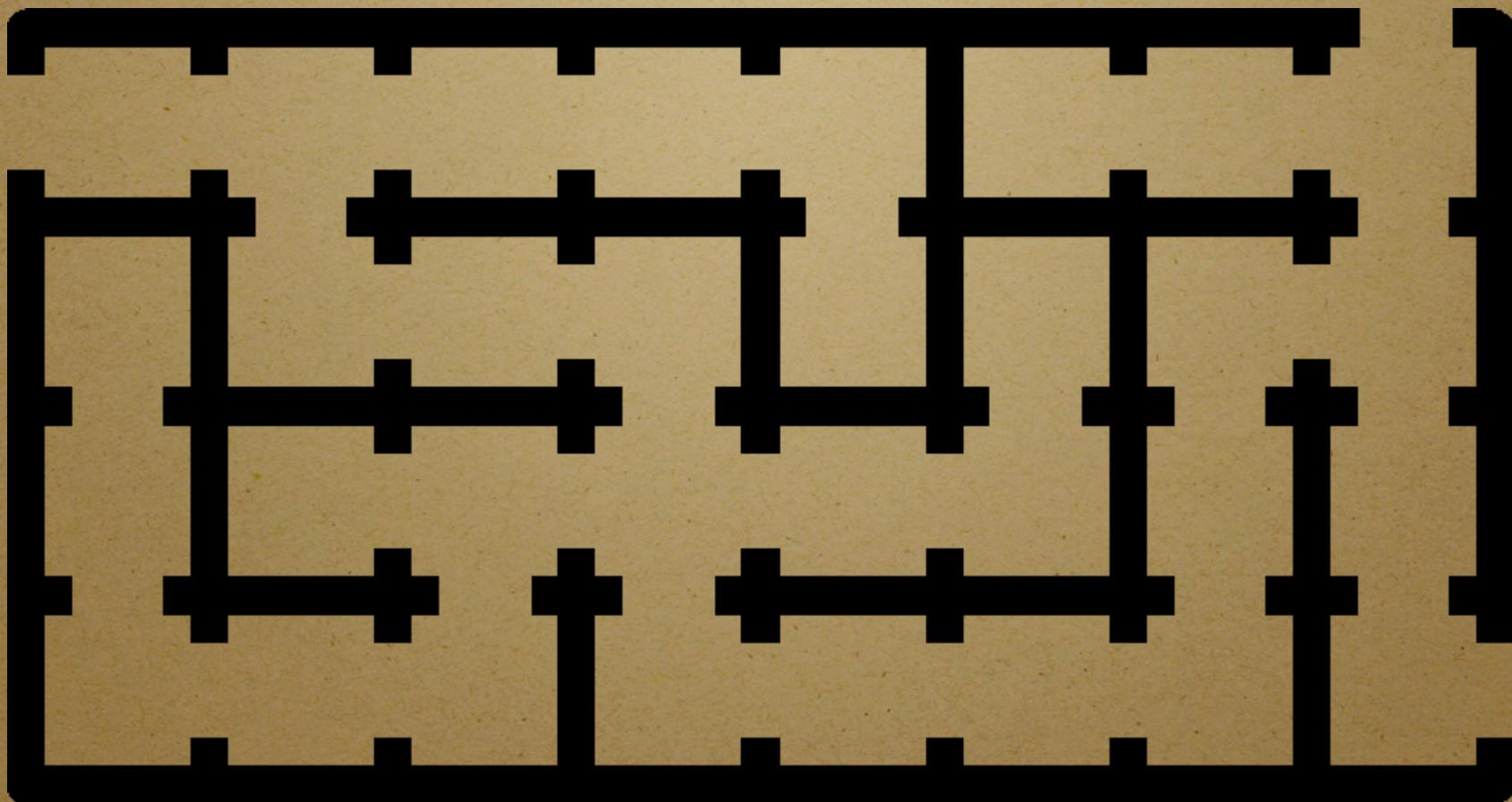
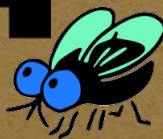
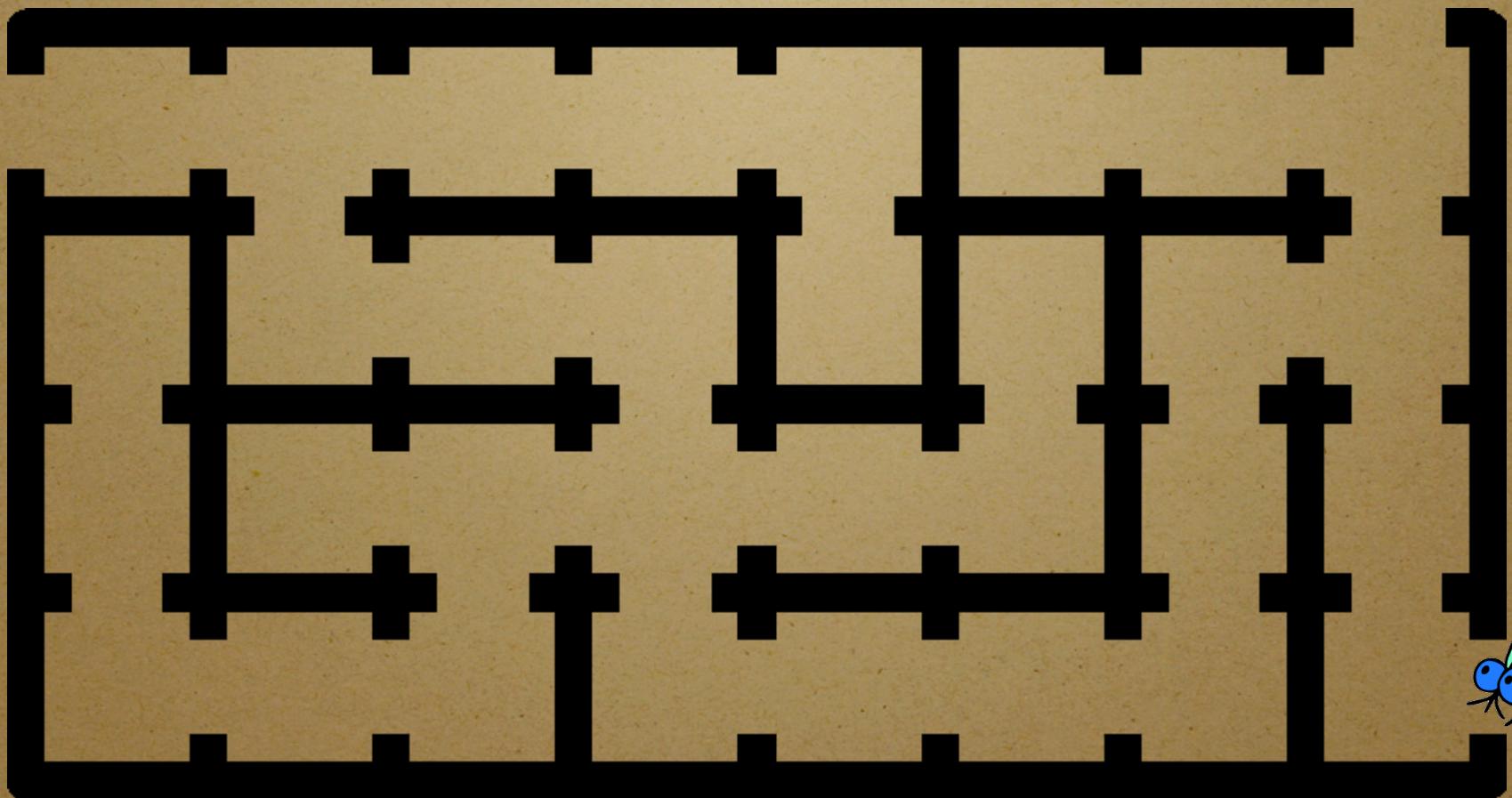


