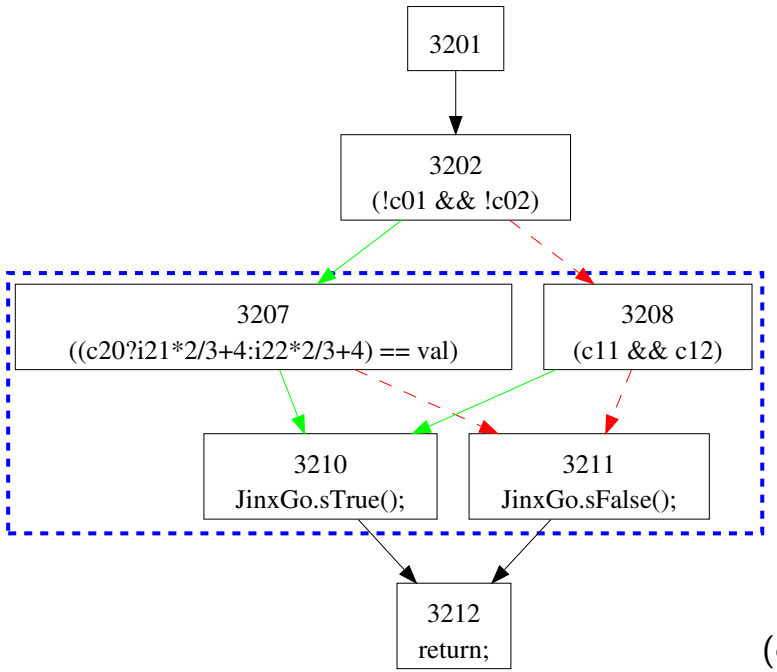
An example with ternary expressions – continued

The input program

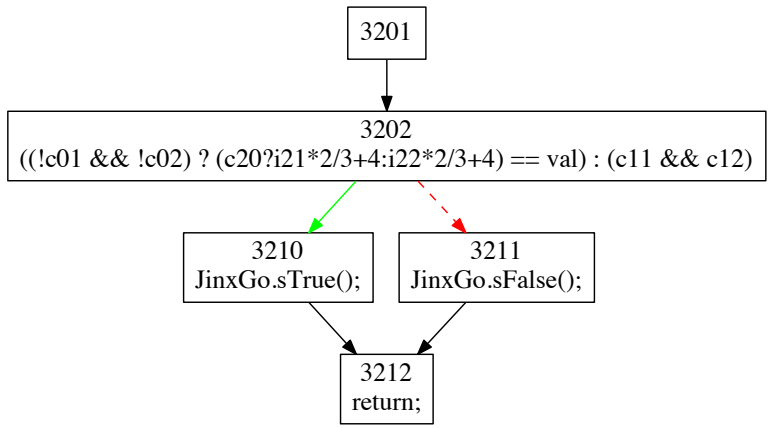
if $((c01||c02)?(c11\&\&c12) : (c20?i21 * 2/3 + 4 : i22 * 2/3 + 4) == val)$ sTrue(); else sFalse();

The output program

if $((!c01\&\&!c02)?(c20?i21 * 2/3 + 4 : i22 * 2/3 + 4) == val : (c11\&\&c12))$ sTrue(); else sFalse();



(d) The CFG after replacing all the top level boolean expressions. The second layer ternary

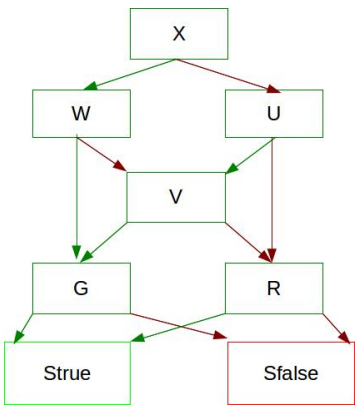


(e) The original program regenerated, albeit with some structural (but equivalent) changes.

The Untwistable DAG: An example

The original input program

if $((X?(W||V) : (U\&\&V))?G : R)$ sTrue(); else sFalse();

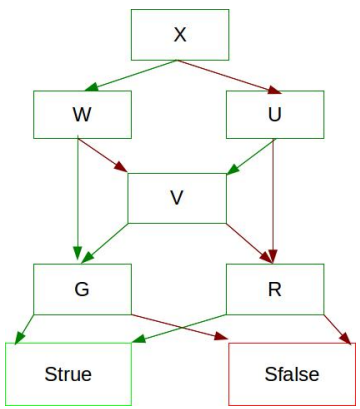


(a) A CFG with an untwistable DAG.