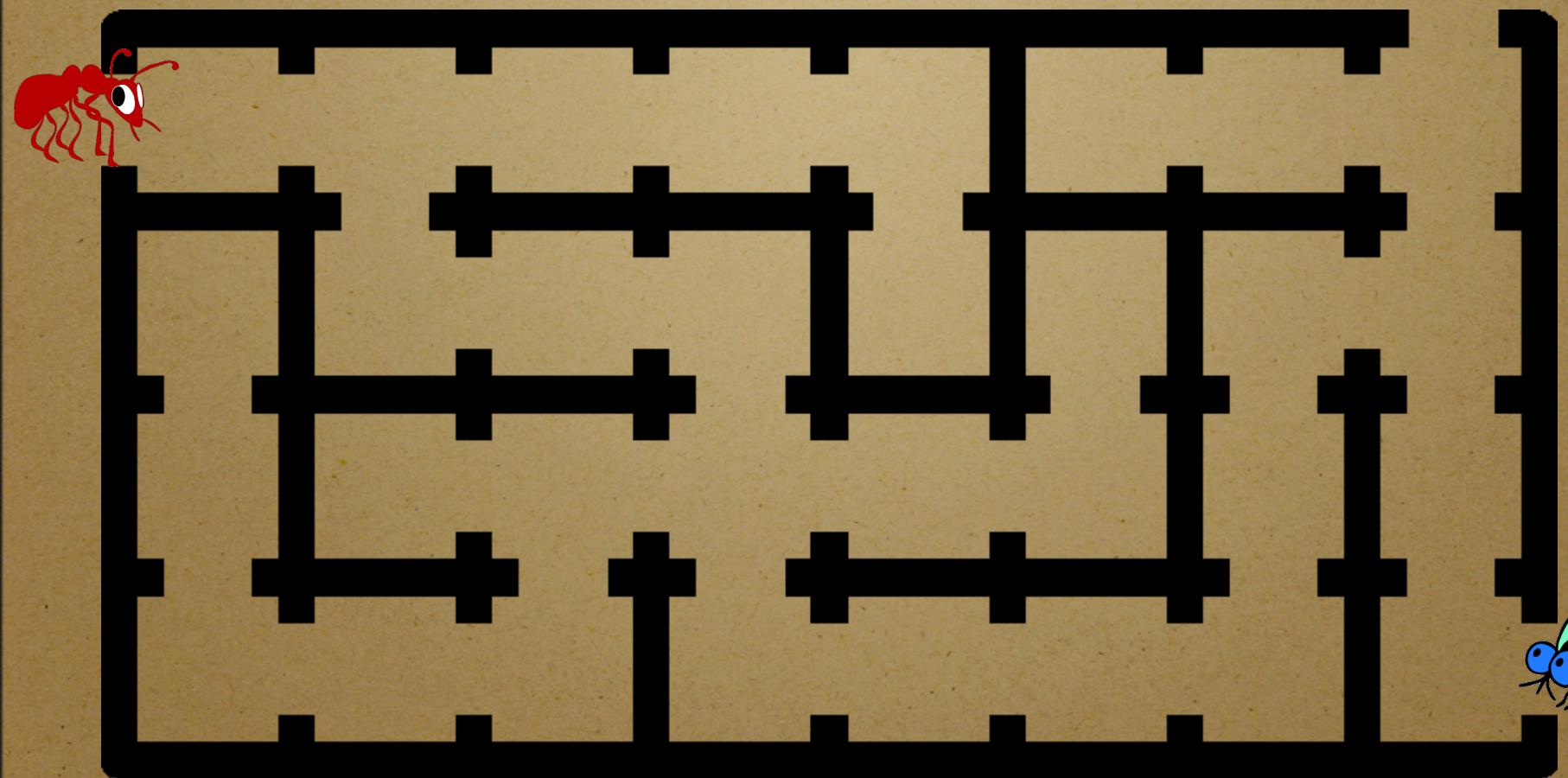
# Scavenger Hunt



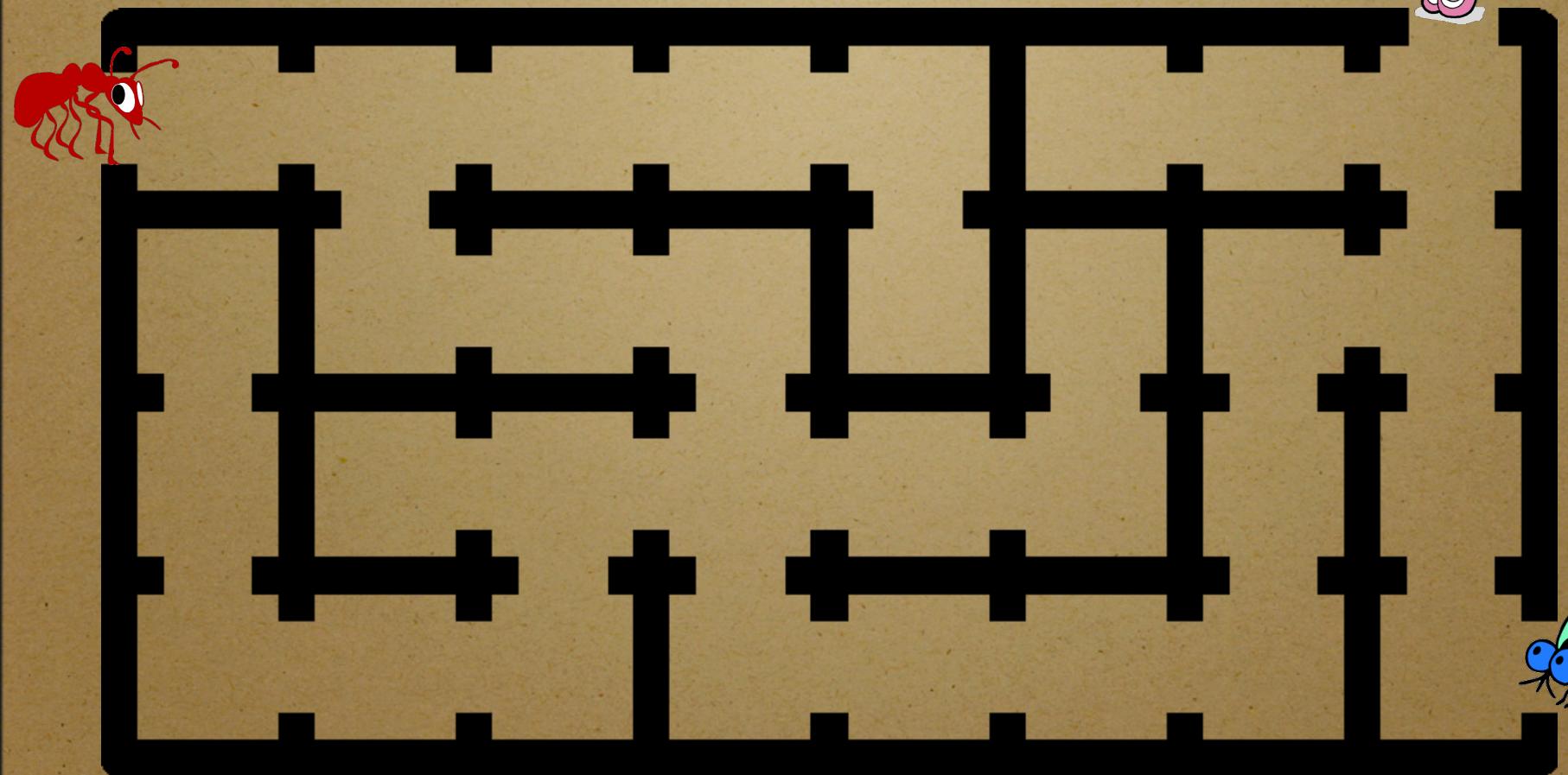
# Scavenger Hunt



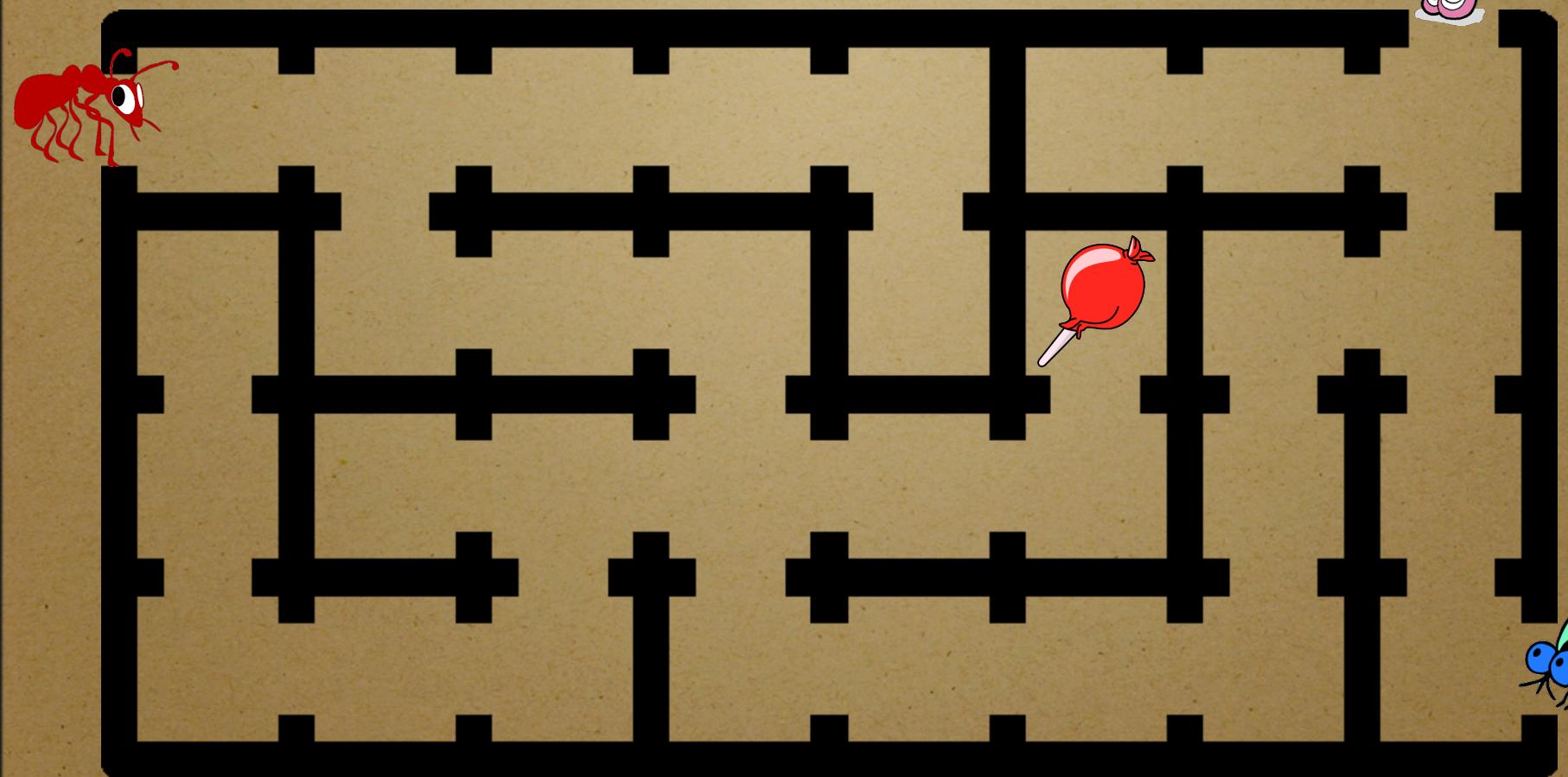
# Scavenger Hunt



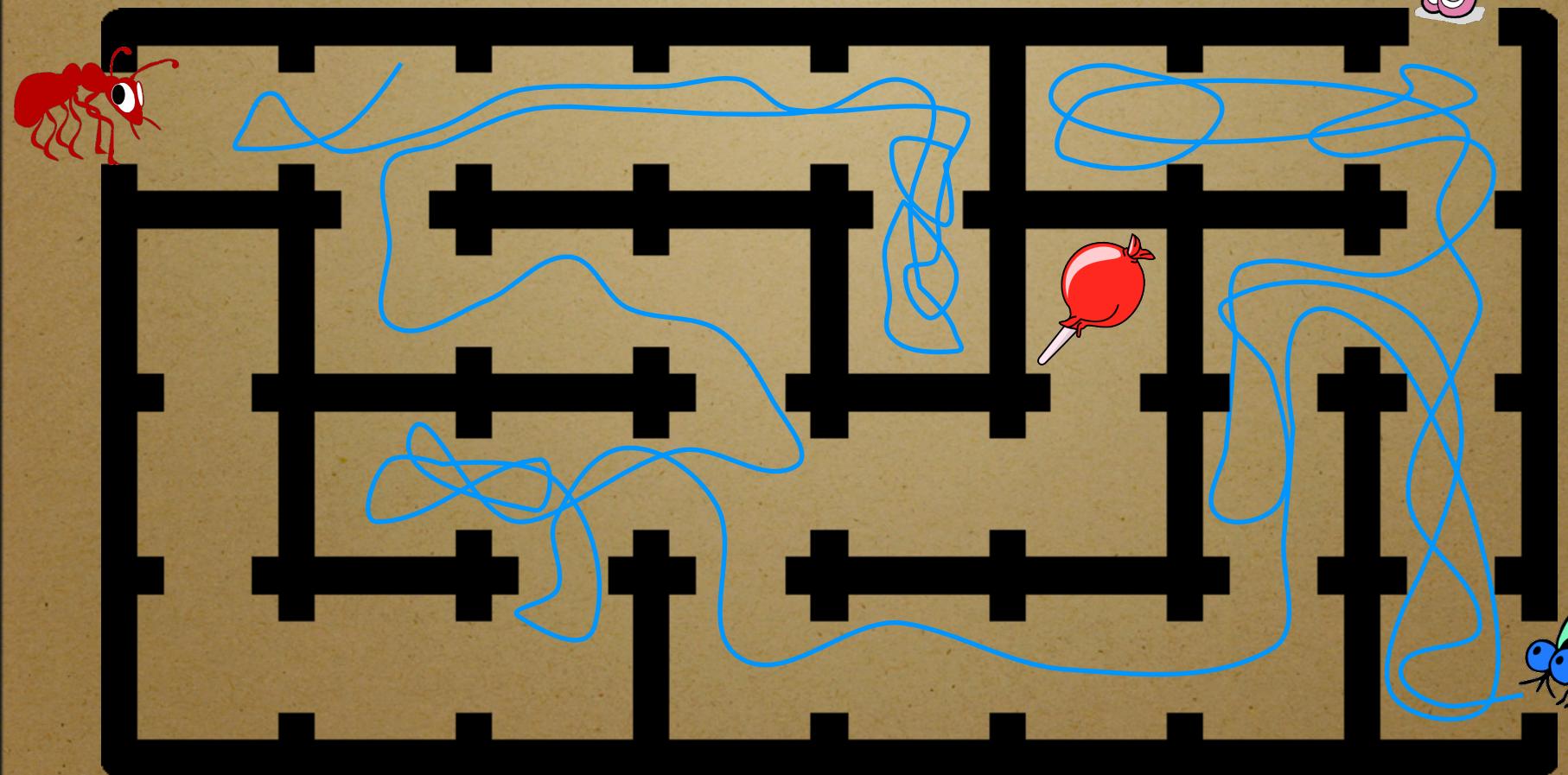
# Scavenger Hunt



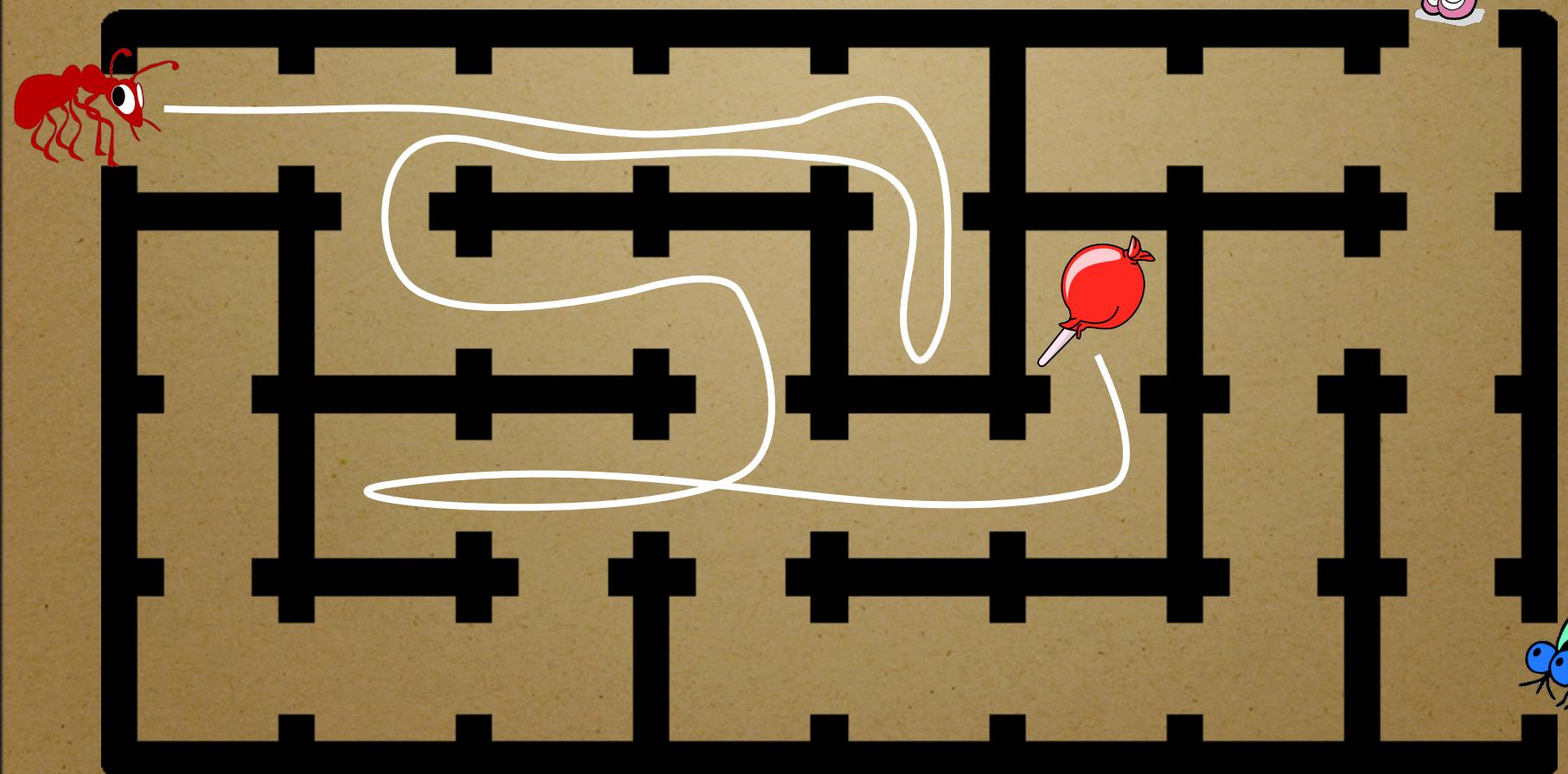
# Scavenger Hunt



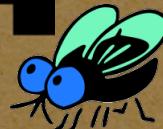
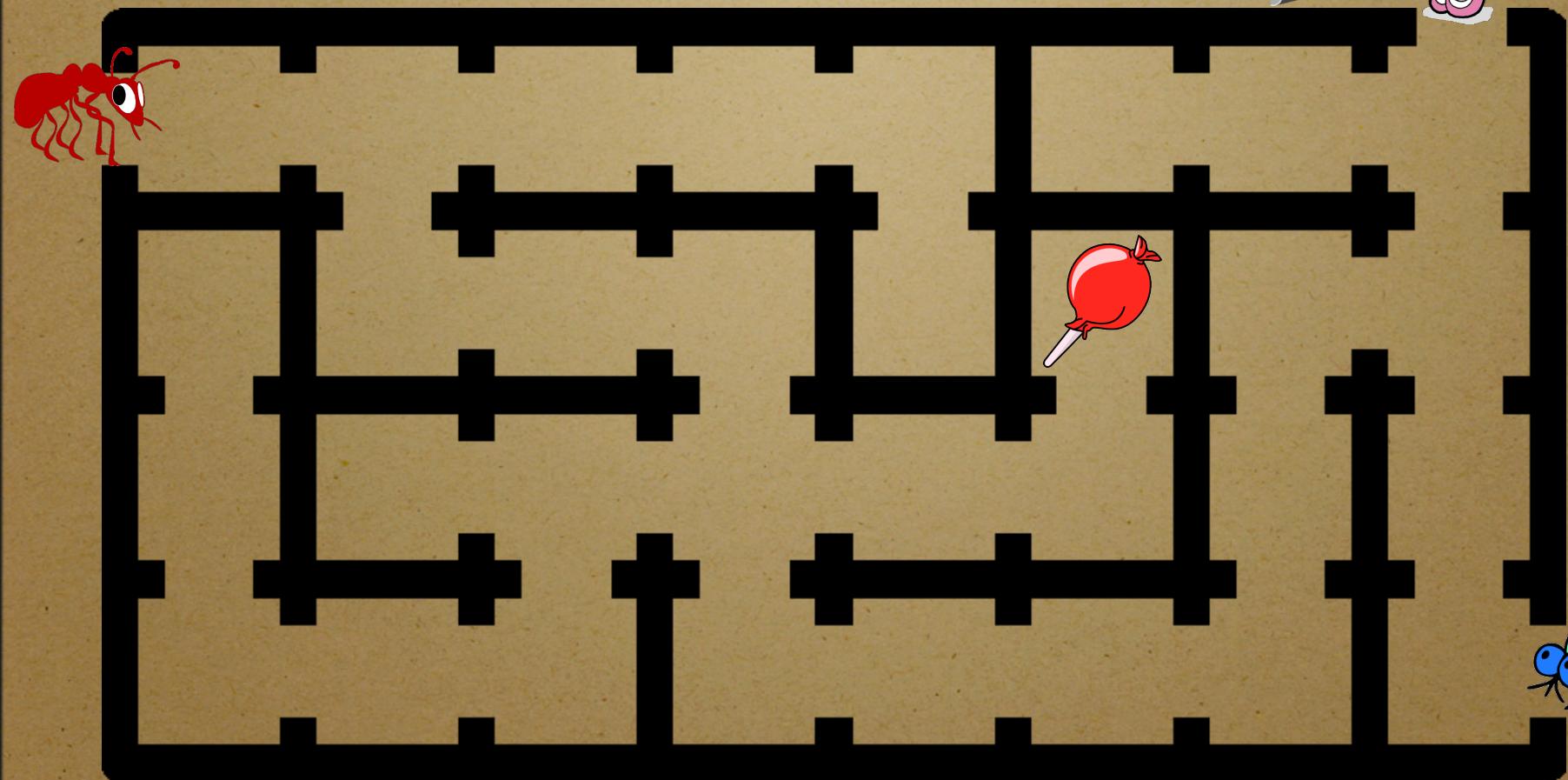
# Scavenger Hunt



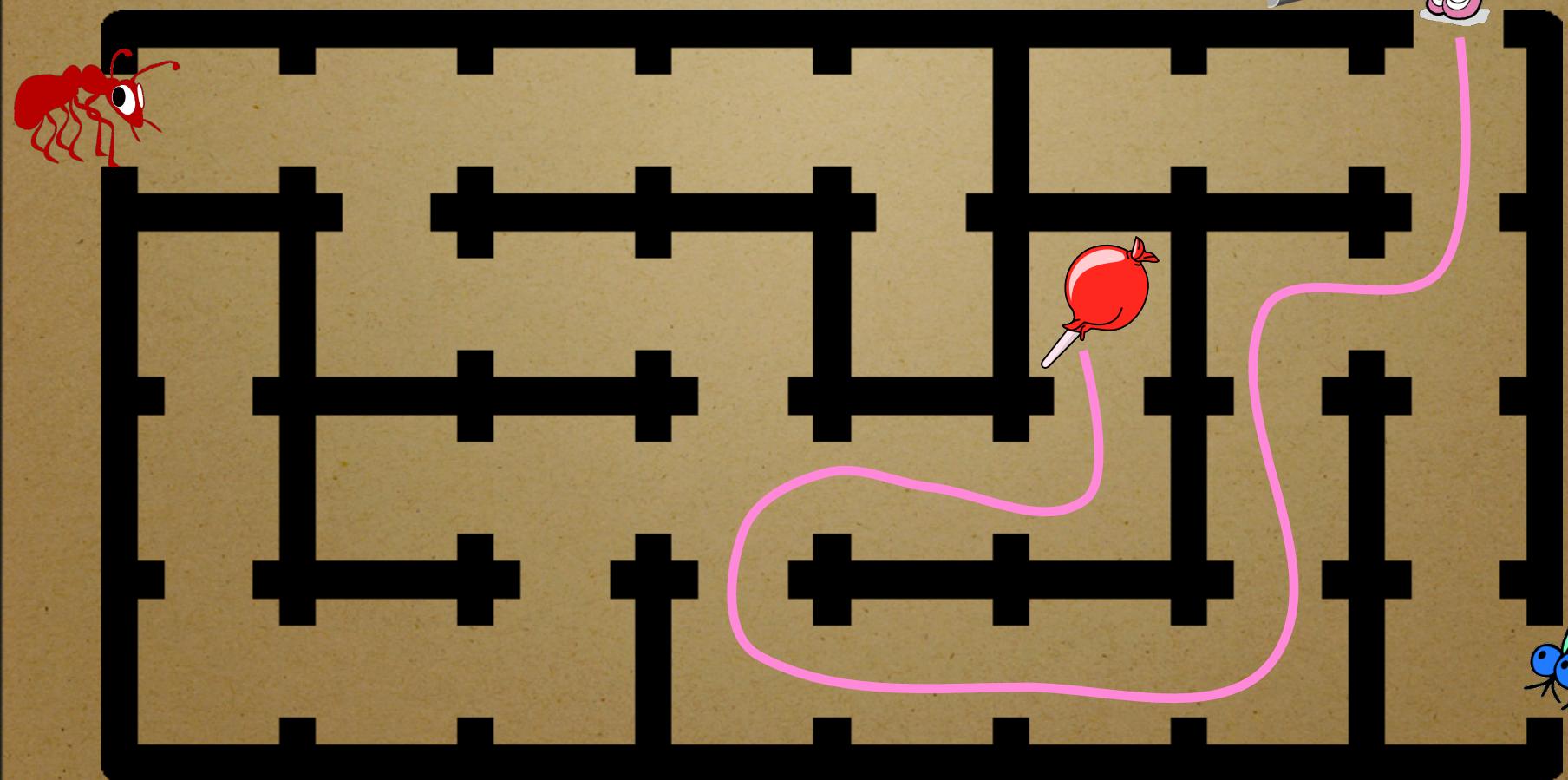
# Scavenger Hunt



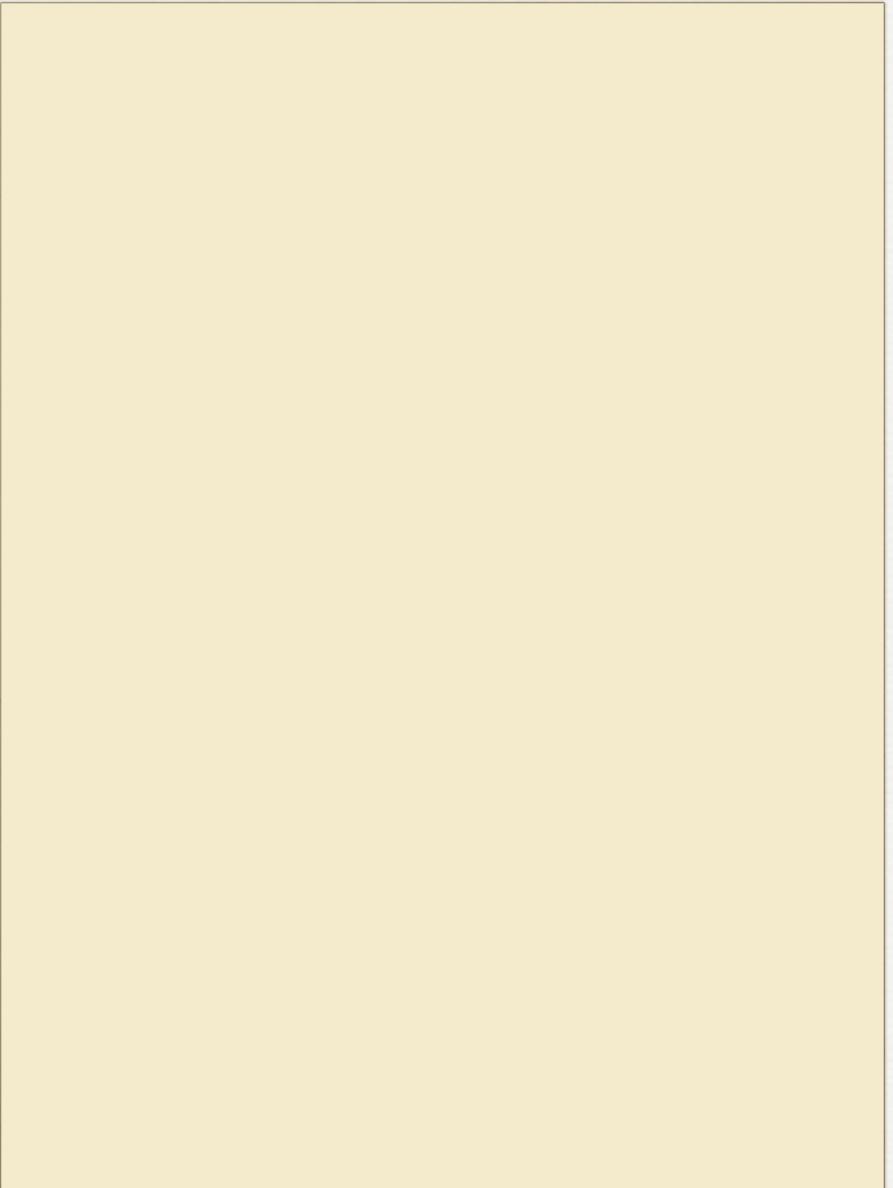
# Scavenger Hunt



# Scavenger Hunt



# Software Testing



# Software Testing

```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

Random Testing?

```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

Random Testing?



```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

Random Testing?



Directed Testing?

```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

Random Testing?



Directed Testing?



```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

Random Testing?



Directed Testing?



Manual Testing?

```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

Random Testing?



Directed Testing?



Manual Testing?



```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182             else if (timImp>0) {  
183                 if (directionUp) {  
184                     ++cnt;  
185                 } else {  
186                     --cnt;  
187                 }  
188             }  
189         }  
190     }  
191     if (ctrl_io_in[SENS_BOTTOM]) {  
192         cnt = 0;  
193         cntValid = TRUE;  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }
```

# Software Testing

Random Testing?



Directed Testing?



Manual Testing?



```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182         }  
183     }  
184     else {  
185         if (ctrl_io_in[SENS_IMPULS]) {  
186             if (ctrl_io_out[MOTOR_ON]) {  
187                 if (directionUp) {  
188                     ++cnt;  
189                 } else {  
190                     --cnt;  
191                 }  
192             }  
193             if (ctrl_io_out[MOTOR_ON]) {  
194                 if (ctrl_io_in[SENS_IMPULS]);  
195                 if (timImp>0) {  
196                     --timImp;  
197                     if (timImp==0) {  
198                         if (cmd!=CMD_NONE) {  
199                             cmd = CMD_NONE;  
200                         }  
201                     }  
202                 }  
203             }  
204         }  
205     }  
206 }
```



# Software Testing

Random Testing?



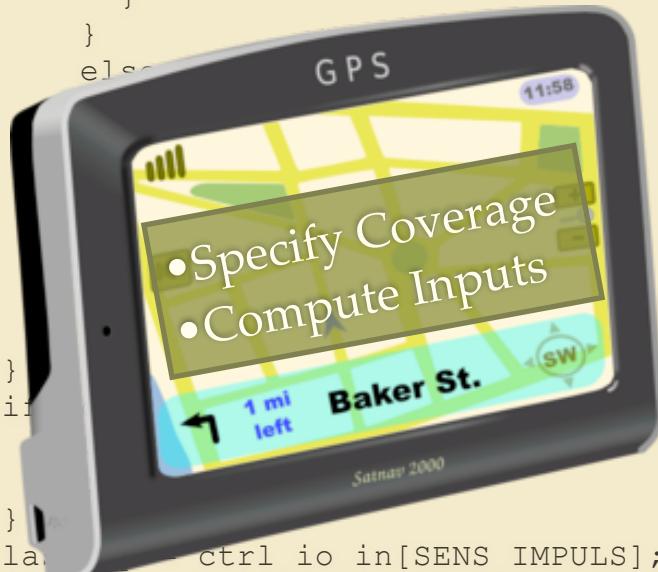
Directed Testing?



Manual Testing?



```
173     if (ctrl_io_in[SENS_IMPULS]) {  
174         if (!lastImp){  
175             if (ctrl_io_out[MOTOR_ON]) {  
176                 if (directionUp) {  
177                     ++cnt;  
178                 } else {  
179                     --cnt;  
180                 }  
181             }  
182         }  
183     }  
184     else {  
185         if (ctrl_io_in[SENS_IMPULS]) {  
186             if (ctrl_io_out[MOTOR_ON]) {  
187                 if (directionUp) {  
188                     ++cnt;  
189                 } else {  
190                     --cnt;  
191                 }  
192             }  
193         }  
194     }  
195     lastImp = ctrl_io_in[SENS_IMPULS];  
196     if (timImp>0) {  
197         --timImp;  
198         if (timImp==0) {  
199             if (cmd!=CMD_NONE) {  
200                 cmd = CMD_NONE;  
201             }  
202         }  
203     }  
204 }
```



# How did you specify your test suite ?

Michael Tautschnig  
Formal Methods in Systems Engineering  
Vienna University of Technology

*Joint work with  
Andreas Holzer, Christian Schallhart, and Helmut Veith*

# Standard Coverage Criteria

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Basic Block Coverage

Requires a test suite such that each basic block is executed at least once

# Standard Coverage Criteria

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Basic Block Coverage

Requires a test suite such that each basic block is executed at least once

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Decision Coverage

Conditional statements evaluate to true / false

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Decision Coverage

Conditional statements evaluate to true / false

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Condition Coverage

Atomic conditions evaluate to true/false

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a) | (b) && (c))
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Condition Coverage

Atomic conditions evaluate to true/false

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Condition/Decision Coverage

Union of condition and decision coverage

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a) | (b) && (c))
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Condition/Decision Coverage

Union of condition and decision coverage

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Multiple Condition Coverage

All Boolean combinations of atomic conditions in complex conditions

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a) | (b) && (c))
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

a	b	c
F	F	F
F	F	T
F	T	F
F	T	T
T	F	F
T	F	T
T	T	F
T	T	T

## Multiple Condition Coverage

All Boolean combinations of atomic conditions in complex conditions

# Standard Coverage Criteria

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Def-Use Coverage

Cover all definition/use pairs of a variable *d*

# Standard Coverage Criteria

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Def-Use Coverage

Cover all definition/use pairs of a variable *d*

# Beyond Standard Criteria

```
1 void insert(int * a, int pos) {  
2     ...  
3 }  
4  
5 int main(int argc, char * argv[ ]) {  
6     int i;  
7     int A[100];  
8  
9     for (i=0; i<100; ++i)  
10        insert(A, i);  
11  
12     return 0;  
13 }
```

## Loop-Bounded Path Coverage

- All paths through *main* and *insert*
- Pass each statement two times

# Beyond Standard Criteria

```
1 void insert(int * a, int pos) {  
2     ...  
3 }  
4  
5 int main(int argc, char * argv[ ]) {  
6     int i;  
7     int A[100];  
8  
9     for (i=0; i<100; ++i)  
10        insert(A, i);  
11  
12    return 0;  
13 }
```



## Loop-Bounded Path Coverage

- All paths through *main* and *insert*
- Pass each statement two times

# Beyond Standard Criteria

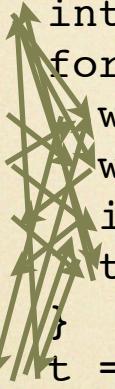
```
1 int partition(int a[], int left, int right) {  
2     int v = a[right], i = left - 1, j = right, t;  
3     for (;;) {  
4         while (a[++i] < v) ;  
5         while (j > left && a[--j] > v) ;  
6         if (i >= j) break;  
7         t = a[i]; a[i] = a[j]; a[j] = t;  
8     }  
9     t = a[i]; a[i] = a[right]; a[right] = t;  
10    return i;  
11 }
```

## Cartesian Block Coverage

All pairs and triples of basic blocks in function *partition*

# Beyond Standard Criteria

```
1 int partition(int a[], int left, int right) {  
2     int v = a[right], i = left - 1, j = right, t;  
3     for (;;) {  
4         while (a[++i] < v) ;  
5         while (j > left && a[--j] > v) ;  
6         if (i >= j) break;  
7         t = a[i]; a[i] = a[j]; a[j] = t;  
8     }  
9     t = a[i]; a[i] = a[right]; a[right] = t;  
10    return i;  
11 }
```



## Cartesian Block Coverage

All pairs and triples of basic blocks in function *partition*

# Specs for Working Programmers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Cover Specific Lines of Code

Lines 4 and 6

# Specs for Working Programmers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

Cover Specific Lines of Code

Lines 4 and 6

# Specs for Working Programmers

```
1 void eval(int * a, int first, int last) {
2     if (first > last) return;
3     printf("a[%d]=%d\n", first, a[first]);
4     eval(a+1, first+1, last);
5     return;
6 }
7
8 int main(int argc, char * argv[ ]) {
9     int i;
10    int A[100];
11
12    for (i=0; i<100; ++i)
13        insert(A, i);
14
15    eval(A, 0, 100);
16    return 0;
17 }
```

## Restricted Scope of Analysis

Basic block coverage in function *eval* only

# Specs for Working Programmers

```
1 void eval(int * a, int first, int last) {
2     if (first > last) return;
3     printf("a[%d]=%d\n", first, a[first]);
4     eval(a+1, first+1, last);
5     return;
6 }
7
8 int main(int argc, char * argv[ ]) {
9     int i;
10    int A[100];
11
12    for (i=0; i<100; ++i)
13        insert(A, i);
14
15    eval(A, 0, 100);
16    return 0;
17 }
```

## Restricted Scope of Analysis

Basic block coverage in function *eval* only

# Specs for Working Programmers

```
1 void sort(int * a, int len) {
2     int i, t;
3     for (i=1; i<len; ++i) {
4         if (compare(a[i-1], a[i])) continue;
5         t=a[i]; a[i]=a[i-1]; a[i-1]=t;
6     }
7 }
8
9 void eval(int * a, int len) {
10    int i;
11    for (i=0; i < 3; ++i)
12        printf("a[%d]=%d\n", i, a[i]);
13 }
14
15 int main(int argc, char * argv[]) {
16    int i; int A[100];
17    for (i=0; i < 100; ++i) sort(A, 100);
18    eval(A, 100);
19    return 0;
20 }
```

## Interaction Coverage

Cover all pairs of conditions in *sort* and basic blocks in *eval*

# Specs for Working Programmers

```
1 void sort(int * a, int len) {  
2     int i, t;  
3     for (i=1; i<len; ++i) {  
4         if (compare(a[i-1], a[i])) continue;  
5         t=a[i]; a[i]=a[i-1]; a[i-1]=t;  
6     }  
7 }  
8  
9 void eval(int * a, int len) {  
10    int i;  
11    for (i=0; i < 3; ++i)  
12        printf("a[%d]=%d\n", i, a[i]);  
13 }  
14  
15 int main(int argc, char * argv[]) {  
16     int i; int A[100];  
17     for (i=0; i < 100; ++i) sort(A, 100);  
18     eval(A, 100);  
19     return 0;  
20 }
```

## Interaction Coverage

Cover all pairs of conditions in *sort* and basic blocks in *eval*

# Specs for Working Programmers

```
1 void sort(int * a, int len) {  
2     int i, t;  
3     for (i=1; i<len; ++i) {  
4         if (compare(a[i-1], a[i])) continue;  
5         t=a[i];  
6         a[i]=a[i-1];  
7         a[i-1]=t;  
8     }  
9     return;  
10 }
```

## Constrained Inputs

- Basic block coverage in *sort*
- Each test case shall use list of 2 to 15 elements

# Specs for Working Programmers

```
1 void sort(int * a, int len) {  
2     int i, t;  
3     for (i=1; i<len; ++i) {  
4         if (compare(a[i-1], a[i])) continue;  
5         t=a[i];  
6         a[i]=a[i-1];  
7         a[i-1]=t;  
8     }  
9     return;  
10 }
```

## Constrained Inputs

- Basic block coverage in *sort*
- Each test case shall use list of 2 to 15 elements

# Specs for Working Programmers

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

## Avoid Unfinished Code

- Basic block coverage in *example*
- Never call *unimplemented*

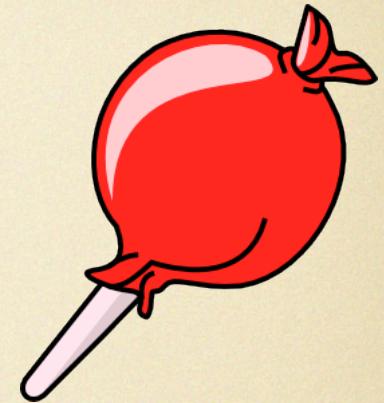
# Specs for Working Programmers

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return a*2;
8 }
```

## Avoid Unfinished Code

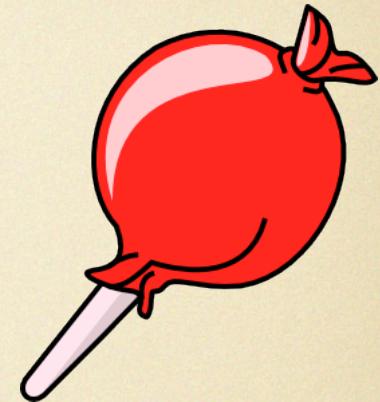
- Basic block coverage in *example*
- Never call *unimplemented*

# Test Specification Language



# Test Specification Language

Precise and simple formalism to describe test suites



Precise semantics

High expressive power

Simple specifications easily expressible

Suitable for working programmer

# Algorithmic Solution

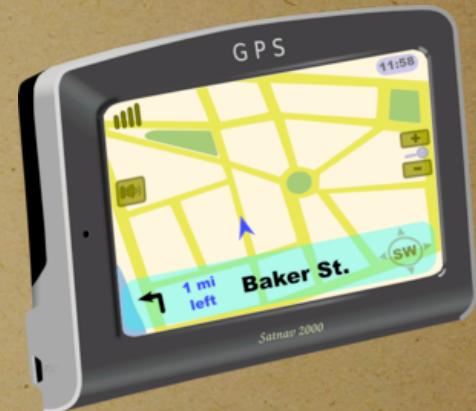


# Algorithmic Solution

Tool for the working programmer

Applicability to real world C code (embedded systems)

Efficient test input generation engines