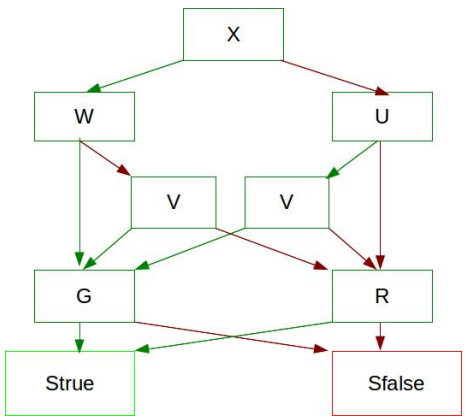
The Untwistable DAG: An example

The original input program

if $((X?(W||V) : (U\&\&V))?G : R)$ sTrue(); else sFalse();



(a) A CFG with an untwistable DAG.

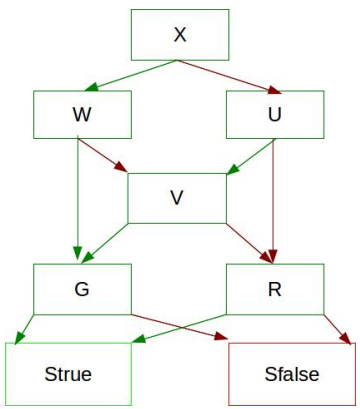


(b) Equivalent CFG after duplicating a node.

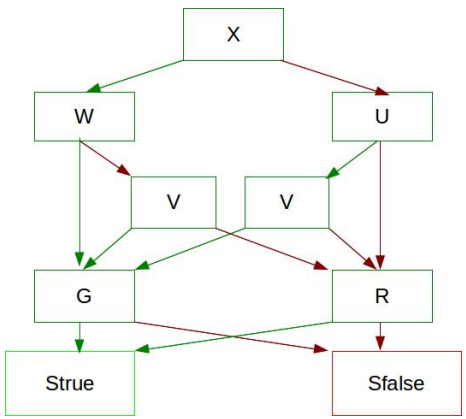
The Untwistable DAG: An example

The original input program

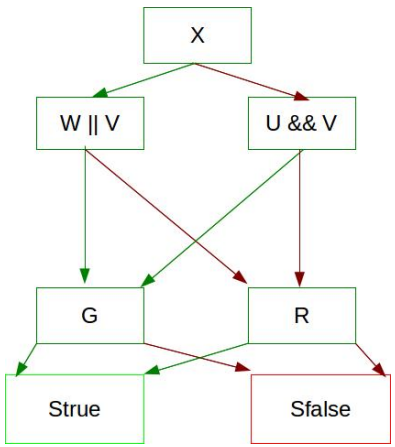
if $((X?(W||V) : (U\&\&V))?G : R)$ sTrue(); else sFalse();



(a) A CFG with an untwistable DAG.



(b) Equivalent CFG after duplicating a node.

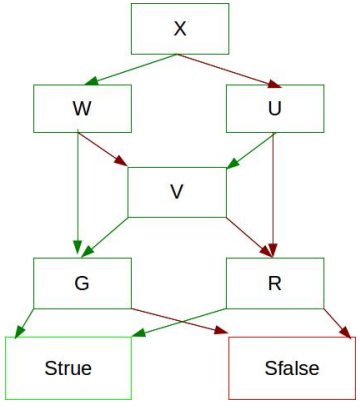


(c) The simplified CFG.

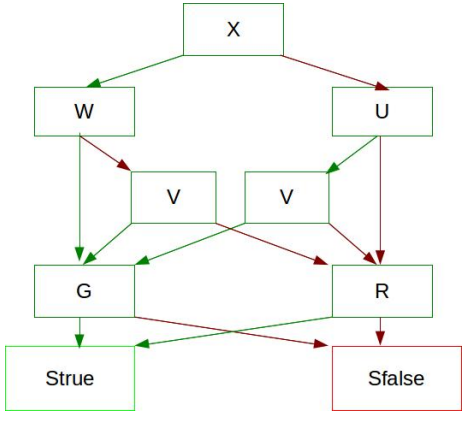
The Untwistable DAG: An example

The original input program

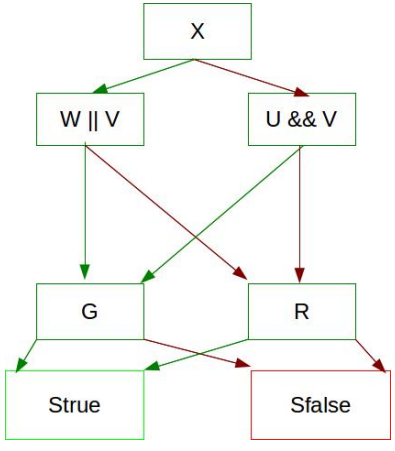
if $((X?(W||V) : (U\&\&V))?G : R)$ sTrue(); else sFalse();



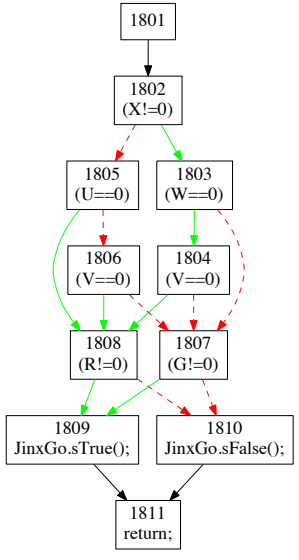
(a) A CFG with an untwistable DAG.