```
333     if (ctrl.io_in[SENS_IMPULS]) {
334         if (!lastImp)
335             if (cmd == CMD_NONE) {
336                 if (directionOp) {
337                     if (directionOp)
338                         if (cnt > 0)
339                             --cnt;
340                     else
341                         ++cnt;
342                 }
343             else if (rightOp) {
344                 if (directionOp) {
345                     if (cnt > 0)
346                         --cnt;
347                     else
348                         ++cnt;
349                 }
350             }
351         if (ctrl.io_in[SENS_BOTTOM]) {
352             if (ctrl.io_in[CMD])
353                 cmd = 0;
354             ctrl.cmd = TRUE;
355             lastImp = ctrl.io_in[SENS_IMPULS];
356             if (ctrl.io_in[CMD])
357                 if (ctrl.io_in[CMD])
358                     if (ctrl.io_in[CMD])
359                         if (ctrl.io_in[CMD] == 0)
360                             if (cmd == CMD_NONE) {
361                                 cmd = CMD_NONE;
362                             }
363                         }
364                     }
365                 }
366             }
367         }
368     }
```

Lines 179, 193, 200

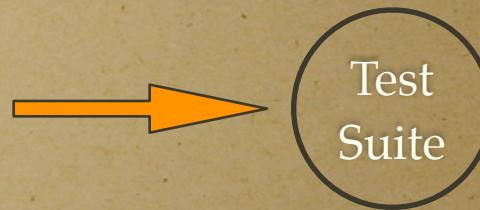
```
313     if (ctrl.io_in[SENS_IMPULS]) {
314         if (!lastImp)
315             if (cmd == CMD_NONE) {
316                 if (directionOpp) {
317                     if (directionOpp)
318                         if (cnt >= 0)
319                             if (lastTempD) {
320                                 if (directionOpp) {
321                                     if (lastTempD)
322                                         if (directionOpp) {
323                                             if (lastTempD)
324                                                 if (directionOpp) {
325                                                     if (lastTempD)
326                                                         if (directionOpp) {
327                                                             if (lastTempD)
328                                                                 if (directionOpp)
329                                                                     if (lastTempD)
330                                                                         if (directionOpp)
331                                                                             if (lastTempD)
332                                                                                 if (directionOpp)
333                                     if (lastTempD)
334                                         if (directionOpp)
335                                         if (lastTempD)
336                                         if (directionOpp)
337                                         if (lastTempD)
338                                         if (directionOpp)
339                                         if (lastTempD)
340                                         if (directionOpp)
341                                         if (lastTempD)
342                                         if (directionOpp)
343                                         if (lastTempD)
344                                         if (directionOpp)
345                                         if (lastTempD)
346                                         if (directionOpp)
347                                         if (lastTempD)
348                                         if (directionOpp)
349                                         if (lastTempD)
350                                         if (directionOpp)
351                                         if (lastTempD)
352                                         if (directionOpp)
353                                         if (lastTempD)
354                                         if (directionOpp)
355                                         if (lastTempD)
356                                         if (directionOpp)
357                                         if (lastTempD)
358                                         if (directionOpp)
359                                         if (lastTempD)
360                                         if (directionOpp)
361                                         if (lastTempD)
362                                         if (directionOpp)
363                                         if (lastTempD)
364                                         if (directionOpp)
365                                         if (lastTempD)
366                                         if (directionOpp)
367                                         if (lastTempD)
368                                         if (directionOpp)
369                                         if (lastTempD)
370                                         if (directionOpp)
371                                         if (lastTempD)
372                                         if (directionOpp)
373                                         if (lastTempD)
374                                         if (directionOpp)
375                                         if (lastTempD)
376                                         if (directionOpp)
377                                         if (lastTempD)
378                                         if (directionOpp)
379                                         if (lastTempD)
380                                         if (directionOpp)
381                                         if (lastTempD)
382                                         if (directionOpp)
383                                         if (lastTempD)
384                                         if (directionOpp)
385                                         if (lastTempD)
386                                         if (directionOpp)
387                                         if (lastTempD)
388                                         if (directionOpp)
389                                         if (lastTempD)
390                                         if (directionOpp)
391                                         if (lastTempD)
392                                         if (directionOpp)
393                                         if (lastTempD)
394                                         if (directionOpp)
395                                         if (lastTempD)
396                                         if (directionOpp)
397                                         if (lastTempD)
398                                         if (directionOpp)
399                                         if (lastTempD)
400                                         if (cmd == CMD_NONE) {
401                                         cmd = CMD_NONE;
402                                         }
403                                         }
```



Lines 179, 193, 200

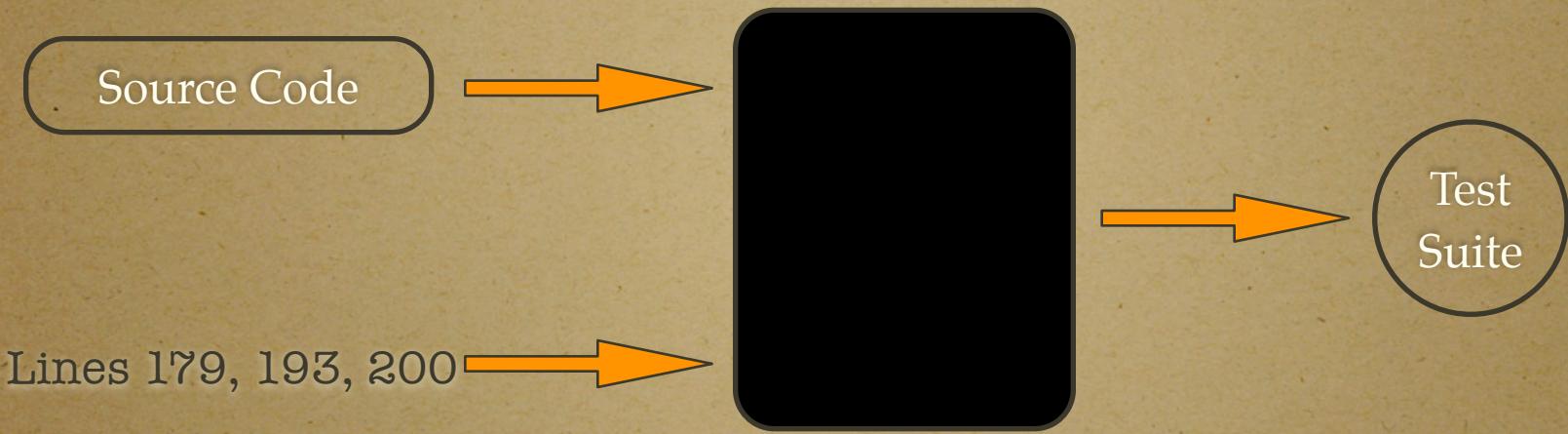


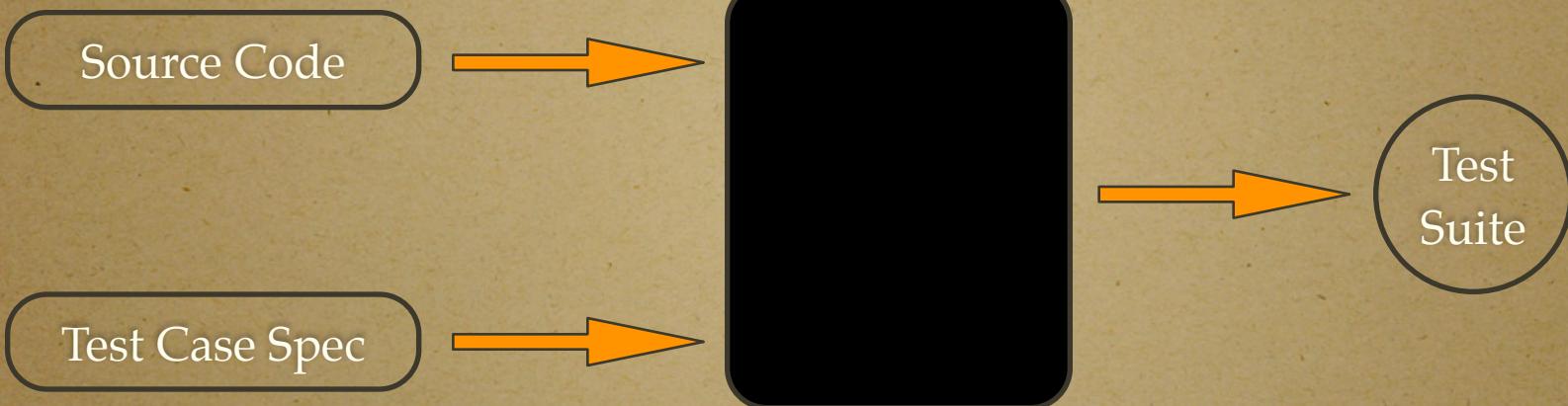
```
313     if (ctrl.io_in[SENS_IMPULS]) {
314         if (!lastImp)
315             if (cmdVal == CMD_NONE) {
316                 if (directionOp) {
317                     if (directionOp) {
318                         if (cnt <= 0)
319                             --cnt;
320                         else
321                             ++cnt;
322                     }
323                 }
324             }
325         else if (lastImp) {
326             if (directionOp) {
327                 if (directionOp) {
328                     if (cnt >= 0)
329                         --cnt;
330                     else
331                         ++cnt;
332                 }
333             }
334         }
335     }
336     if (ctrl.io_in[SENS_BOTTOM]) {
337         if (ctrl.io_in[CMD]) {
338             if (cmdVal == CMD_NONE) {
339                 if (lastImp)
340                     if (directionOp) {
341                         if (cmdVal == CMD_NONE) {
342                             cmd = CMD_NONE;
343                         else
344                             cmd = directionOp;
345                         directionOp = !directionOp;
346                     }
347                 }
348             }
349         }
350     }
351 }
```

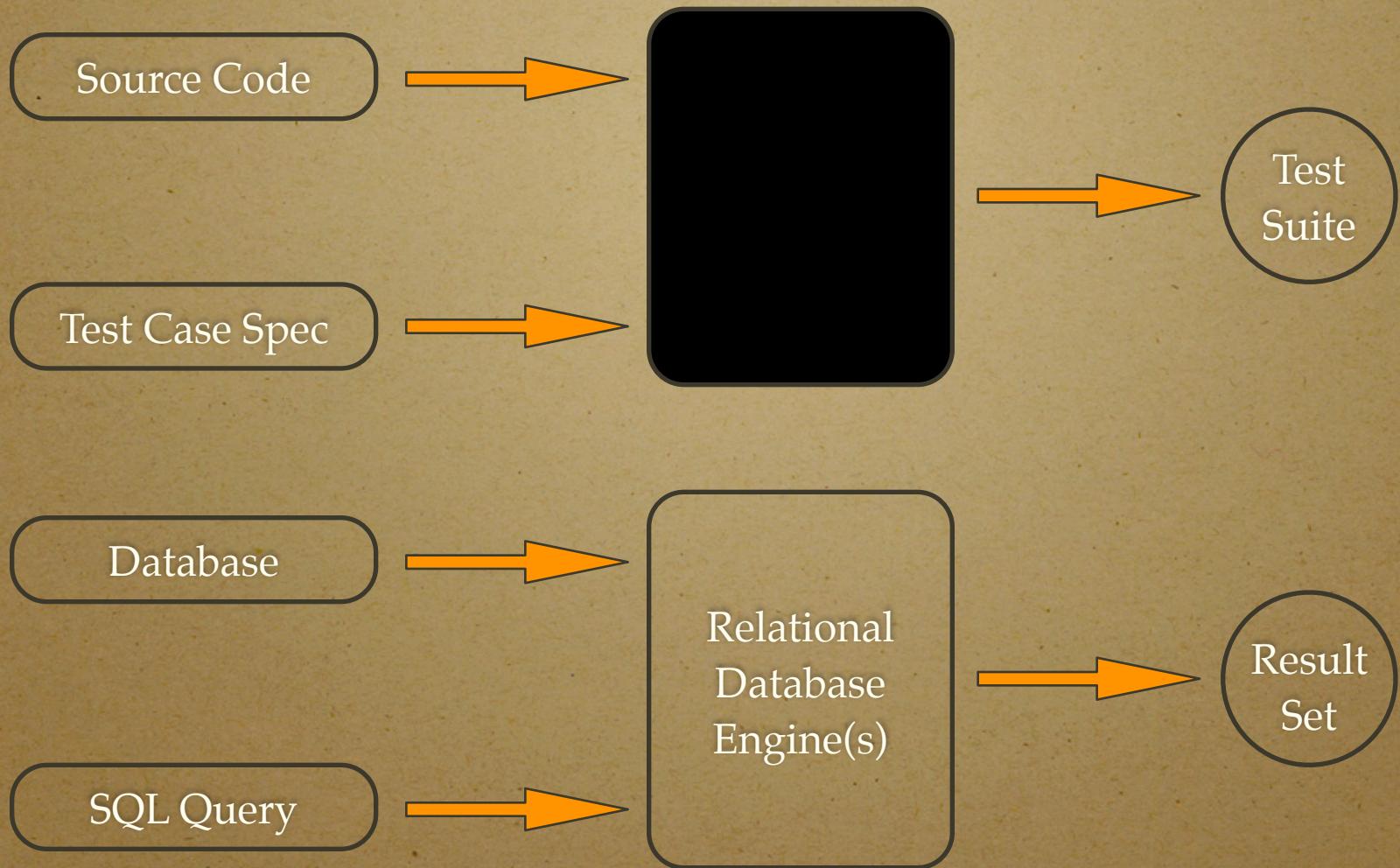


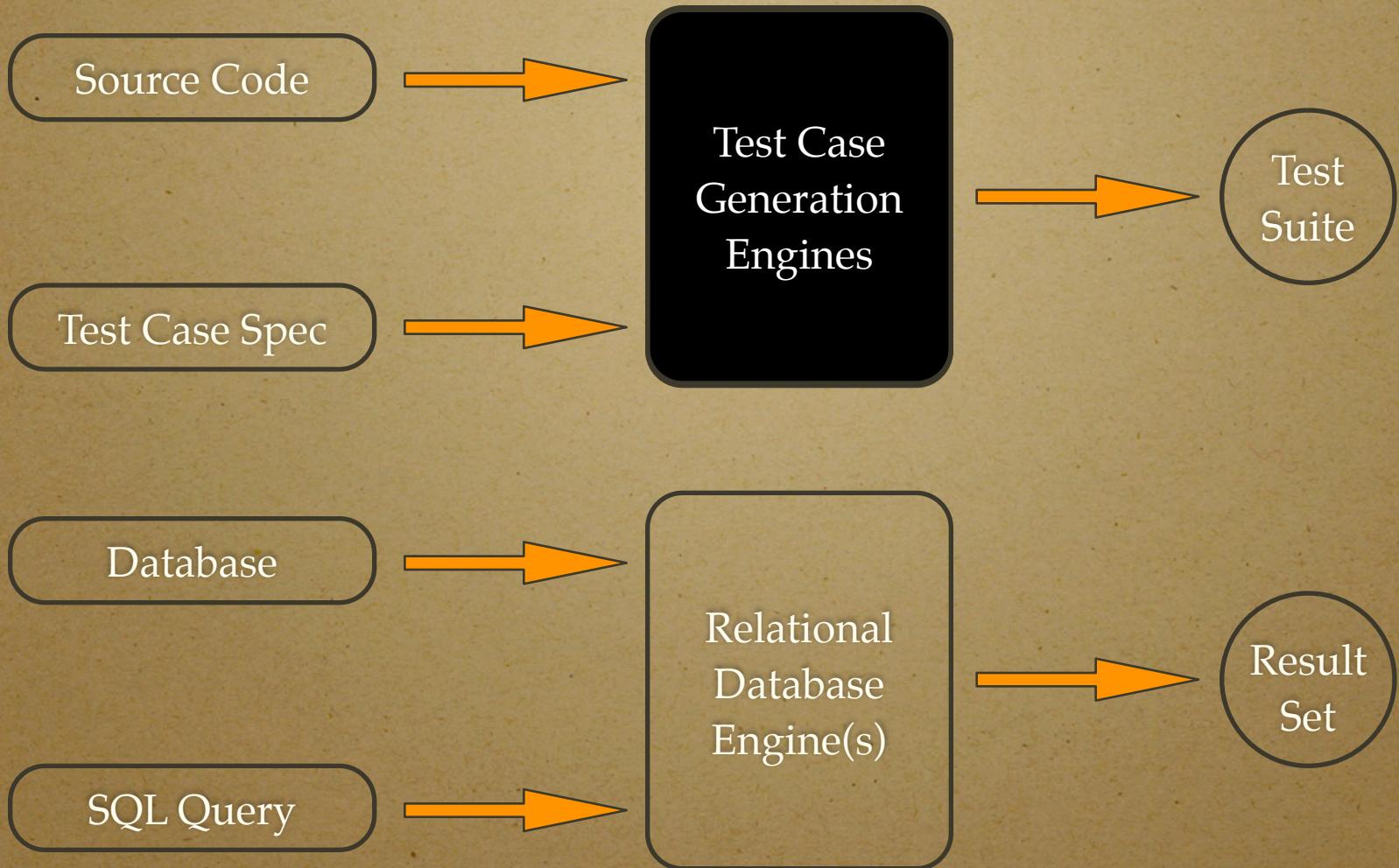
Lines 179, 193, 200



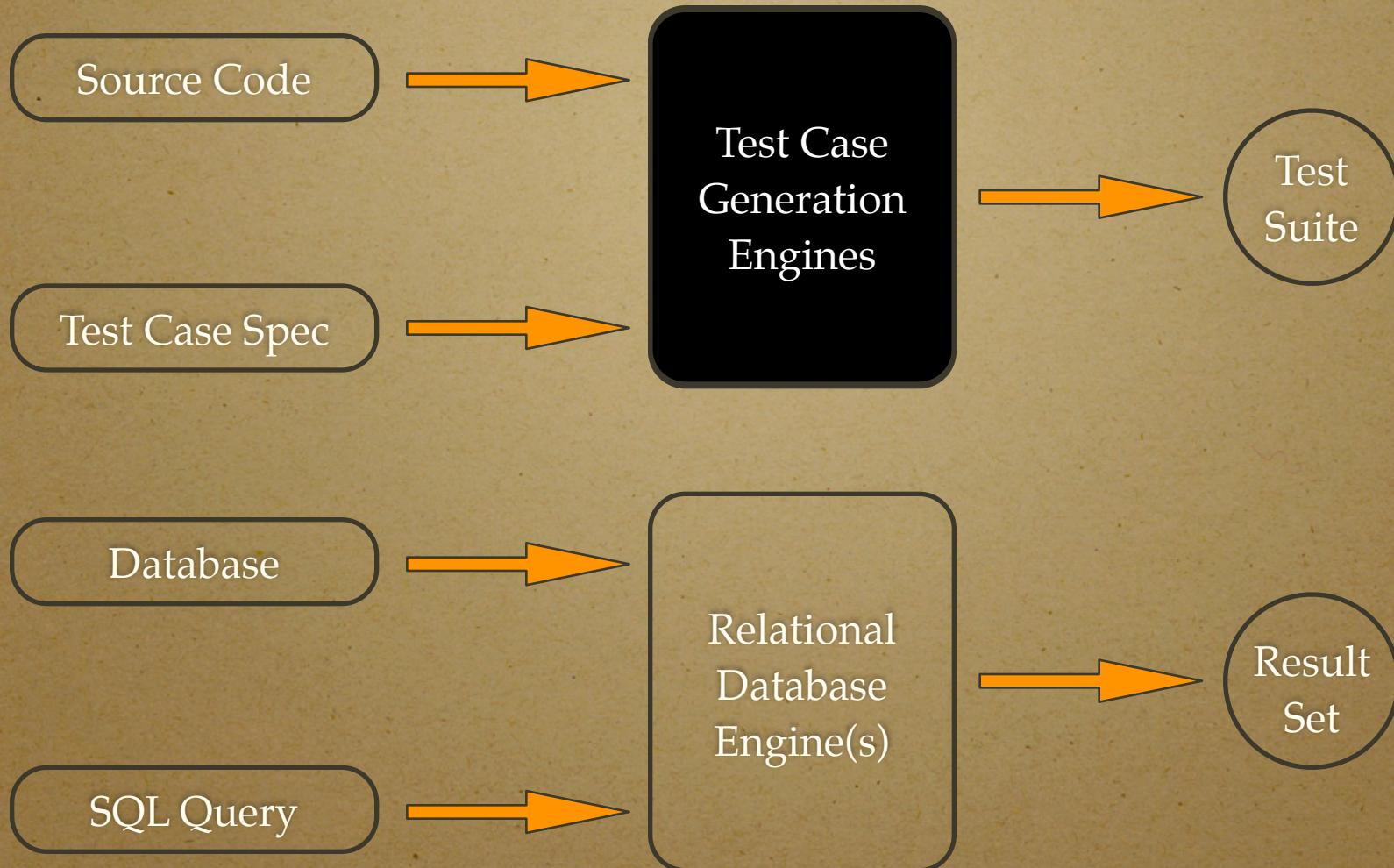




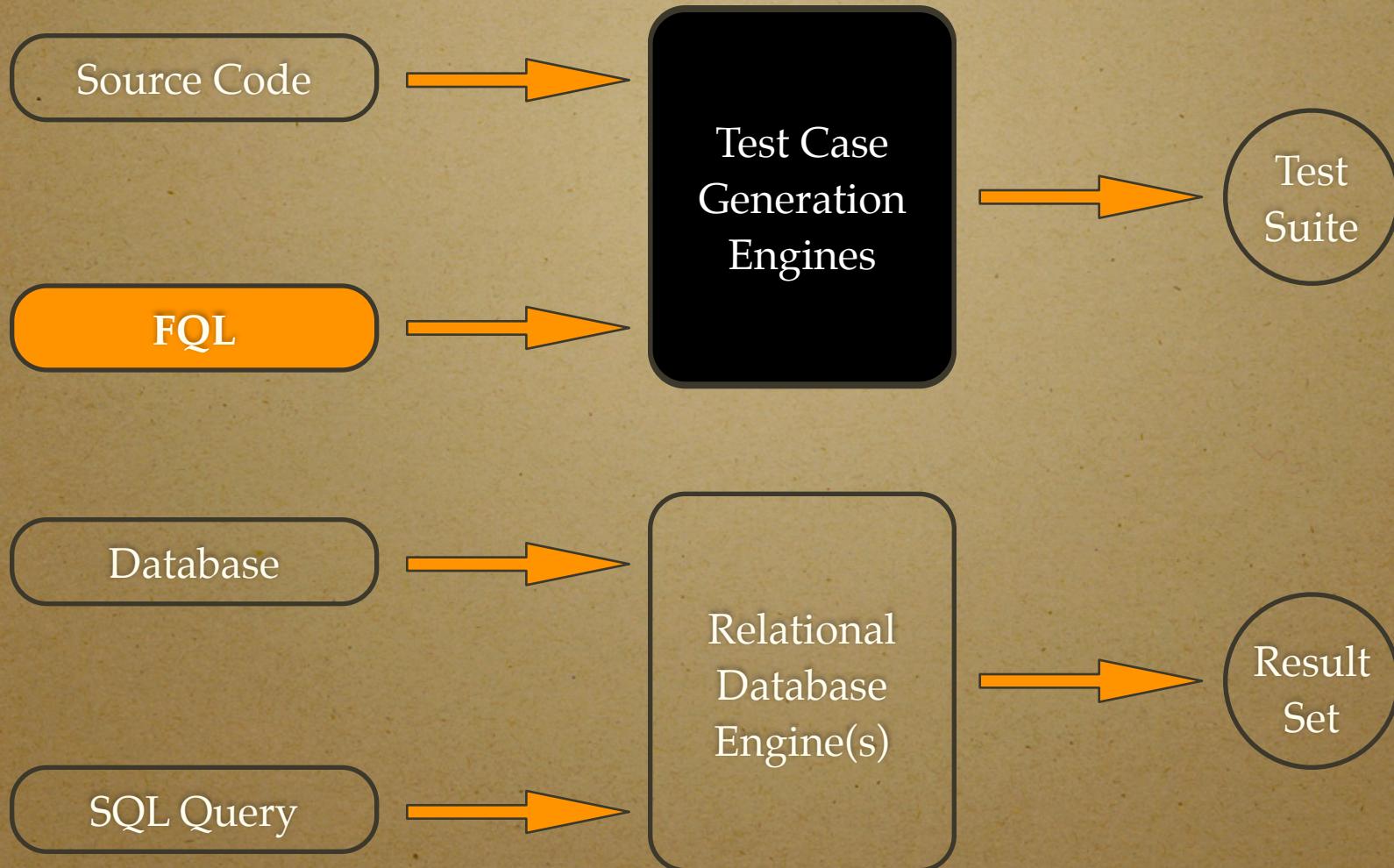




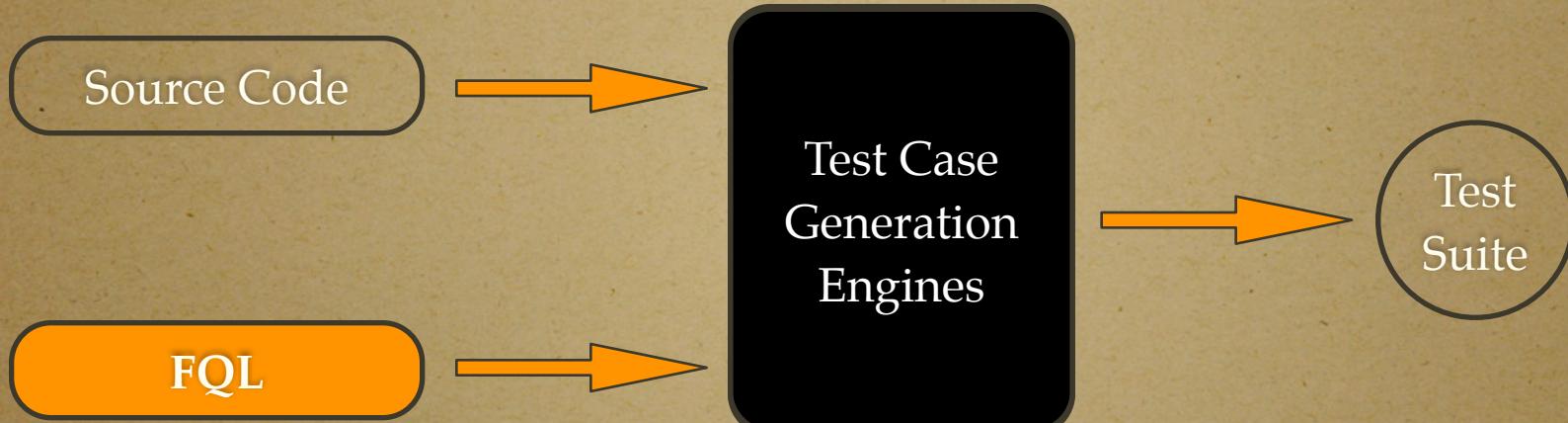
# Query-Driven Program Testing



# Query-Driven Program Testing

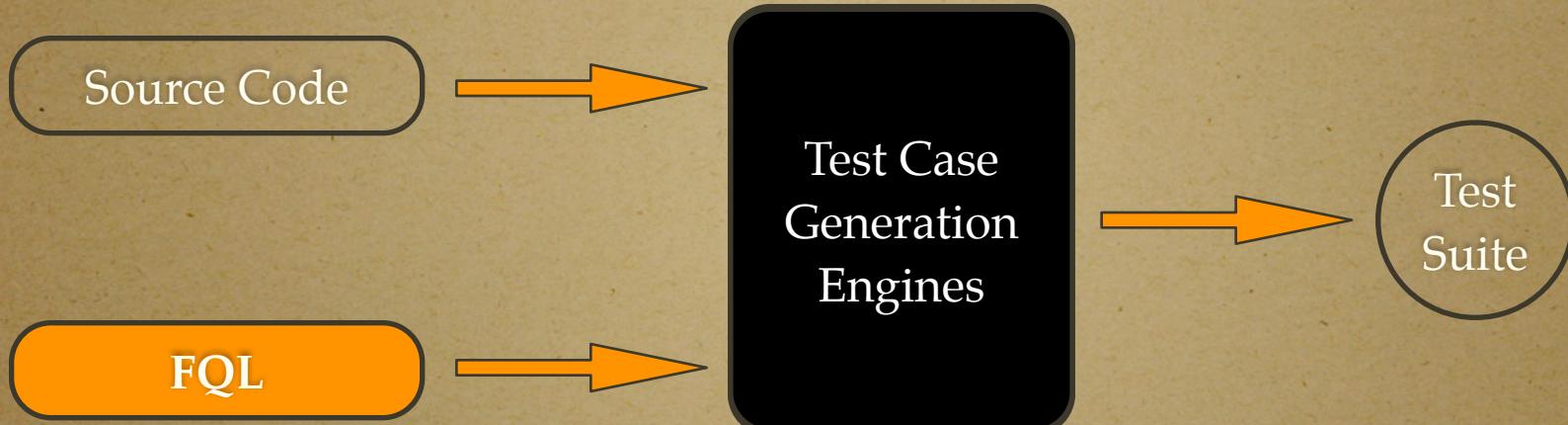


# Query-Driven Program Testing



- Algorithmic approach presented at CAV'08, VMCAI'09
- Two engines already exist
  - FShell: Based on CBMC (C Bounded Model Checker)
  - FlleSh: Uses CPAchecker (Configurable Program Analysis)

# Query-Driven Program Testing



- Algorithmic approach presented at CAV'08, VMCAI'09
- Two engines already exist
  - FShell: Based on CBMC (C Bounded Model Checker)
  - FlleSh: Uses CPAchecker (Configurable Program Analysis)

**<http://code.forsyte.de/fshell>**

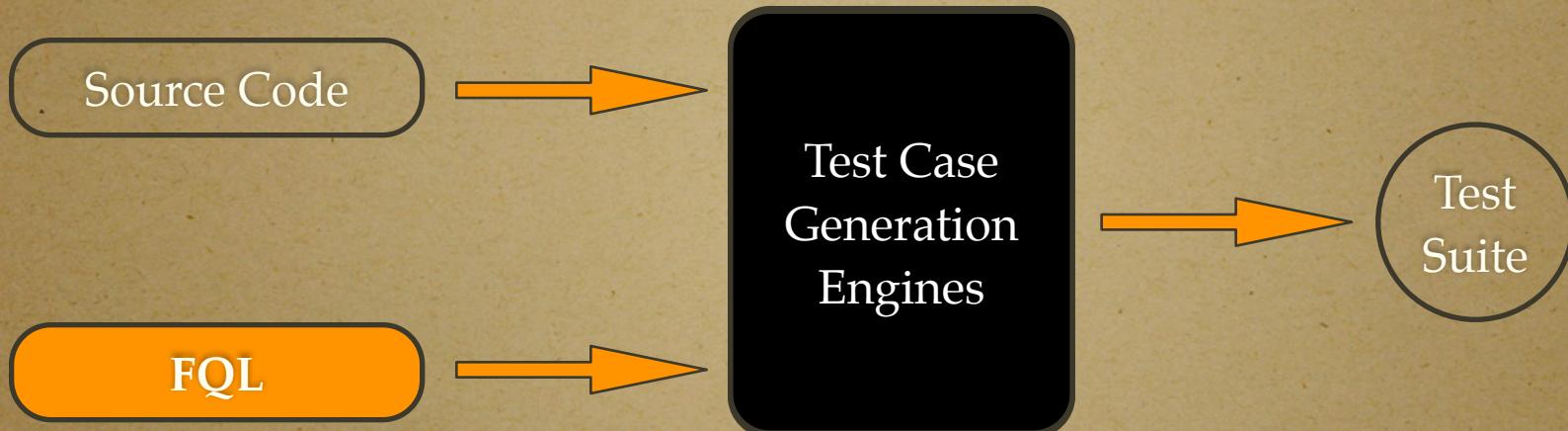
# Using FShell to Compute a Test Suite

```
FShell2> [
```

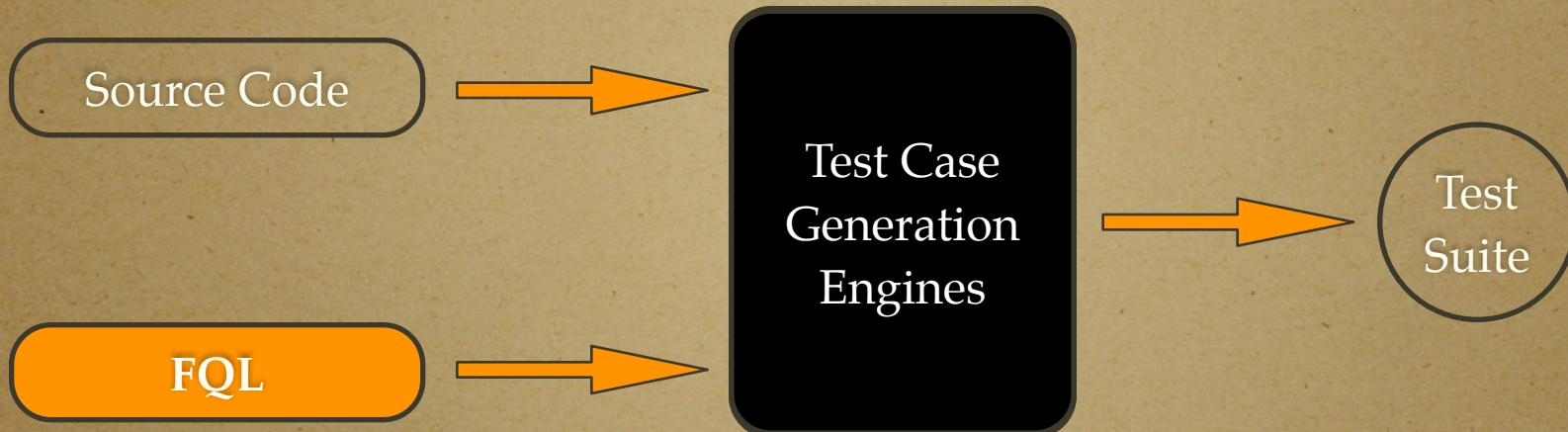


**<http://code.forsyte.de/fshell>**

# Applications of FQL/FShell



# Applications of FQL/FShell



- Test case generation
- Certification
- Requirement-driven testing
- Coverage Evaluation
- Reasoning about test specifications

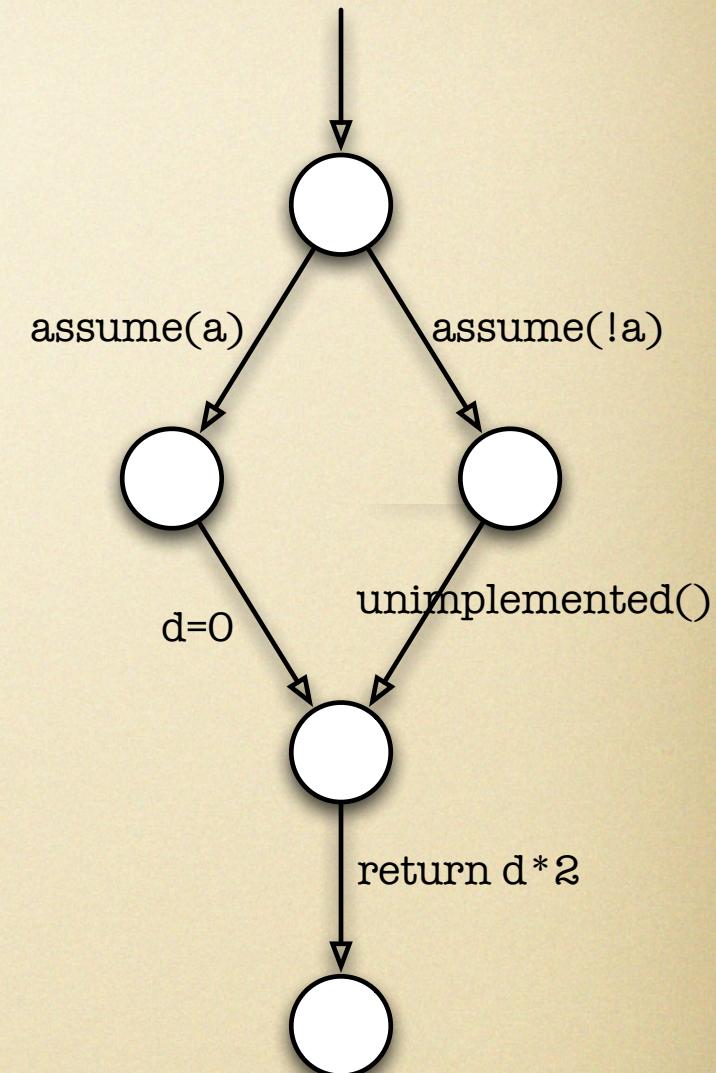
# Software Testing using FQL

- Program Representation: Control Flow Automata
- Describing Single Test Cases
- Describing Test Suites

# Control Flow Automaton (CFA)

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

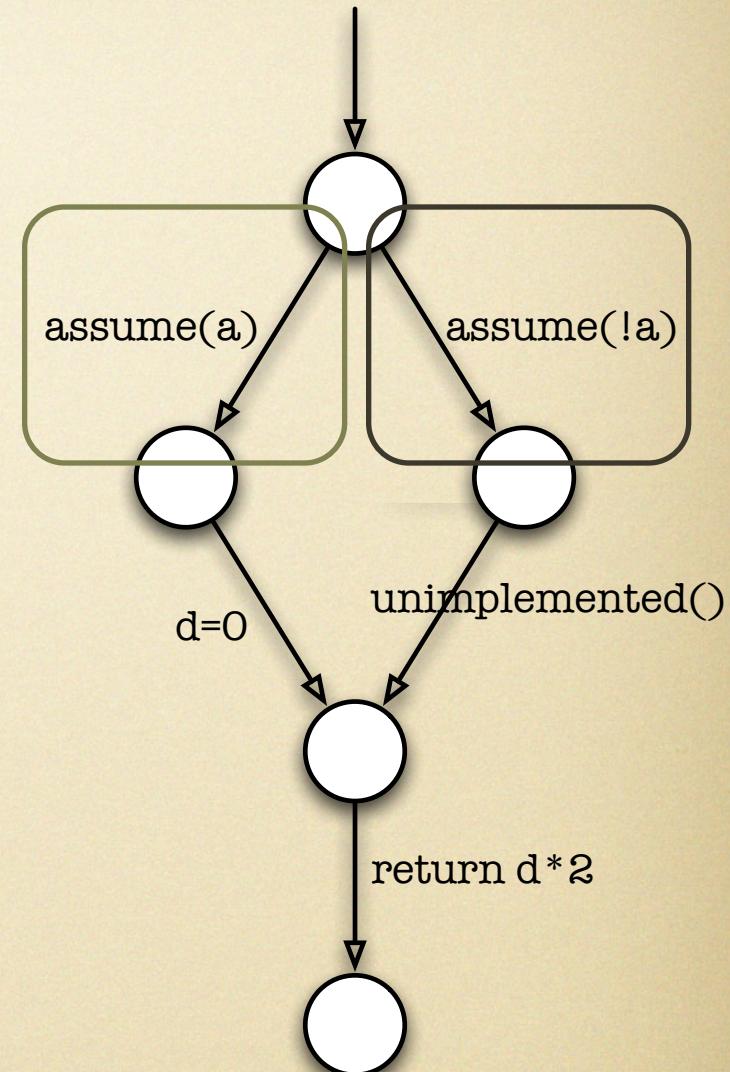
- Henzinger et al. (POPL'02)
- Edges:
  - assume (conditional branches)
  - assignment
  - function call
  - function return
  - skip
- Nodes: locations



# Control Flow Automaton (CFA)

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

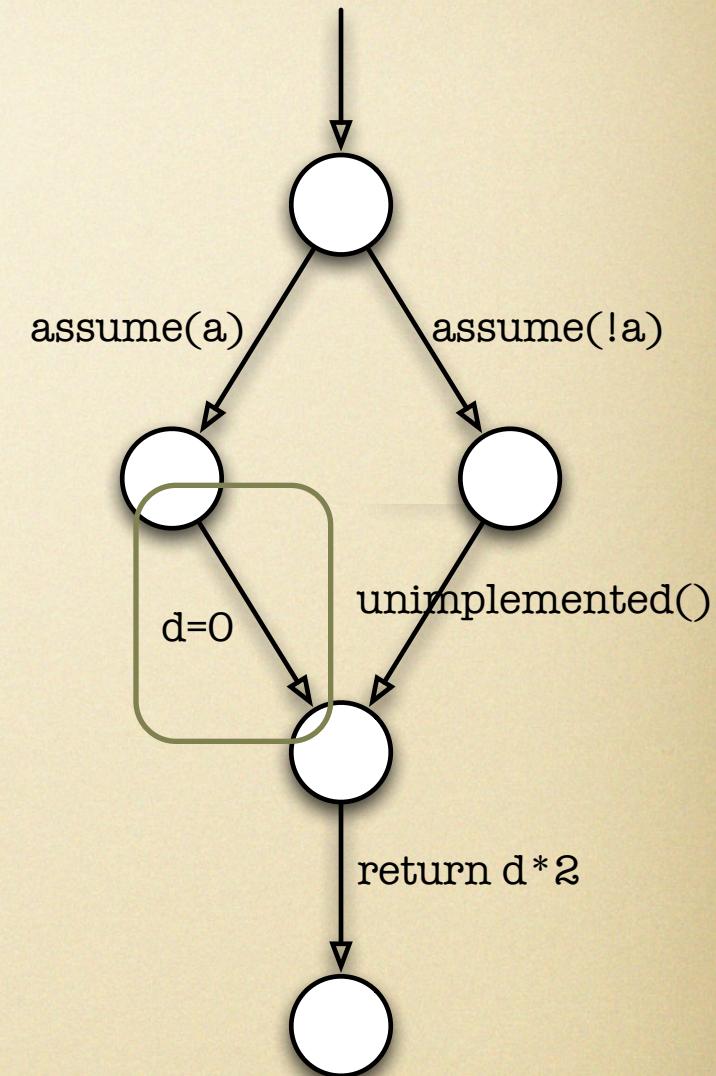
- Henzinger et al. (POPL'02)
- Edges:
  - assume (conditional branches)
  - assignment
  - function call
  - function return
  - skip
- Nodes: locations



# Control Flow Automaton (CFA)

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

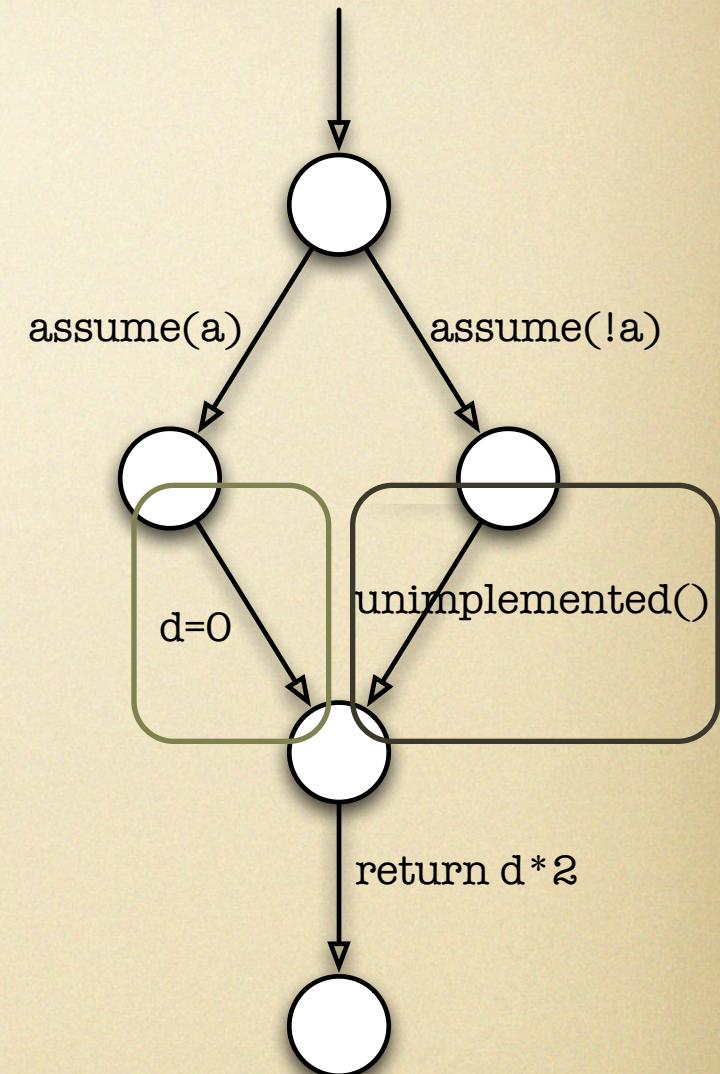
- Henzinger et al. (POPL'02)
- Edges:
  - assume (conditional branches)
  - assignment
  - function call
  - function return
  - skip
- Nodes: locations



# Control Flow Automaton (CFA)

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

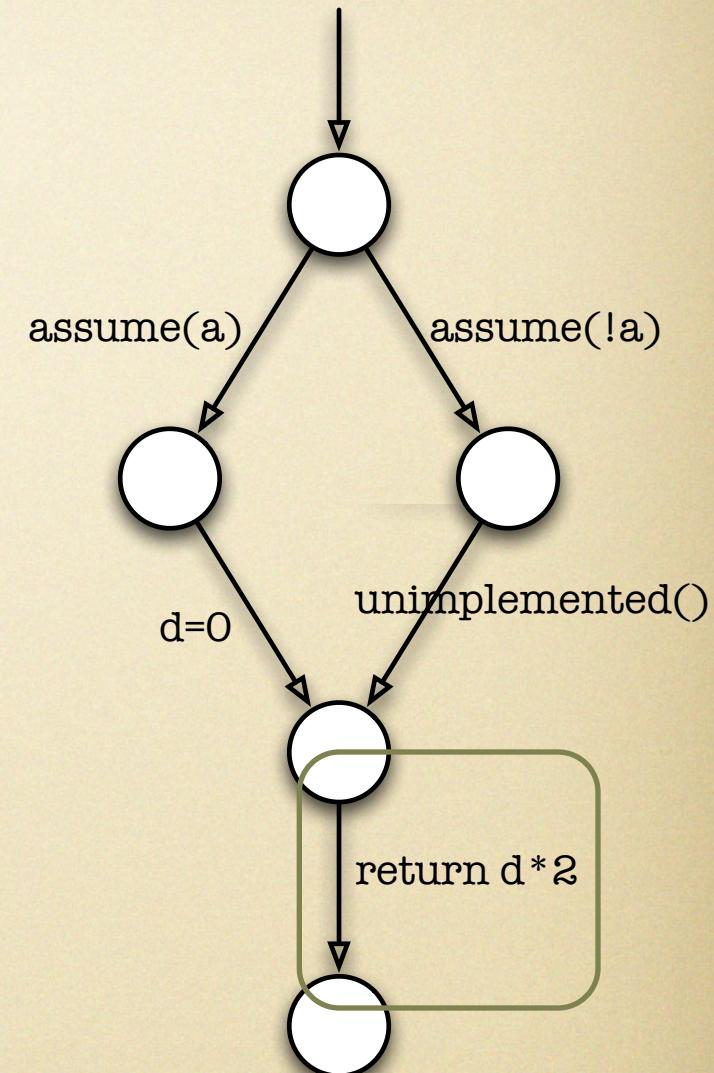
- Henzinger et al. (POPL'02)
- Edges:
  - assume (conditional branches)
  - assignment
  - function call
  - function return
  - skip
- Nodes: locations



# Control Flow Automaton (CFA)

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Henzinger et al. (POPL'02)
- Edges:
  - assume (conditional branches)
  - assignment
  - function call
  - function return
  - skip
- Nodes: locations



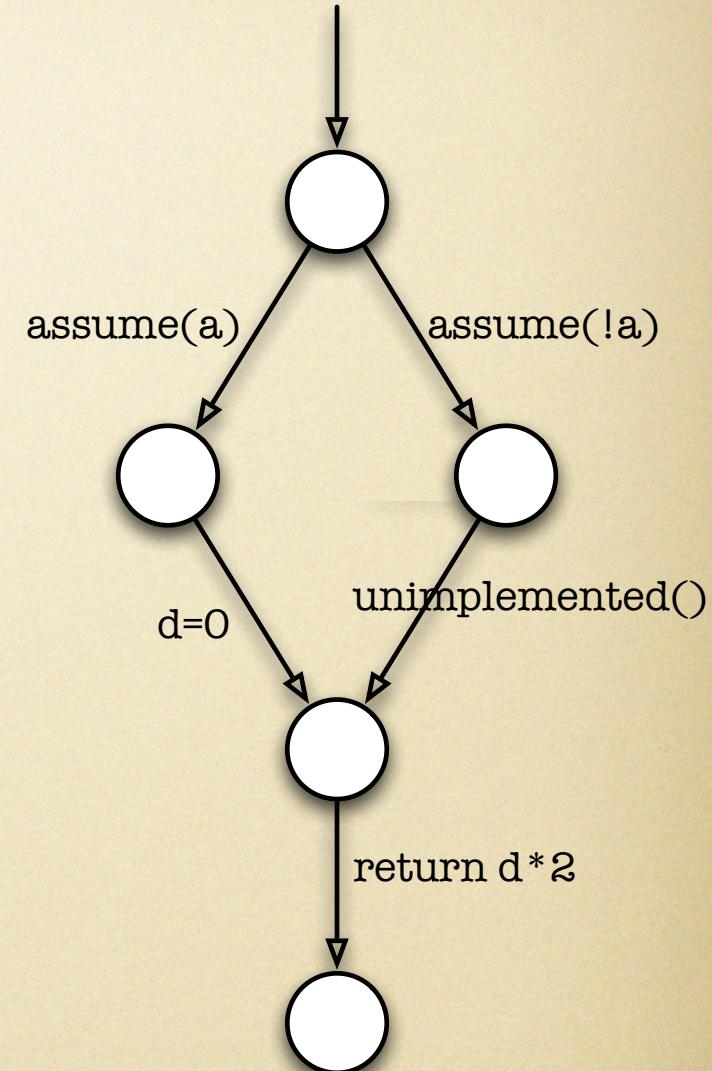
# Software Testing using FQL

Program Representation: Control Flow Automata

- Describing Single Test Cases
- Describing Test Suites

# Filter Functions

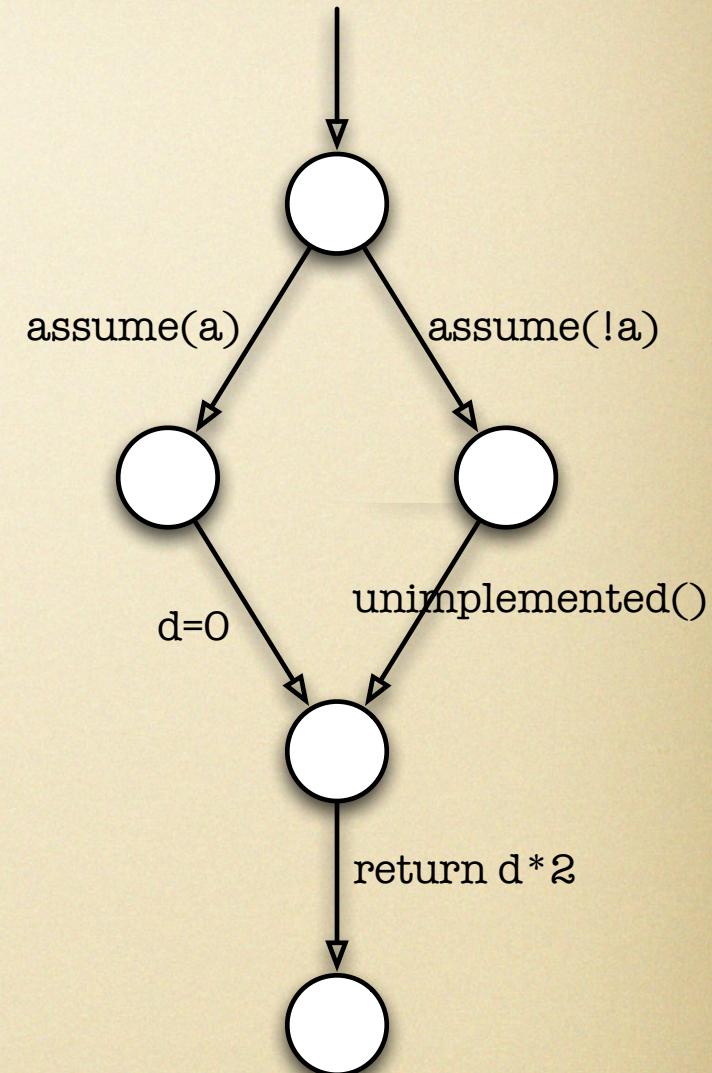
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Filter Functions

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

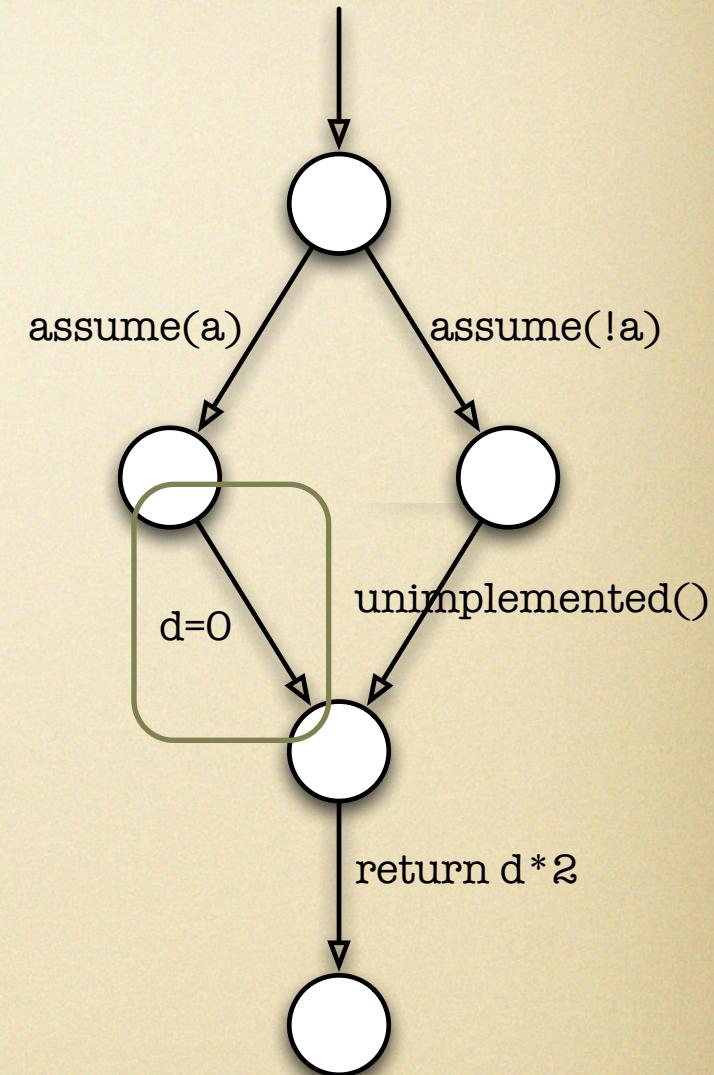
- Cover line 4