(b) Equivalent CFG after duplicating a node.



(c) The simplified CFG.

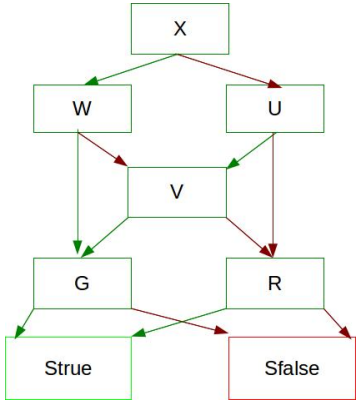


(d) Generated graph equivalent to the graph in (b)

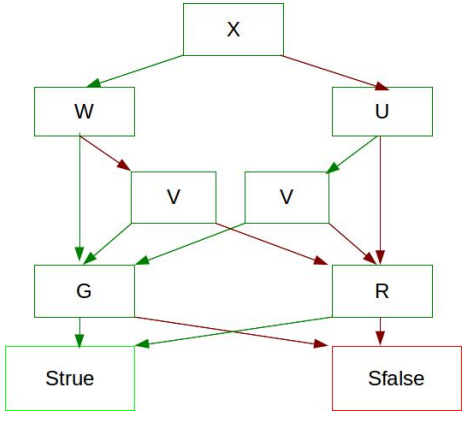
The Untwistable DAG: An example

The original input program

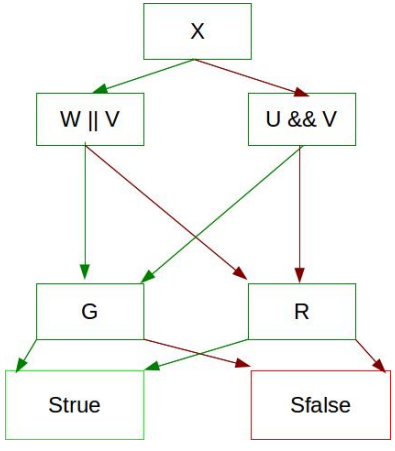
if $((X?(W||V) : (U\&\&V))?G : R)$ sTrue(); else sFalse();



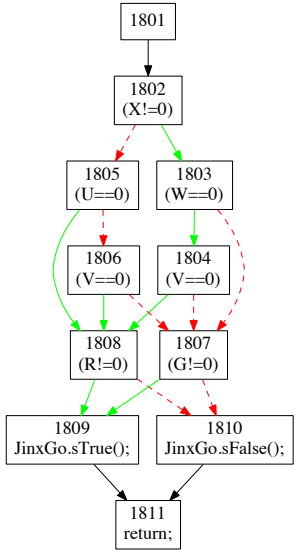
(a) A CFG with an untwistable DAG.



(b) Equivalent CFG after duplicating a node.



(c) The simplified CFG.



(d) Generated graph equivalent to the graph in (b)

Results: Efficiency

Subject programs used in the empirical studies.

Subject	Classes	Methods	Bytecode instructions
ANTLR	507	2582	103797
XERCES	51	341	14585
DAO	3	30	930
APP A	29	195	2588
APP B	243	2134	36997
APP C	94	749	39769

Time for analysis in seconds.

Subject	Parsing Building	Pointer Analysis	Decompiling expressions	Total time
ANTLR	25	24	1	51
XERCES	8	5	0	14
DAO	2	0	0	2
APP A	7	0	0	8
APP B	11	2	0	15
APP C	6	4	1	11

Results: efficacy

Number of `AND` and `OR` clauses.

Subject	exactly 2	exactly 3	>3	Total expressions	max size
ANTLR	255	64	48	367	19
XERCES	98	24	25	147	10
DAO	10	0	6	16	7
APP A	12	0	1	13	7
APP B	233	50	33	316	13
APP C	86	16	8	110	8

Number of Ternary clauses.

Subject	exactly 1	exactly 2	>2	Total expressions	max size
ANTLR	14	-	-	14	1
XERCES	-	-	-	-	1
DAO	23	9	19	51	19
APP A	2	-	-	2	1
APP B	10	-	-	10	1
APP C	2	-	-	2	1

Results: Comparison with other tools

Number of mixed `AND` and `OR` and ternary clauses detected
"C" correctly,
"M" in a "Messed"up way
"I" Incorrectly

by each tool.

Tool Name	exactly 2		≥3		Total	
	C	M/I	C	M/I	C	M/I
JinxGo	2	0/0	5	0/0	7	0/0
Jode	0	2/0	0	5/0	0	7/0
JReversePro	0	2/0	0	5/0	0	7/0
Procyon	2	0/0	0	5/0	2	5/0
FernFlower	2	0/0	0	5/0	2	5/0
CFR	2	0/0	2	0/3	4	0/3
Soot	0	2/0	0	5/0	0	7/0