# Filter Functions

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

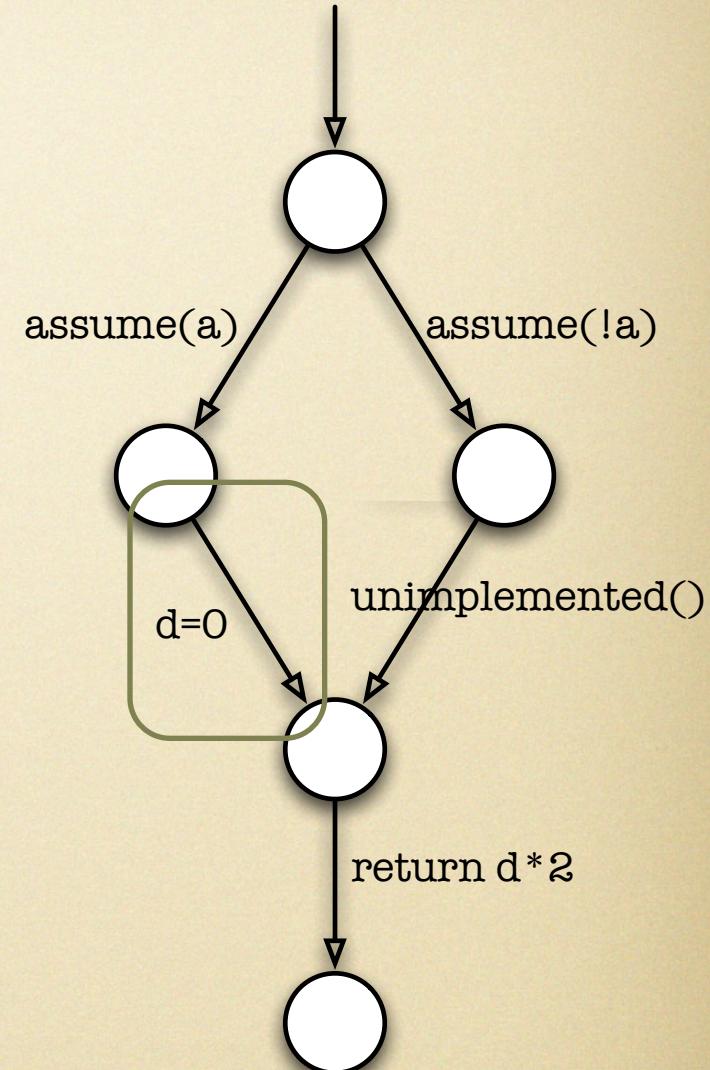
- Cover line 4



# Filter Functions

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Cover line 4
  - **cover “@4”**



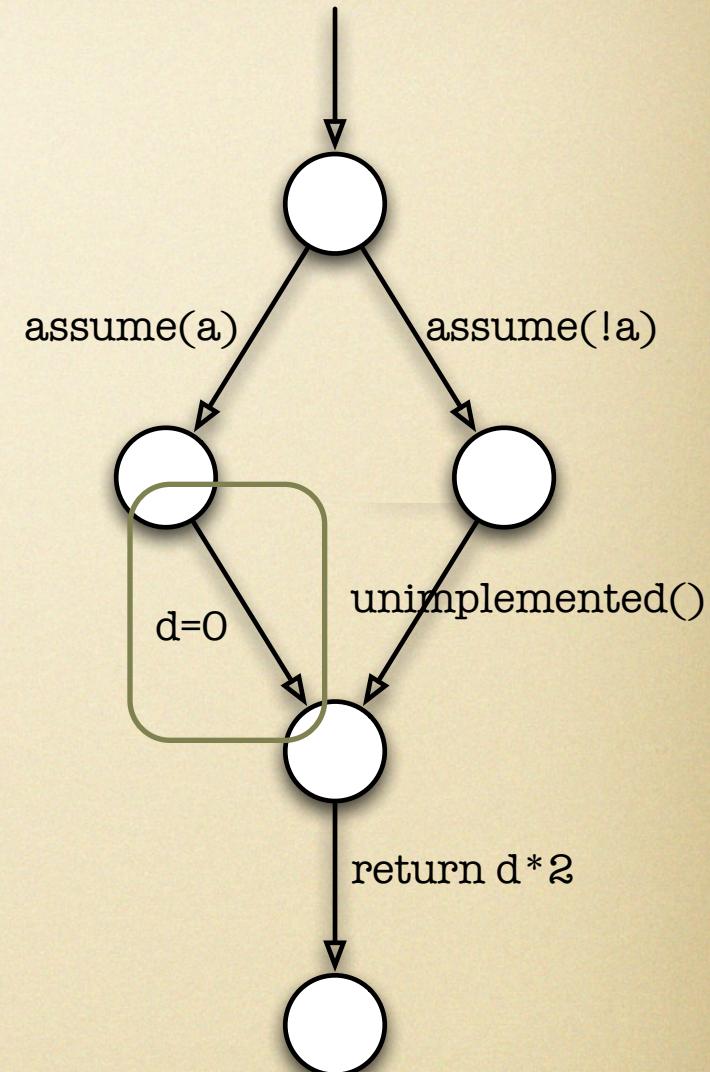
# Filter Functions

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Cover line 4
  - **cover “@4”**

IN:

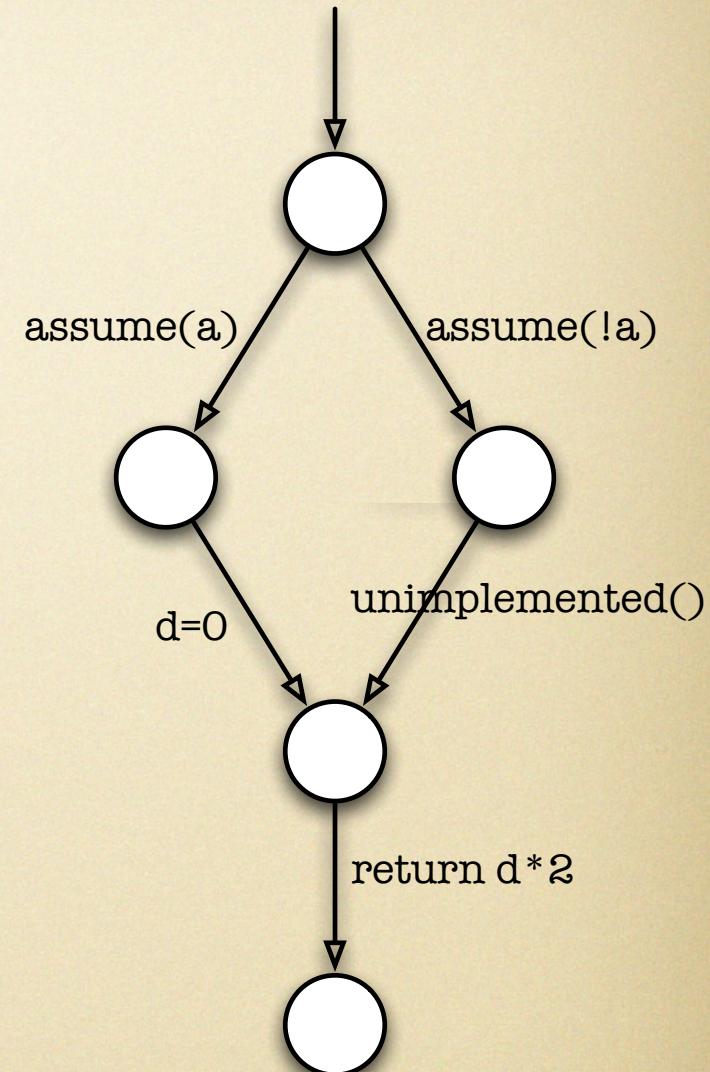
a=2  
d=0



# Filter Functions

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Cover *line 4*
  - **cover “@4”**
- IN:
  - a=2
  - d=0
- Cover *some statement*



# Filter Functions

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

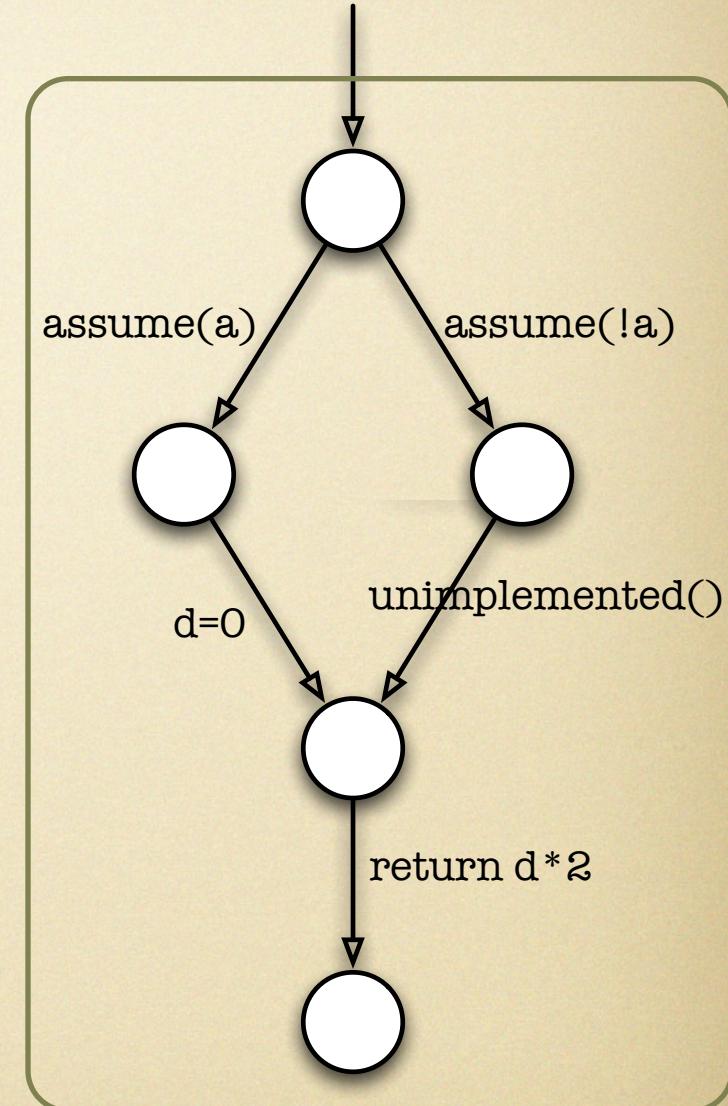
- Cover *line 4*

- **cover** “@4”

IN:

a=2  
d=0

- Cover *some statement*



# Filter Functions

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Cover *line 4*

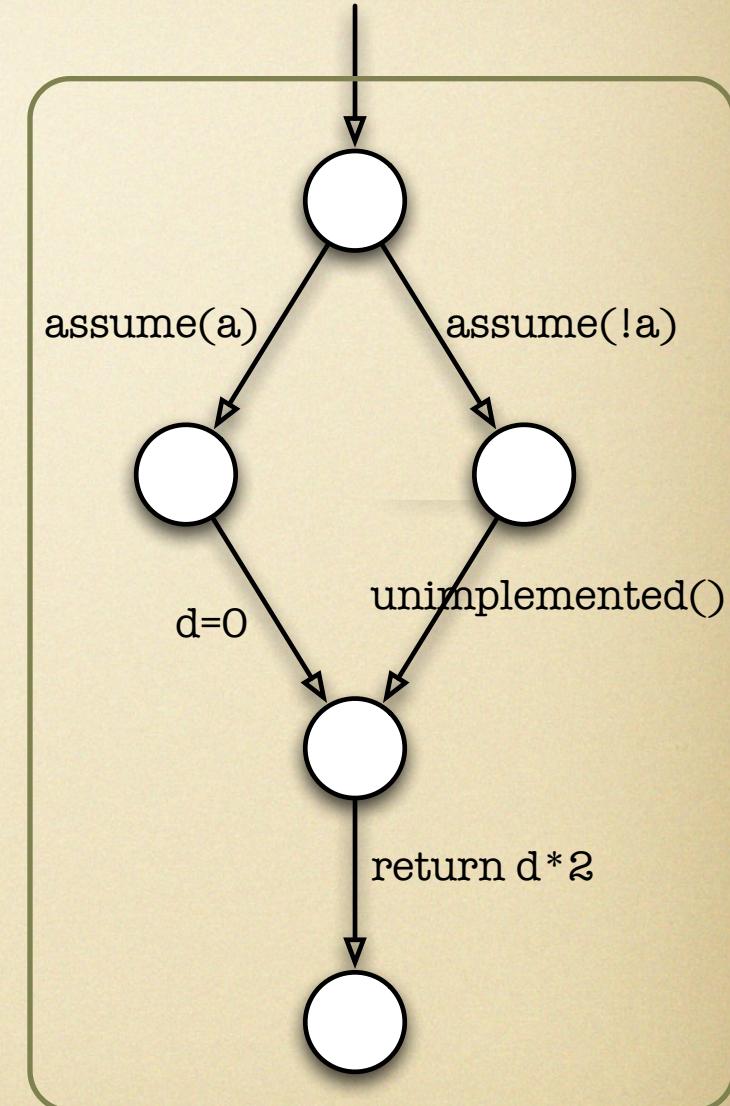
- **cover** “@4”

IN:

a=2  
d=0

- Cover *some statement*

- **cover** “ID”



# Filter Functions

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Cover *line 4*

- **cover "4"**

IN:

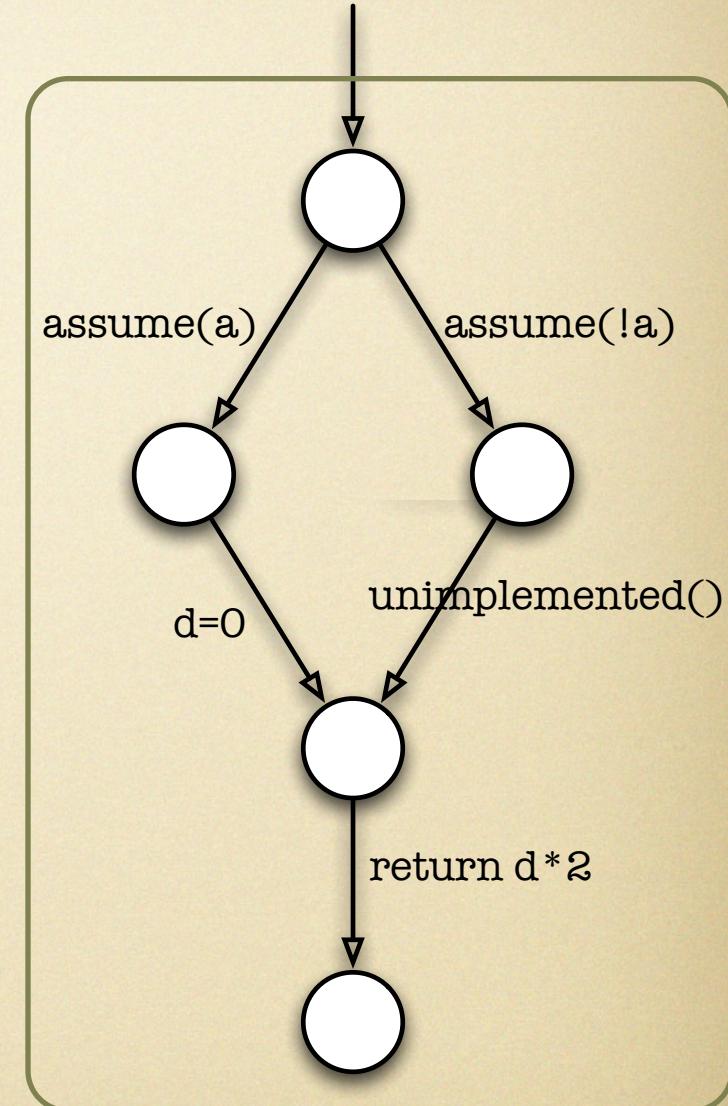
a=2  
d=0

- Cover *some statement*

- **cover "ID"**

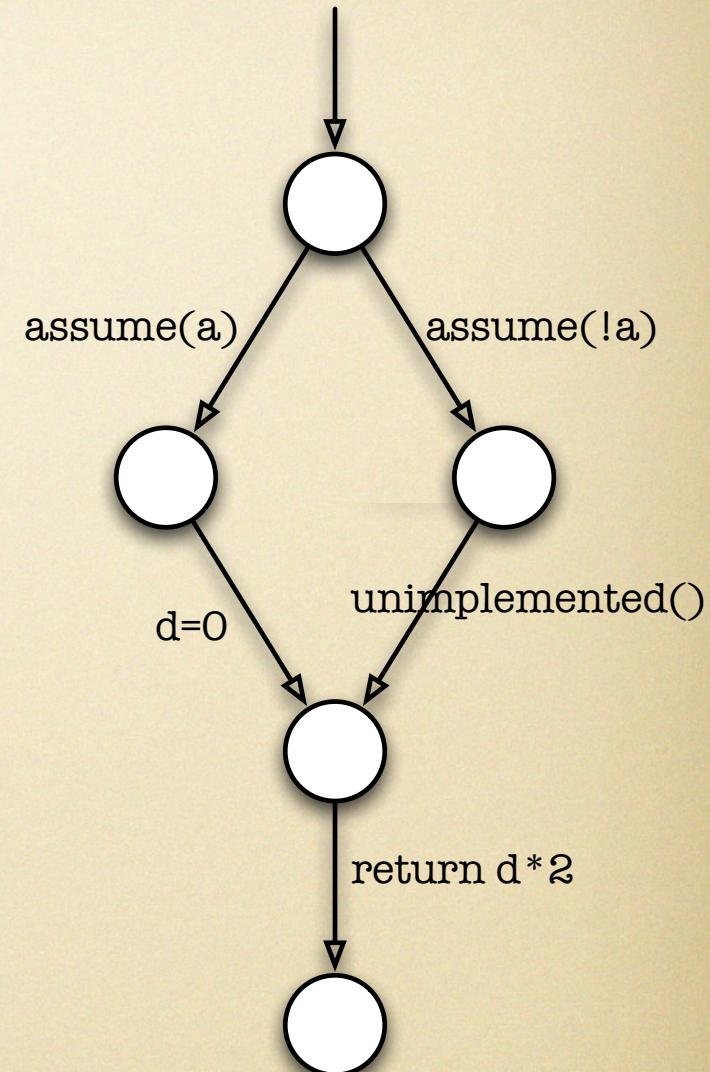
IN:

a=8192  
d=0



# Describing Single Test Cases

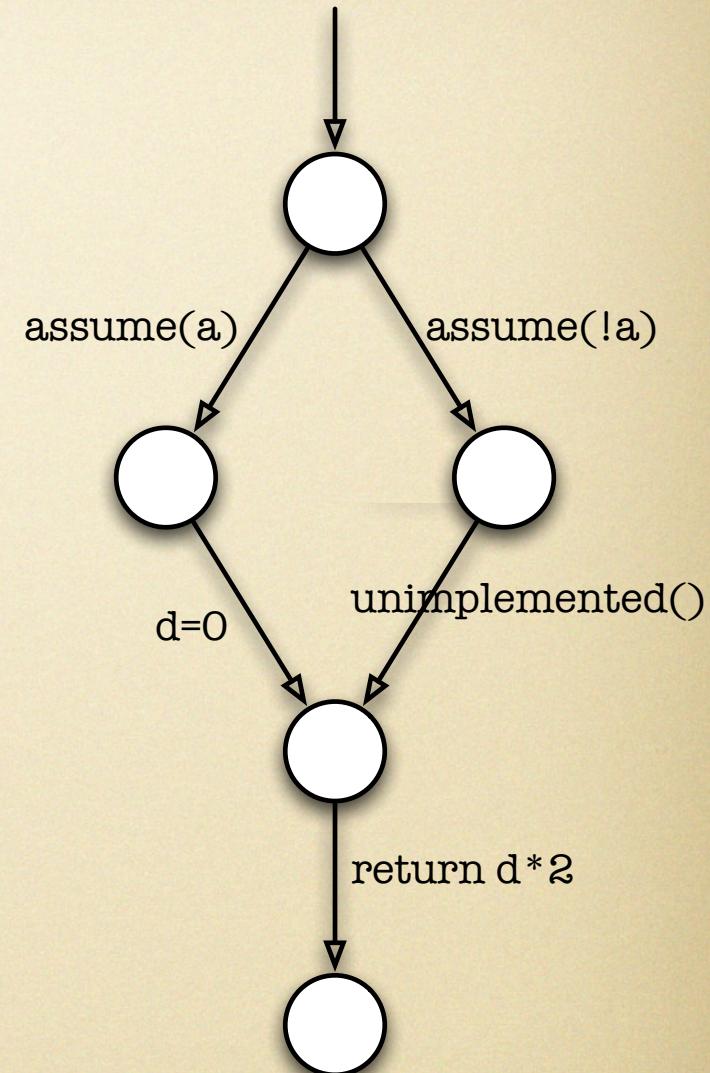
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

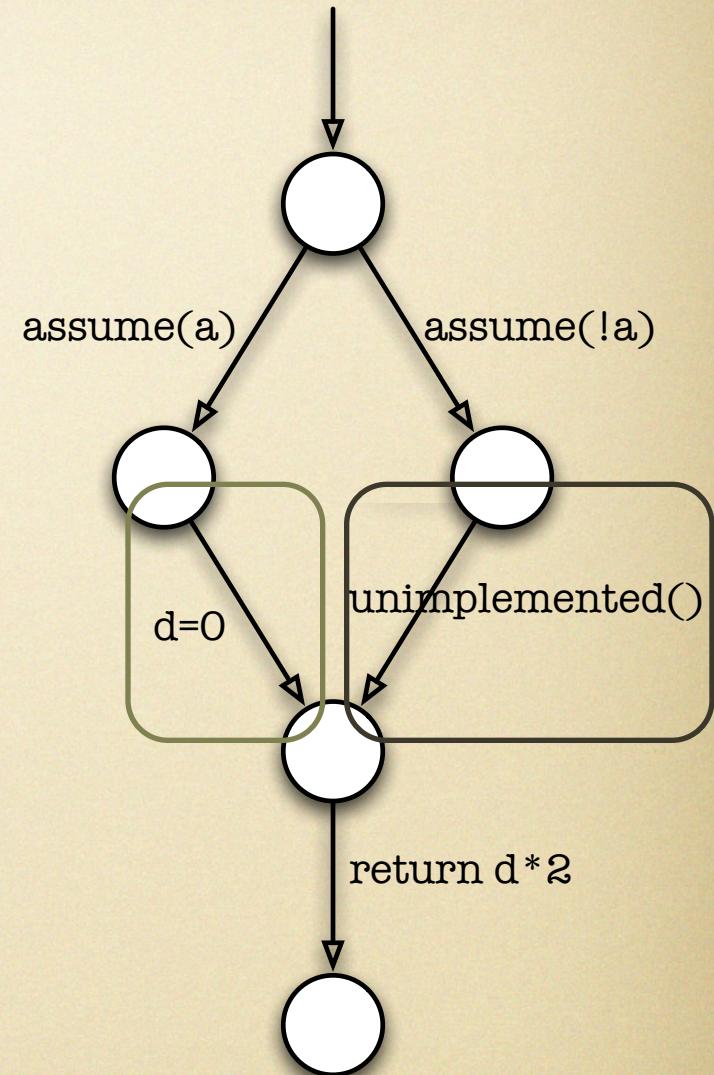
- Cover *line 4 or line 6*



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

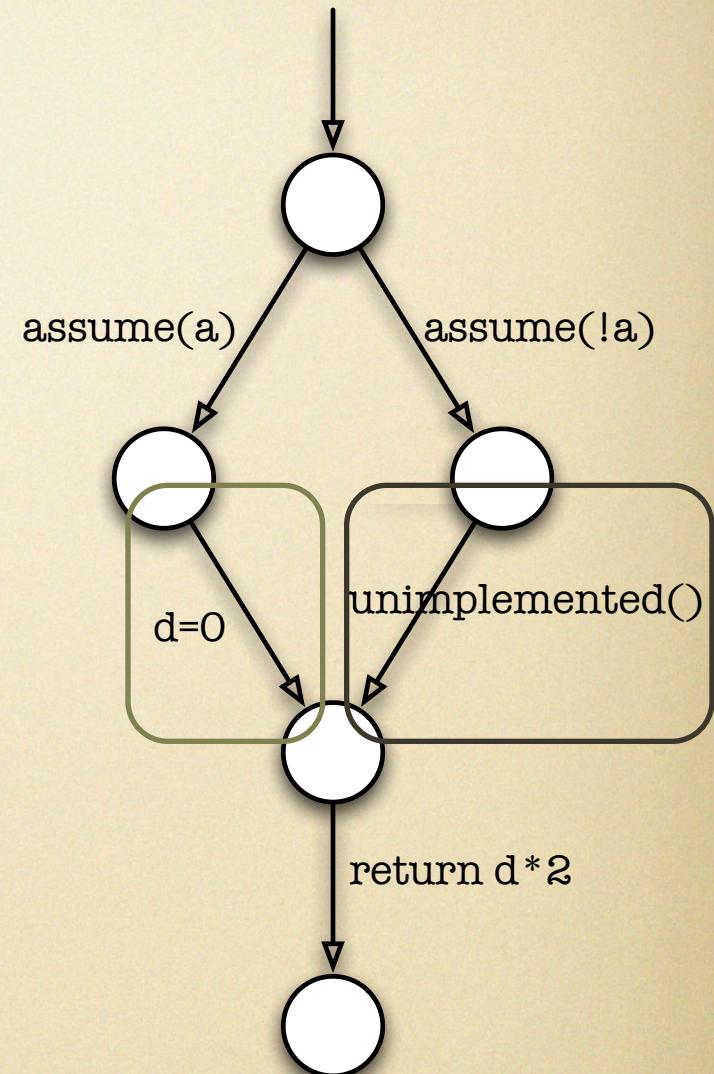
- Cover line 4 or line 6



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

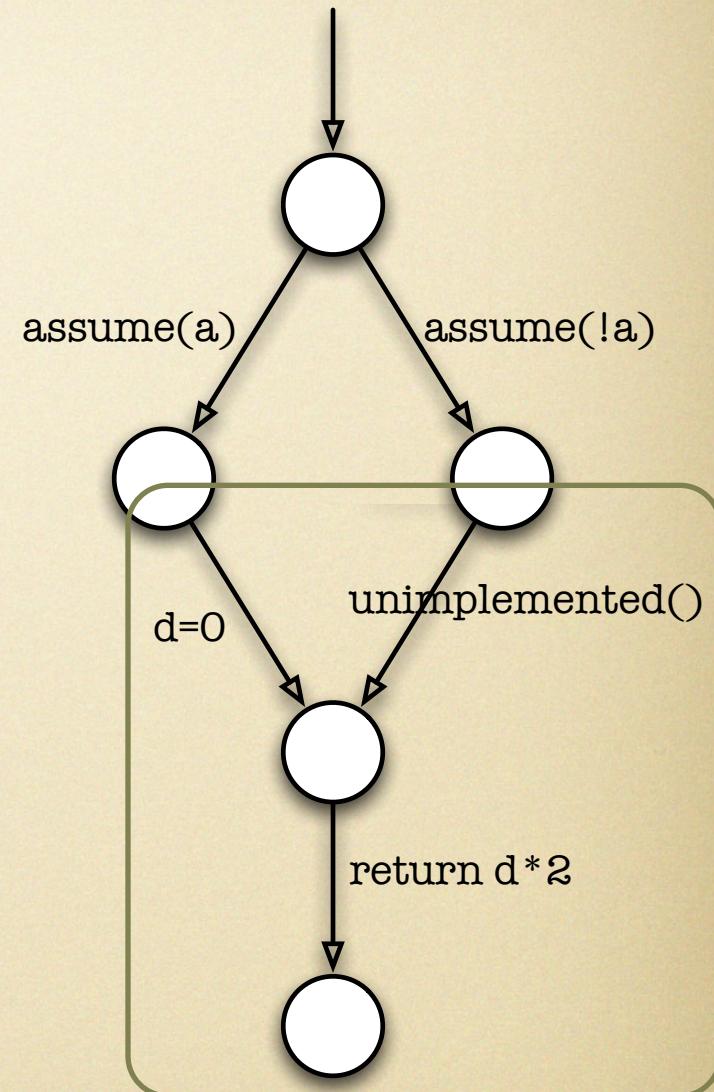
- Cover *line 4 or line 6*
  - **cover “@4+@6”**



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

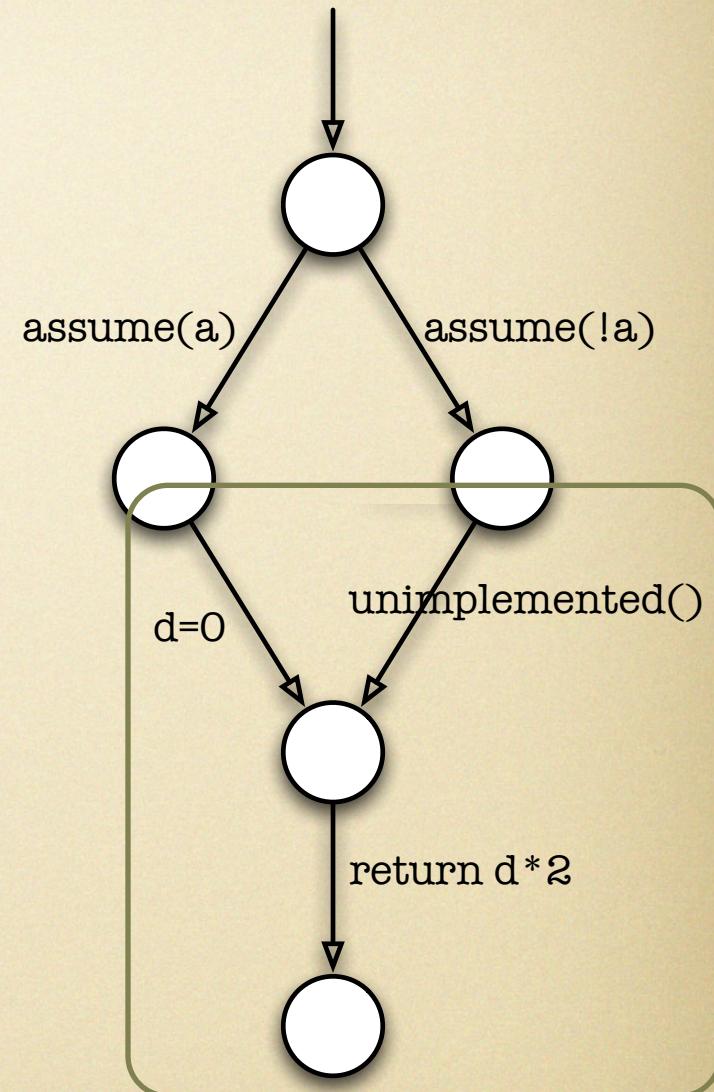
- Cover *line 4 or line 6*
  - **cover “@4+@6”**
- Cover a *basic block*



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

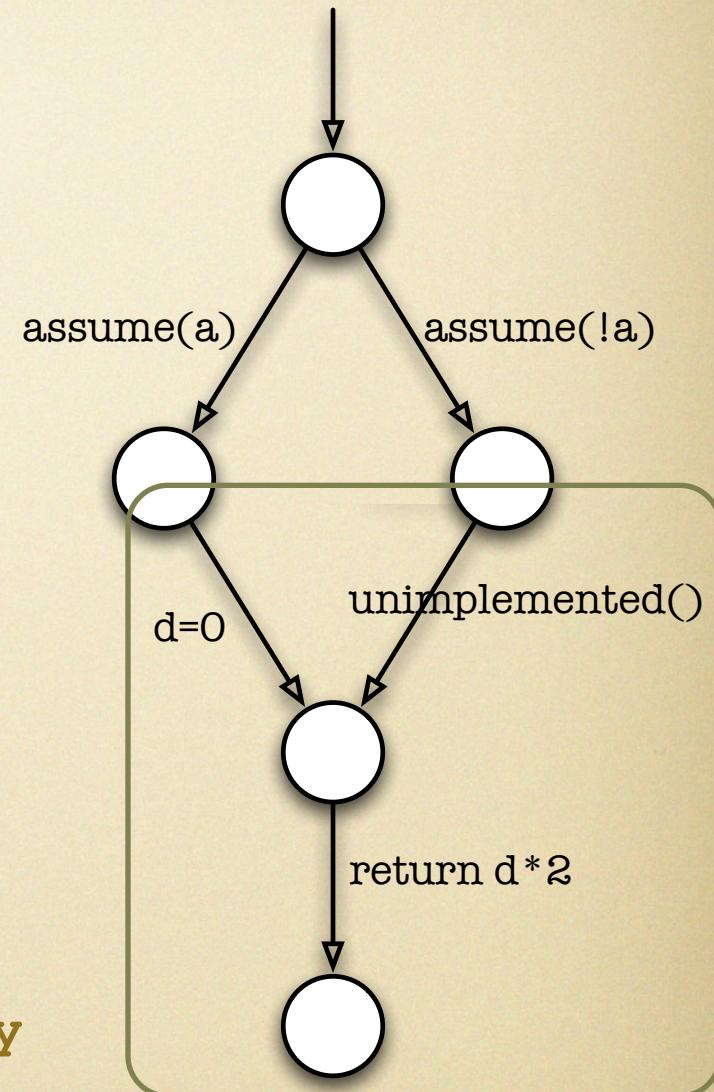
- Cover *line 4 or line 6*
  - **cover “@4+@6”**
- Cover a *basic block*
  - **cover “@basicblockentry”**



# Describing Single Test Cases

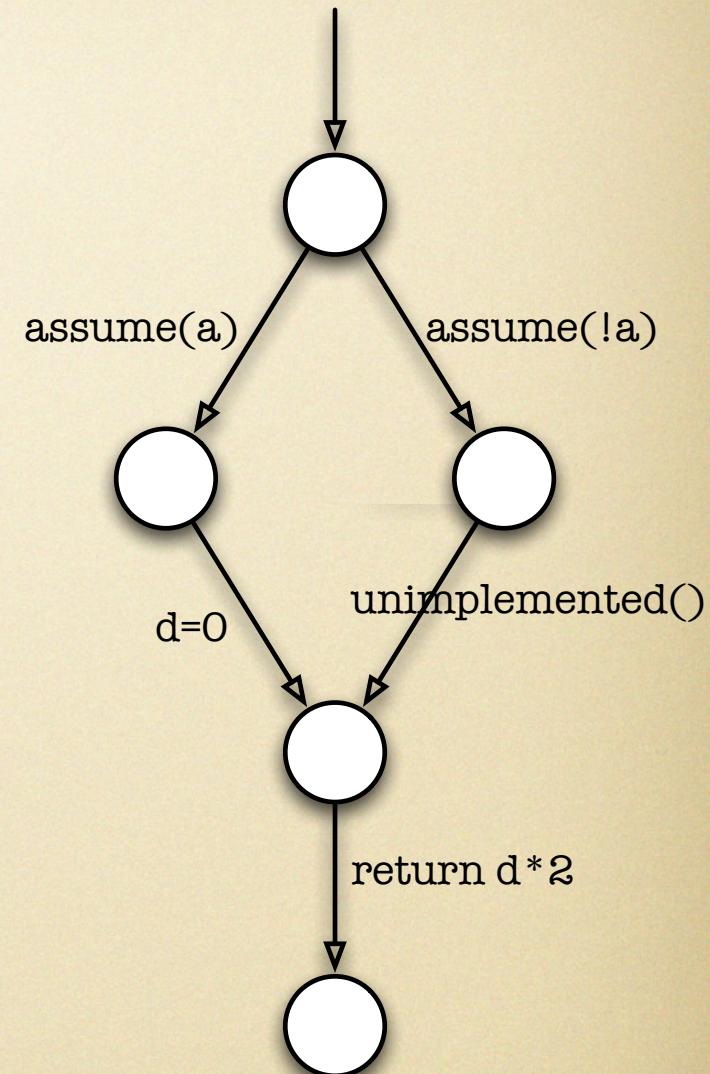
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Cover *line 4 or line 6*
  - **cover “@4+@6”**
- Cover a *basic block*
  - **cover “@basicblockentry”**
- For this example **@basicblockentry** is tantamount to **@4+@6+@7**



# Describing Single Test Cases

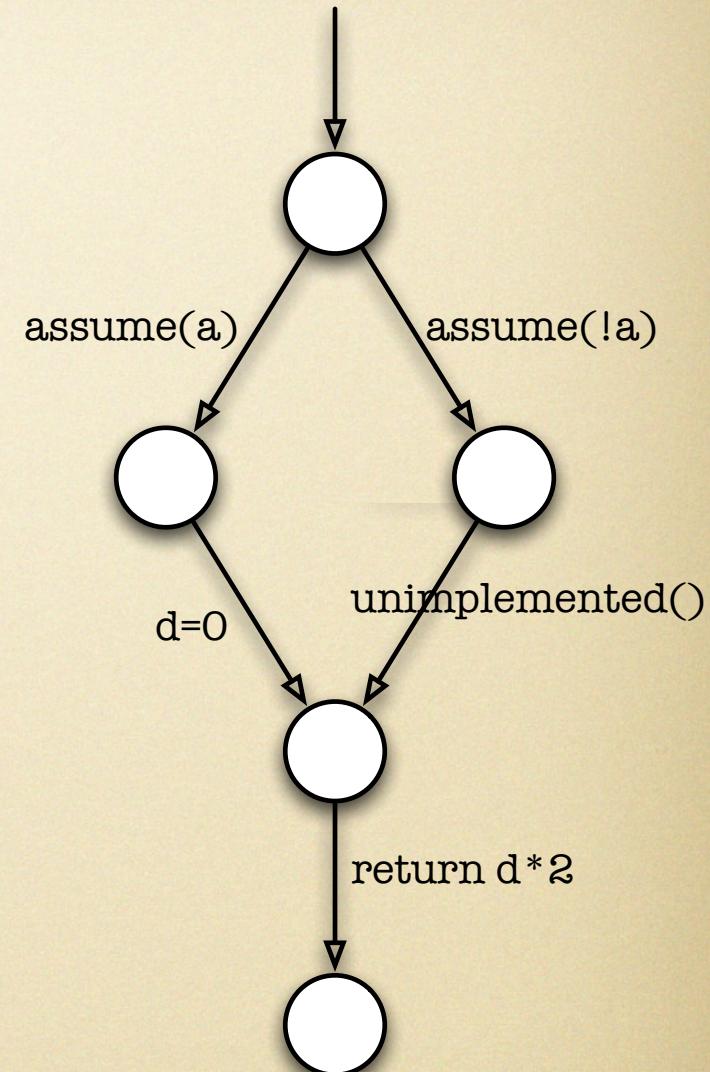
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

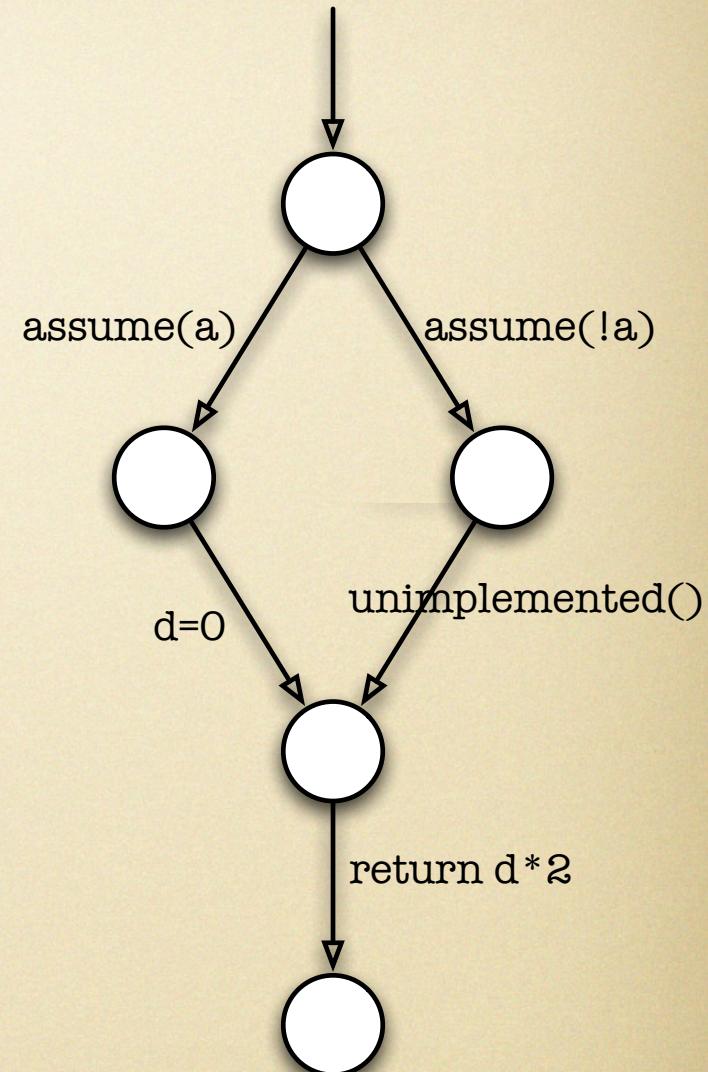
- Test case is execution (program path)



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

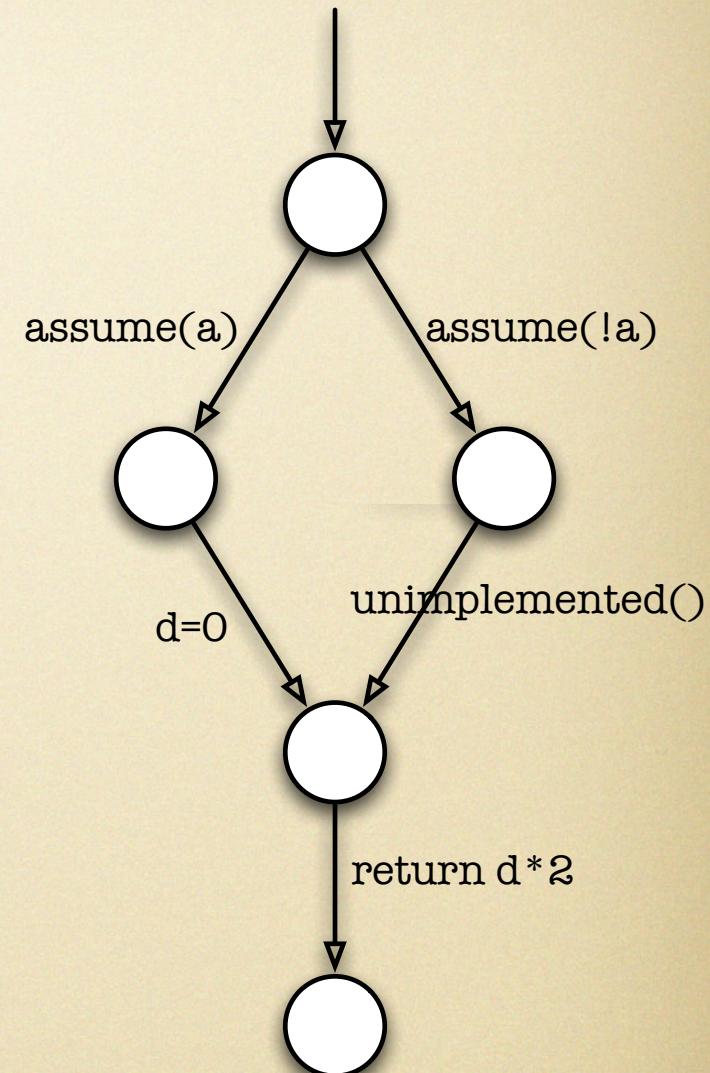
- Test case is execution (program path)
- Intuitively: “cover line 4” =



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

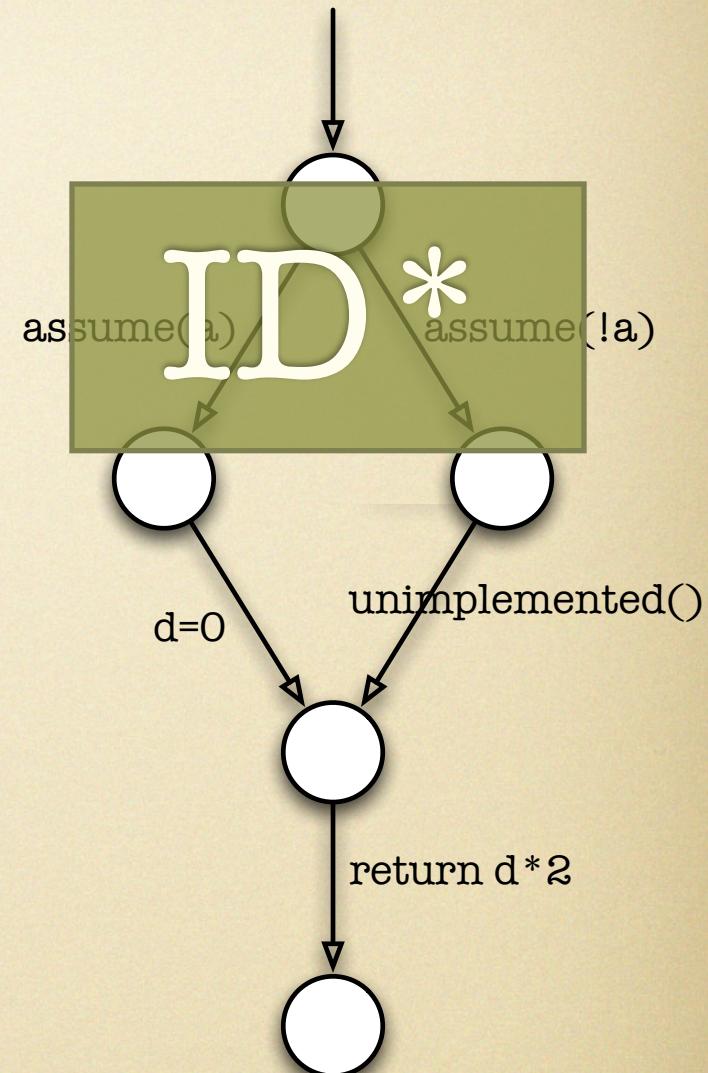
- Test case is execution (program path)
- Intuitively: “cover line 4” =
  - Execute some statement(s)



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

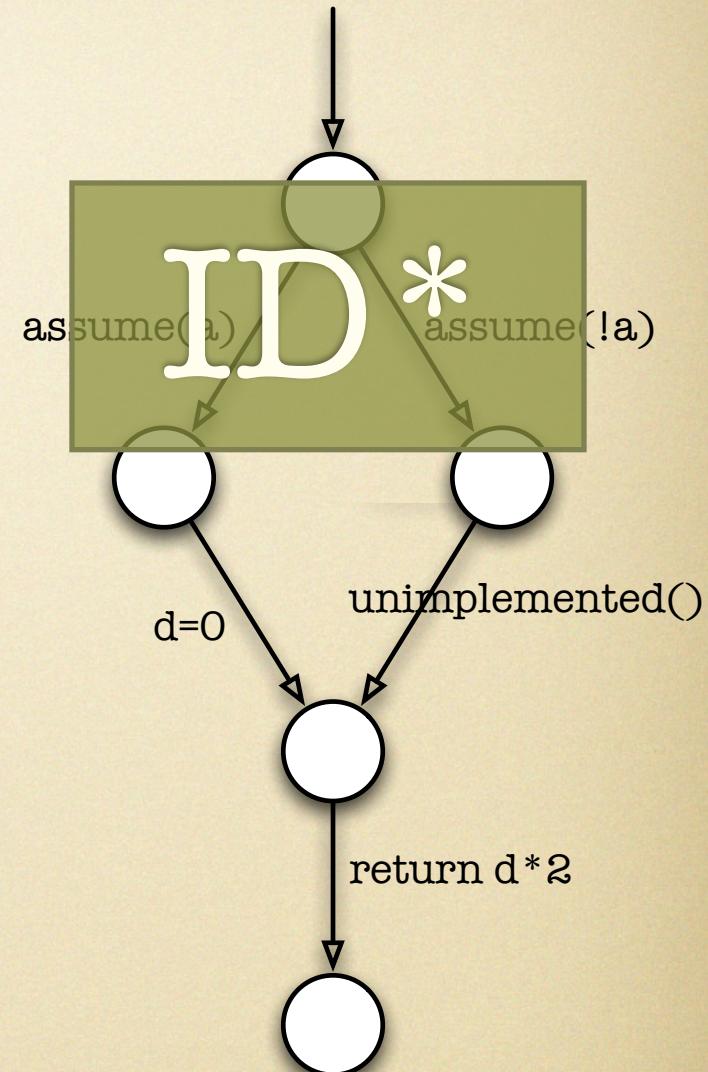
- Test case is execution (program path)
- Intuitively: “cover line 4” =
  - Execute some statement(s)



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

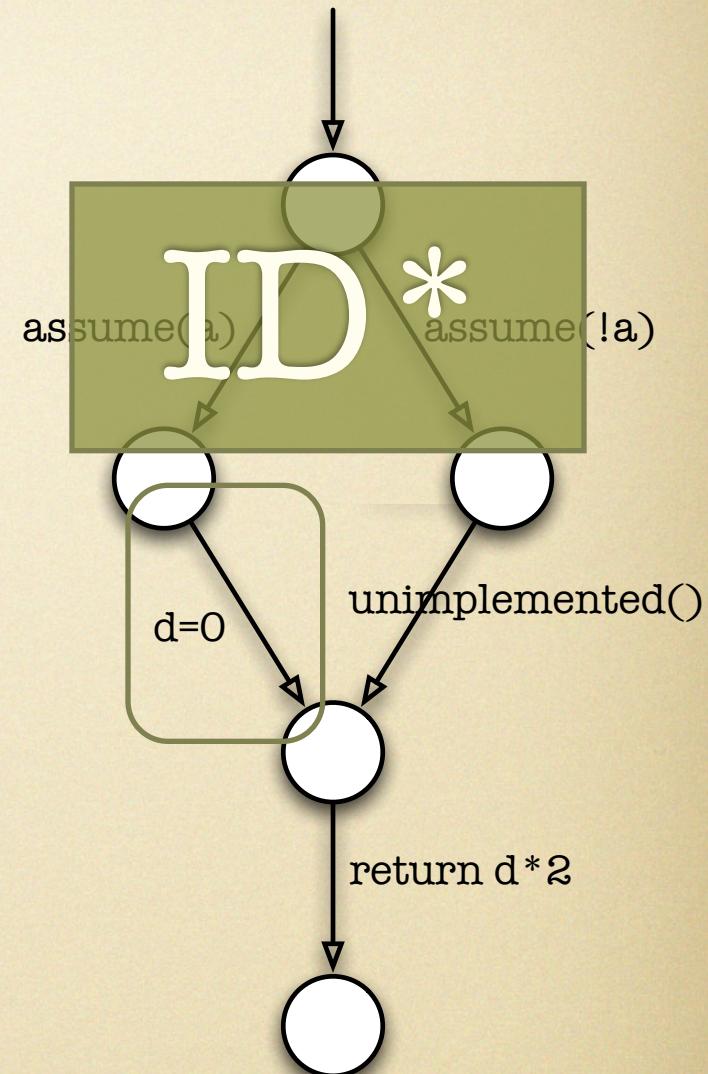
- Test case is execution (program path)
- Intuitively: “cover line 4” =
  - Execute some statement(s)
  - Reach line 4



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

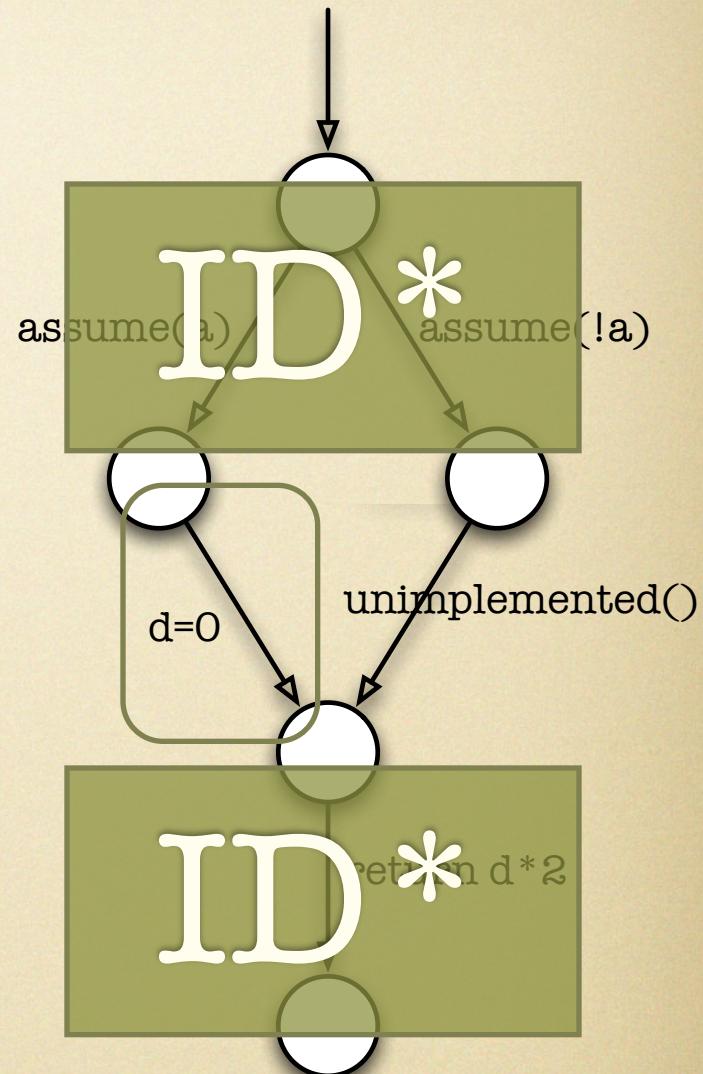
- Test case is execution (program path)
- Intuitively: “cover line 4” =
  - Execute some statement(s)
  - Reach line 4



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

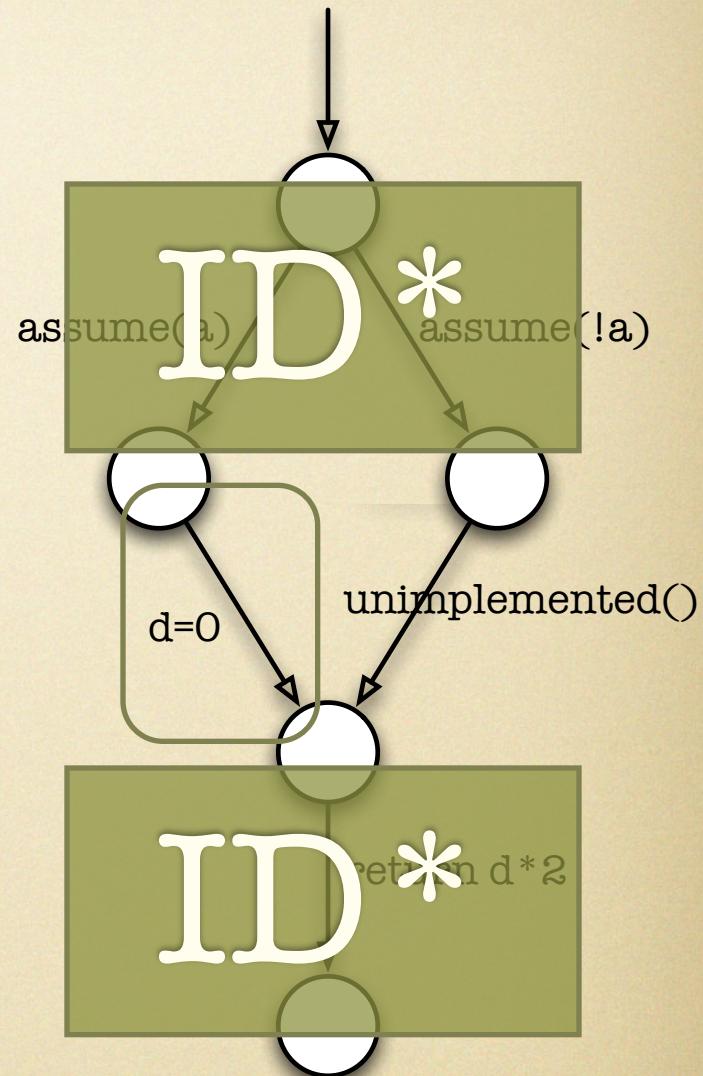
- Test case is execution (program path)
- Intuitively: “cover line 4” =
  - Execute some statement(s)
  - Reach line 4
  - More statements until program exit



# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Test case is execution (program path)
- Intuitively: “cover line 4” =
  - Execute some statement(s)
  - Reach line 4
  - More statements until program exit
  - **cover “ID\*.@4.ID\*”**



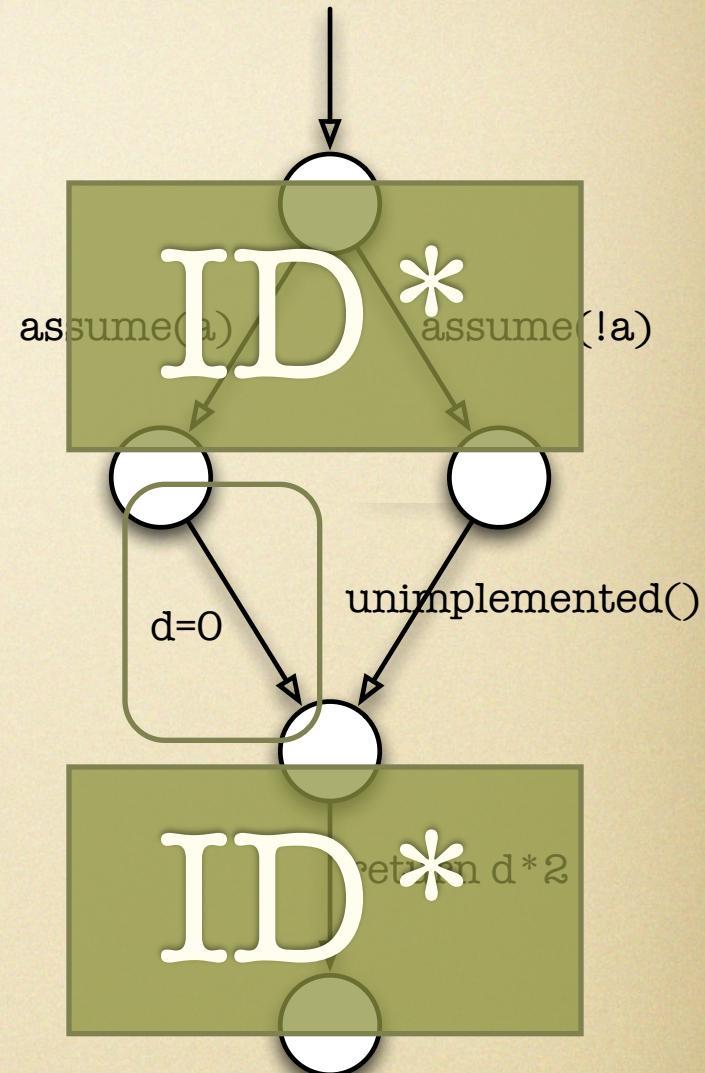
# Describing Single Test Cases

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Test case is execution (program path)
- Intuitively: “cover line 4” =
  - Execute some statement(s)
  - Reach line 4
  - More statements until program exit
  - **cover “ID\*.@4.ID\*”**

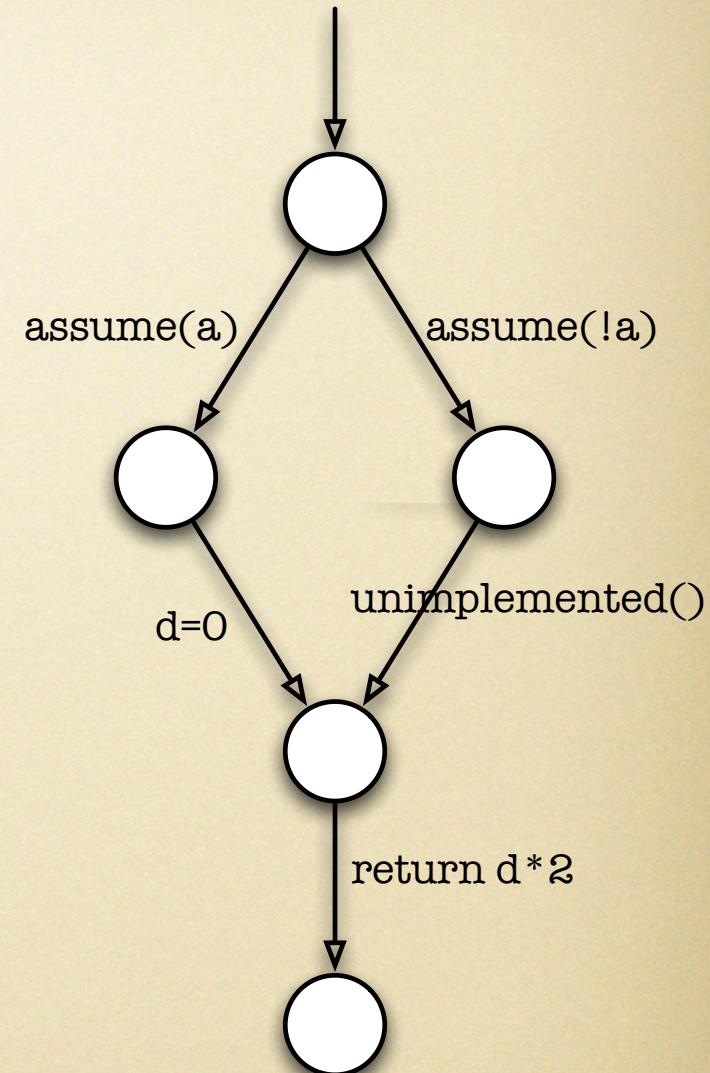
IN:

```
a=2
d=0
```



# Path Patterns

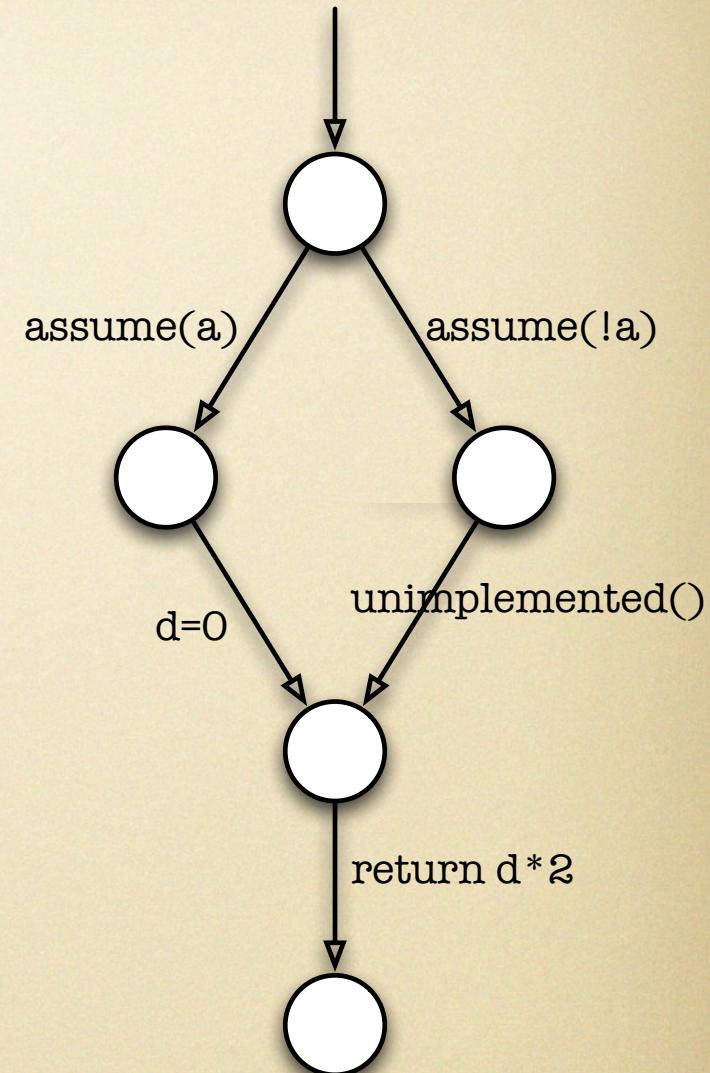
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Path Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

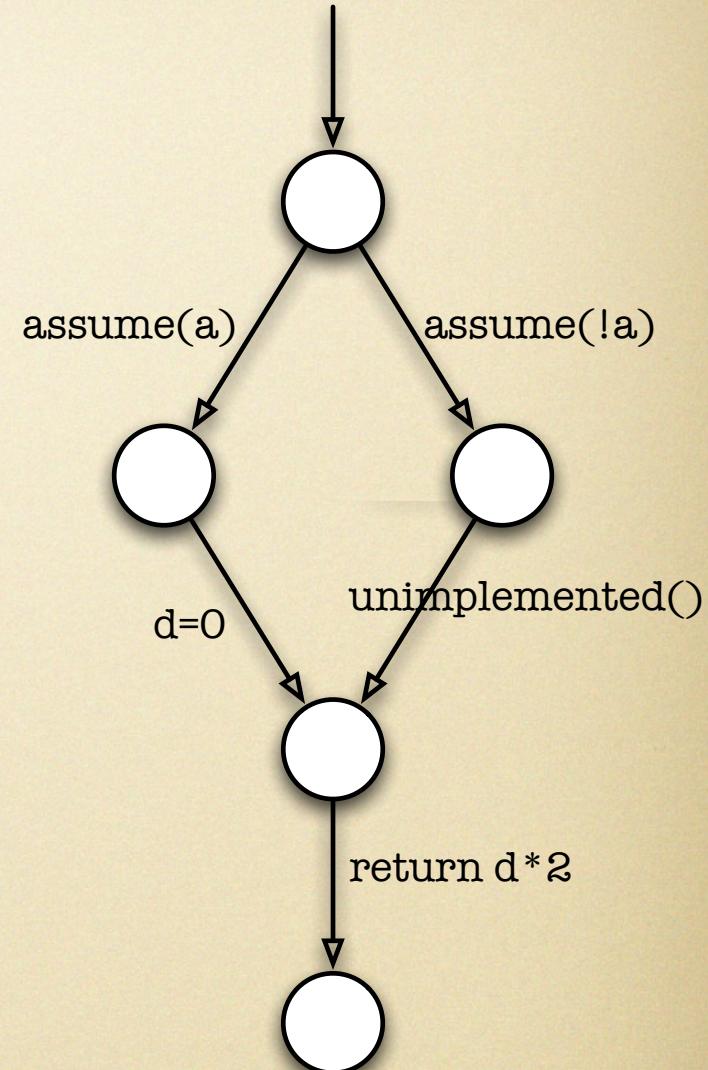
- **cover "ID\* . @4 . ID\* + ID\* . @6 . ID\*"**



# Path Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

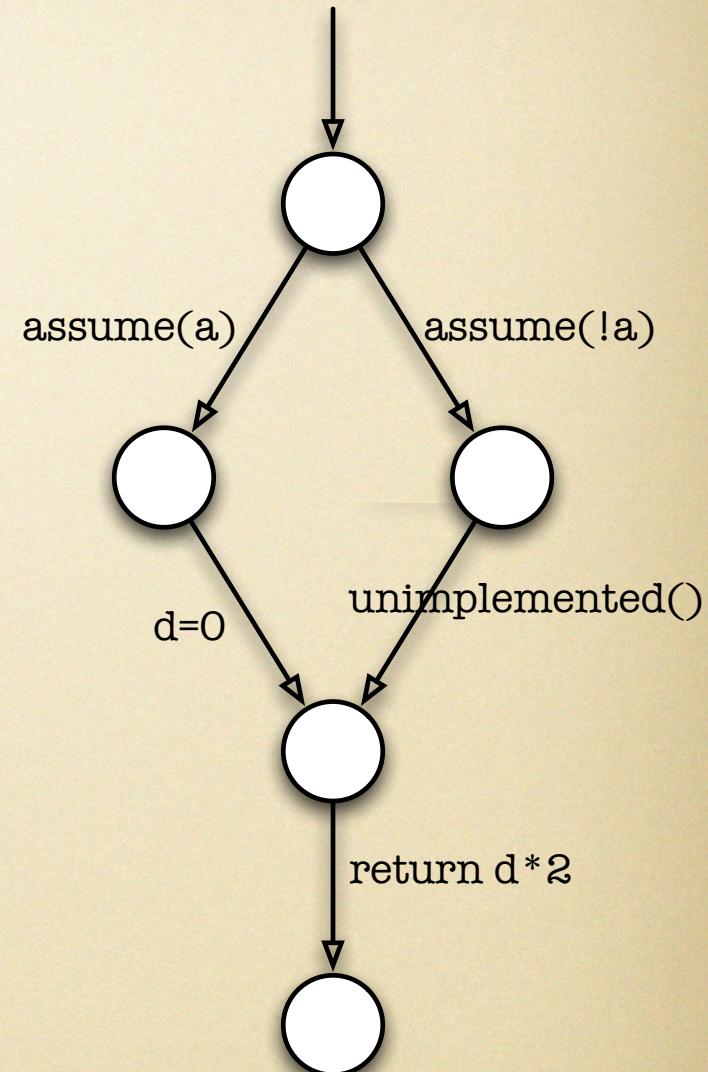
- **cover "ID\* . @4 . ID\* + ID\* . @6 . ID\*"**
- Path patterns are regular expressions:



# Path Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

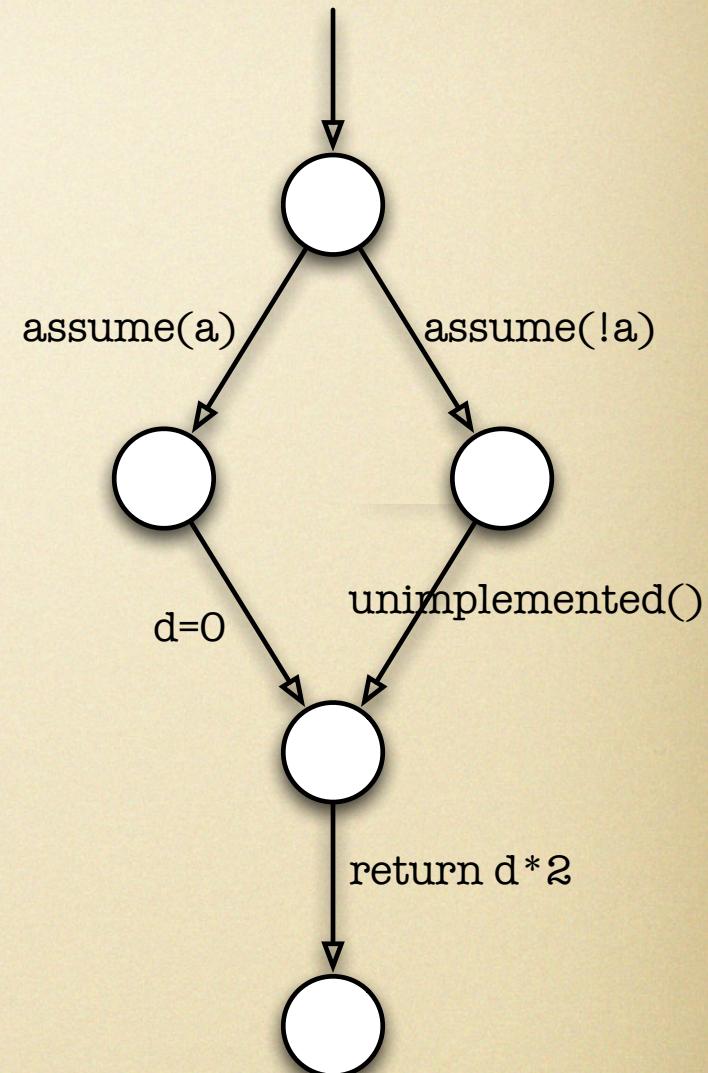
- **cover "ID\* . @4 . ID\* + ID\* . @6 . ID\*"**
- Path patterns are regular expressions:
  - Kleene star (\*)



# Path Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

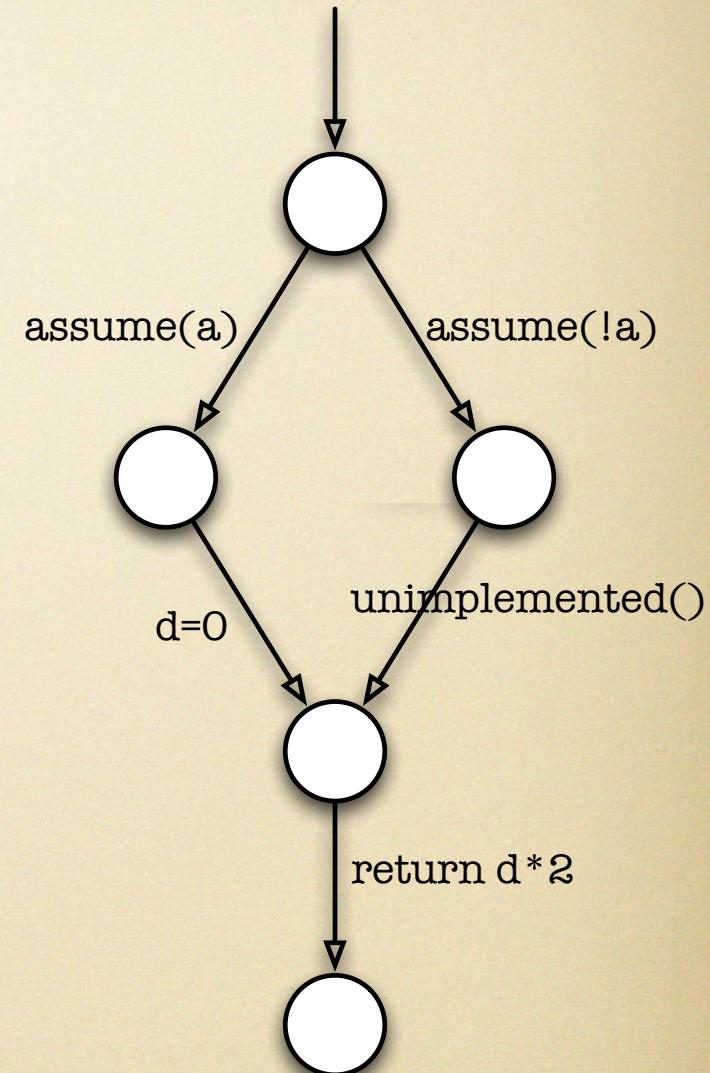
- **cover "ID\* . @4 . ID\* + ID\* . @6 . ID\*"**
- Path patterns are regular expressions:
  - Kleene star (\*)
  - Concatenation (.)



# Path Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- **cover “ID\* . @4 . ID\* + ID\* . @6 . ID\*”**
- Path patterns are regular expressions:
  - Kleene star (\*)
  - Concatenation (.)
  - Alternative/Union (+)



# Alphabet of Path Patterns

# Alphabet of Path Patterns

- Further filter functions

# Alphabet of Path Patterns

- Further filter functions
- CFA transformers

# Alphabet of Path Patterns

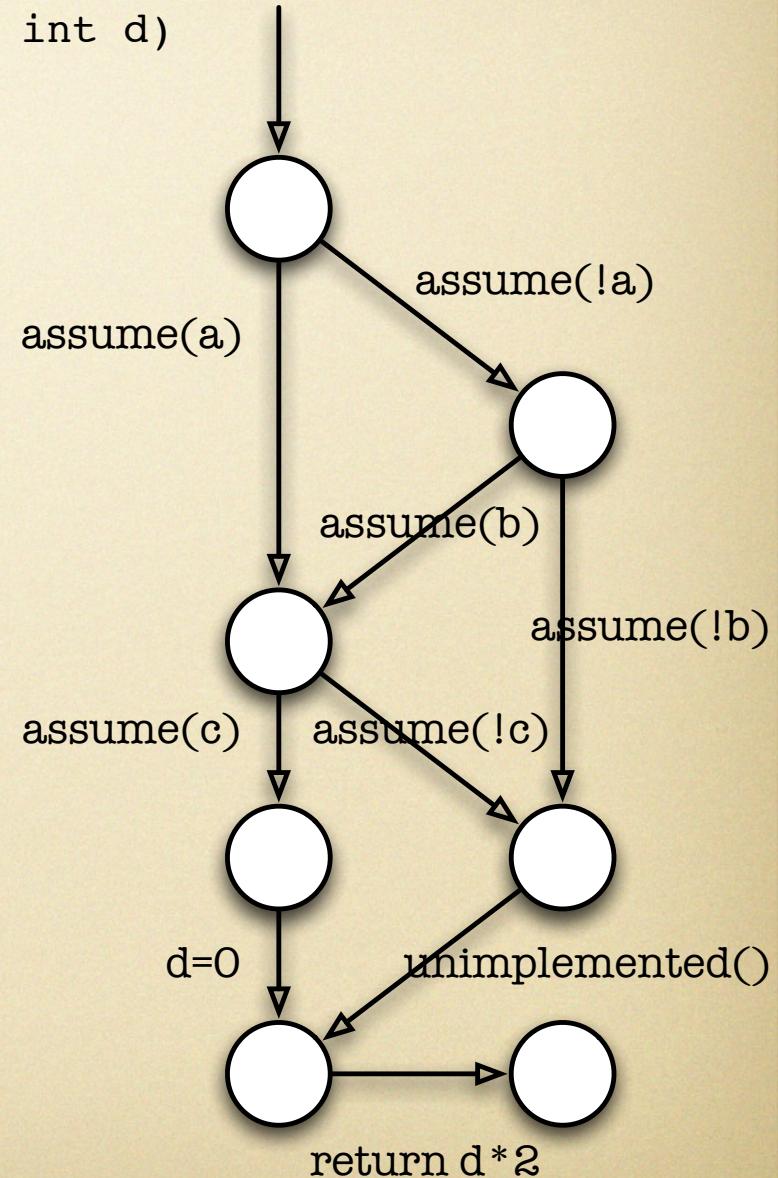
- Further filter functions
- CFA transformers
- Predicates over program variables

# Alphabet of Path Patterns

- Further filter functions
- CFA transformers
- Predicates over program variables
- Beyond CFA edges (statements)

# Further Filter Functions

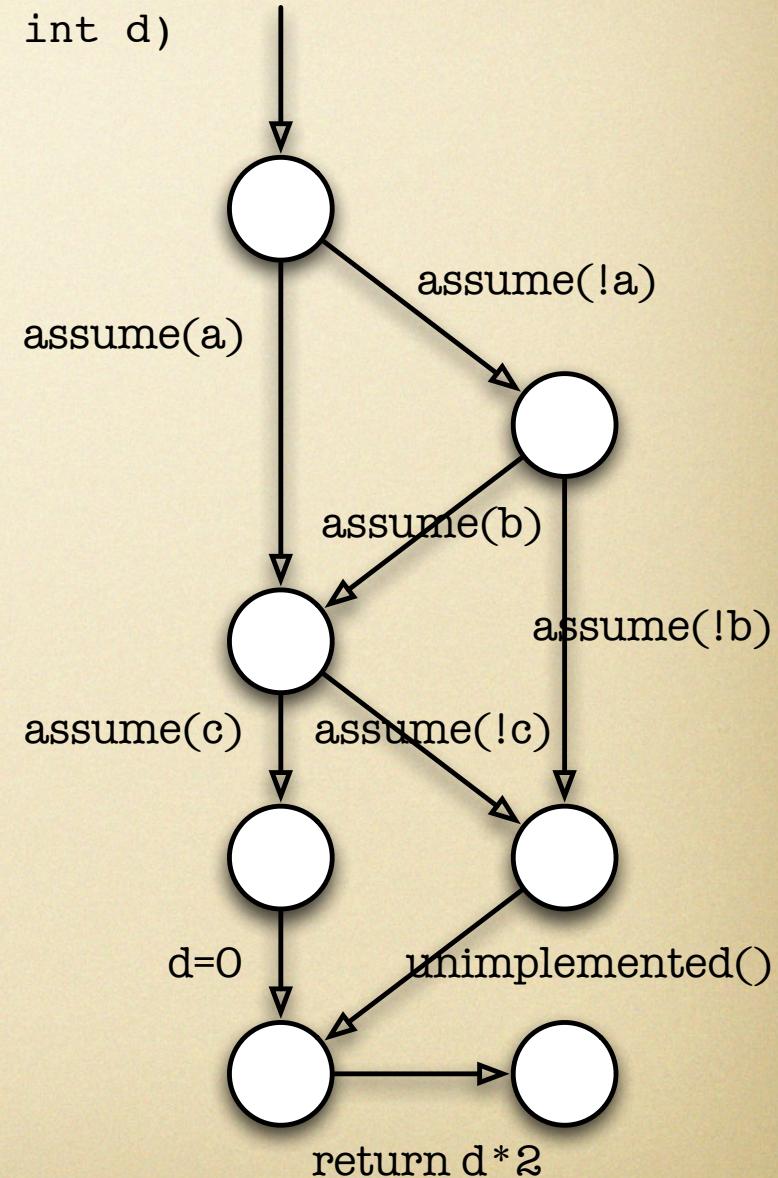
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Further Filter Functions

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

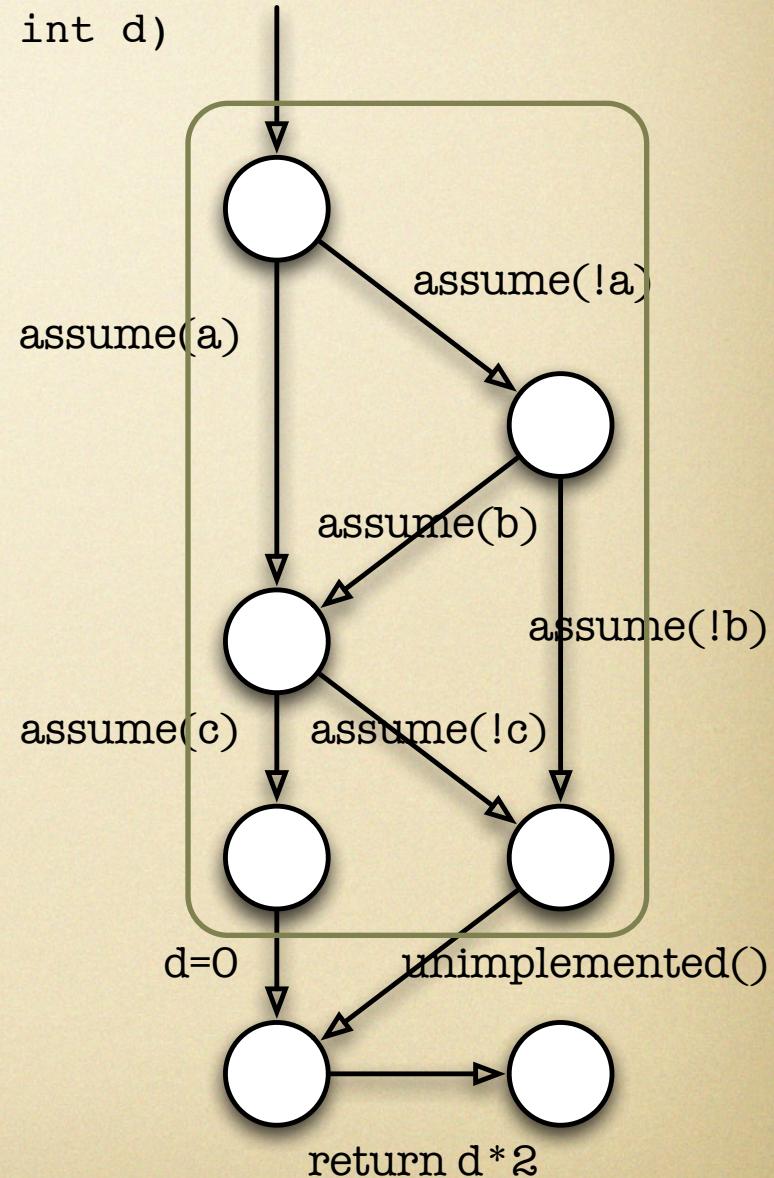
- cover "@conditionedge"



# Further Filter Functions

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

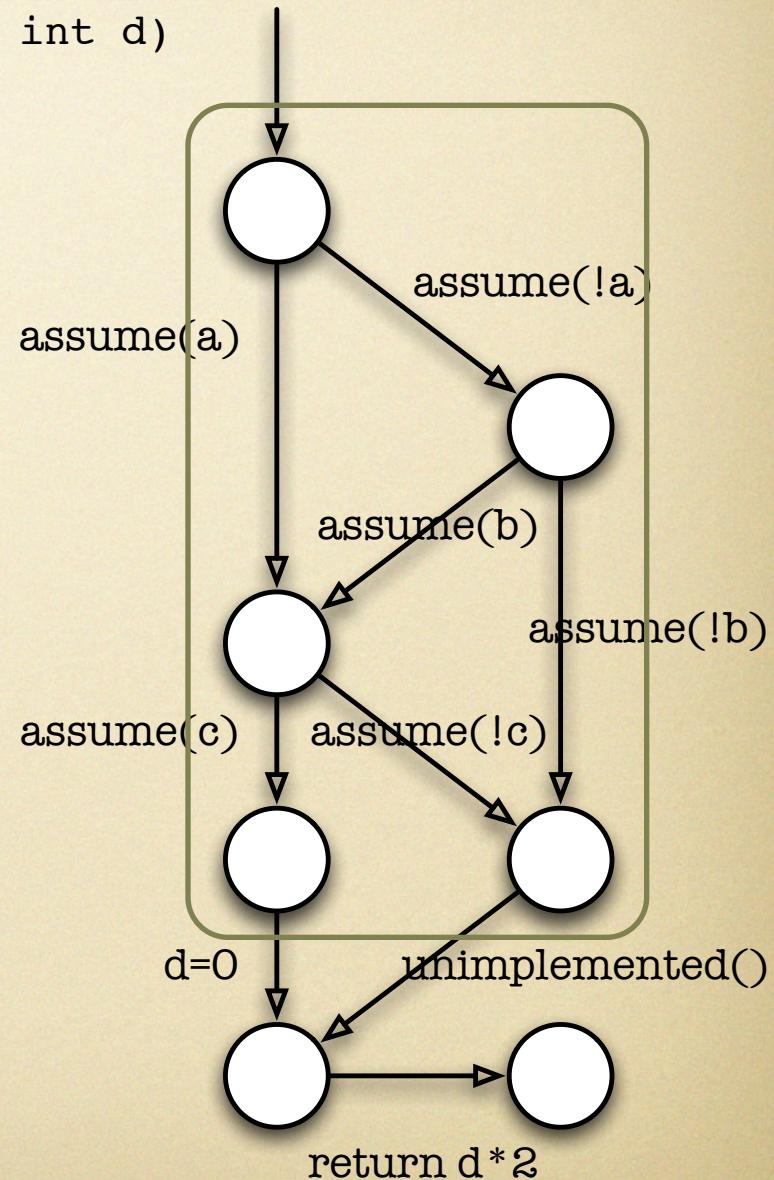
- cover "@conditionedge"



# Further Filter Functions

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

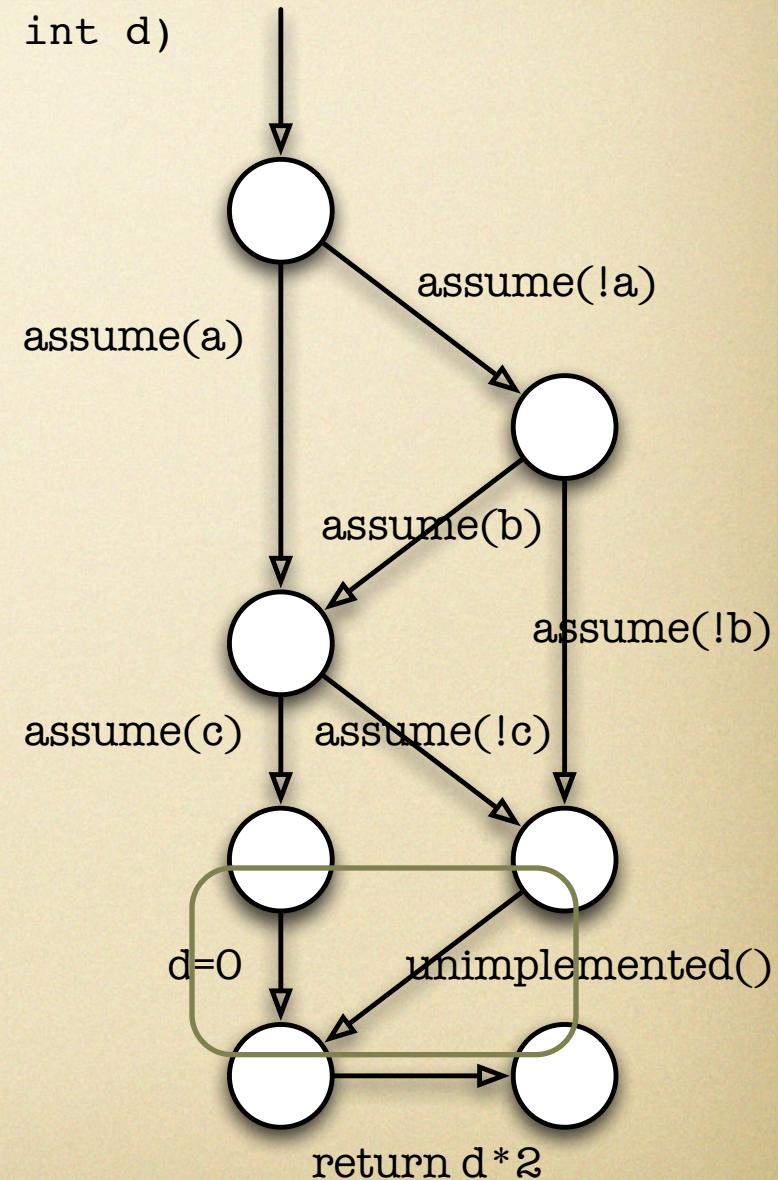
- cover "@conditionedge"
- cover "@conditiongraph"



# Further Filter Functions

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

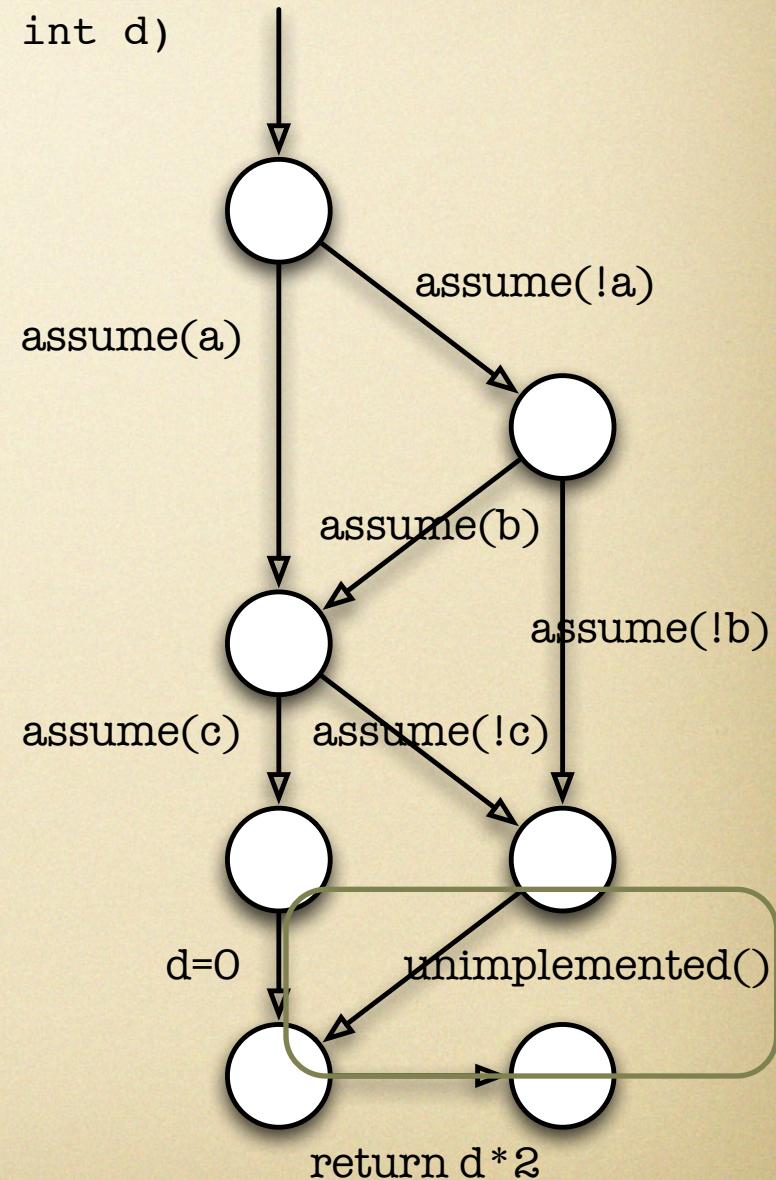
- **cover "@conditionedge"**
- **cover "@conditiongraph"**
- **cover "@decisionedge"**



# Further Filter Functions

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

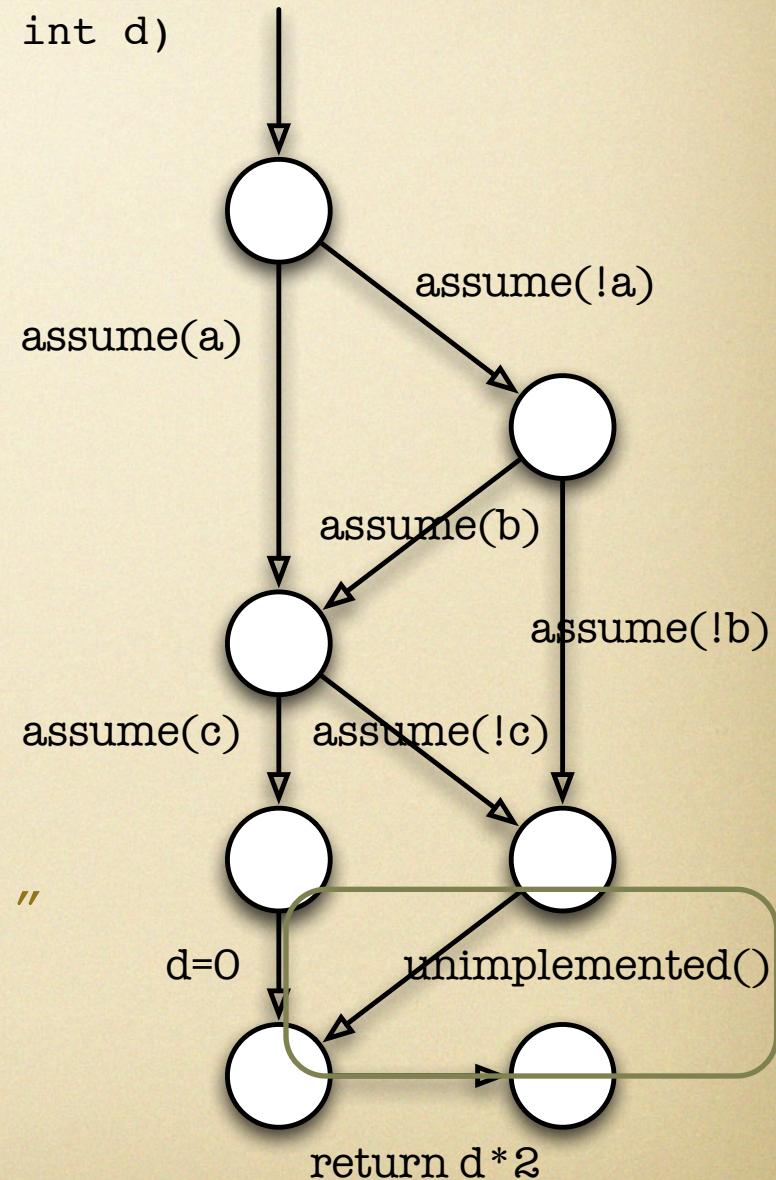
- **cover "@conditionedge"**
- **cover "@conditiongraph"**
- **cover "@decisionedge"**
- **cover "@calls"**



# Further Filter Functions

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

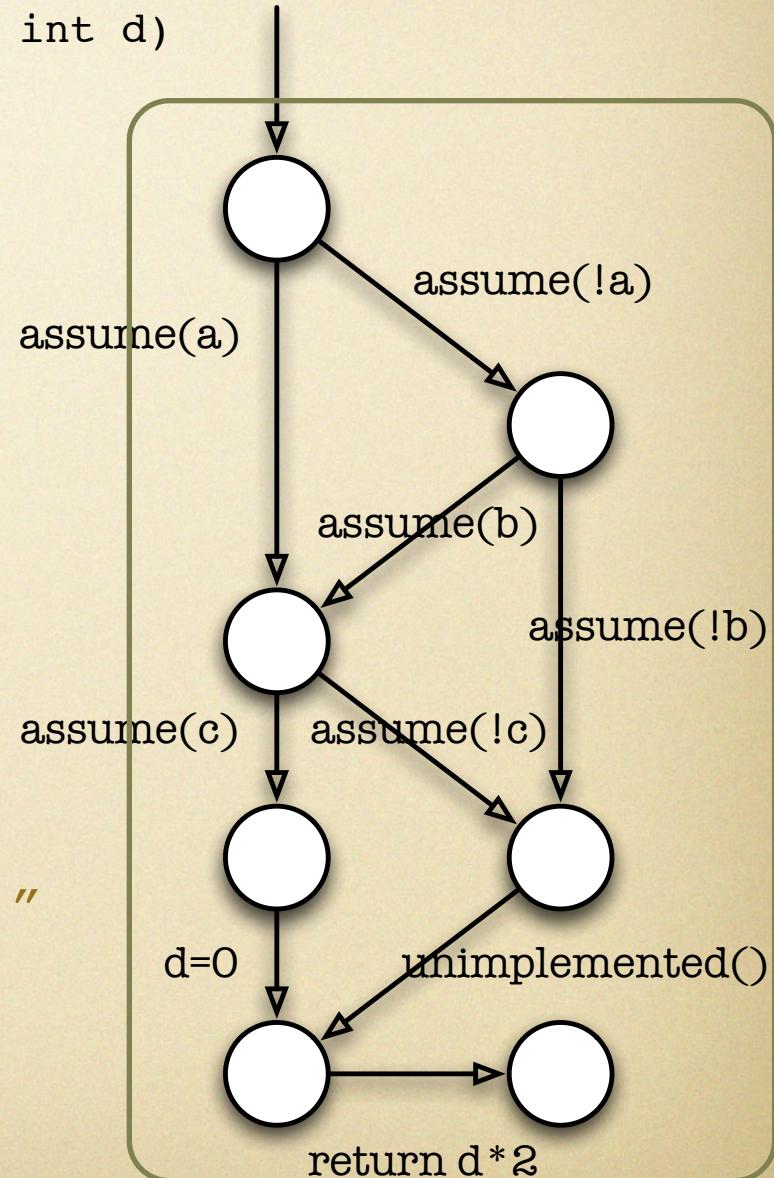
- **cover "@conditionedge"**
- **cover "@conditiongraph"**
- **cover "@decisionedge"**
- **cover "@calls"**
- **cover "@call(unimplemented)"**



# Further Filter Functions

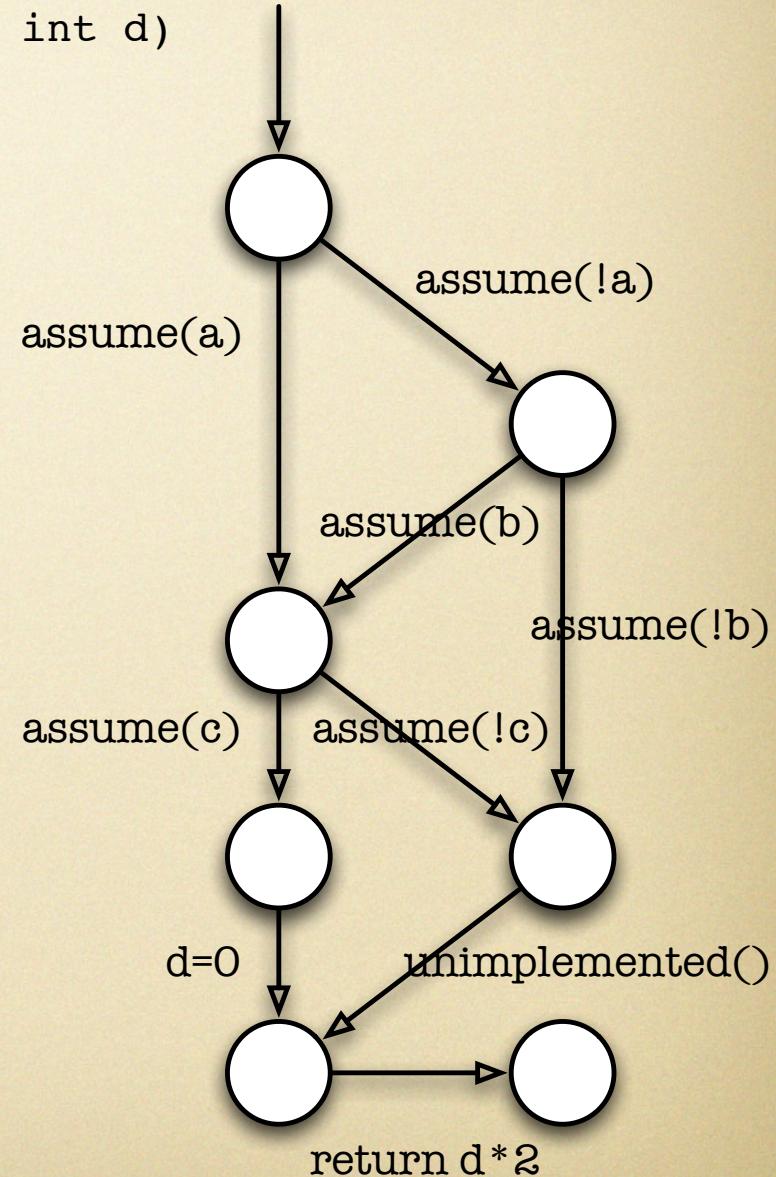
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- **cover "@conditionedge"**
- **cover "@conditiongraph"**
- **cover "@decisionedge"**
- **cover "@calls"**
- **cover "@call(unimplemented)"**
- **cover "@func(example)"**



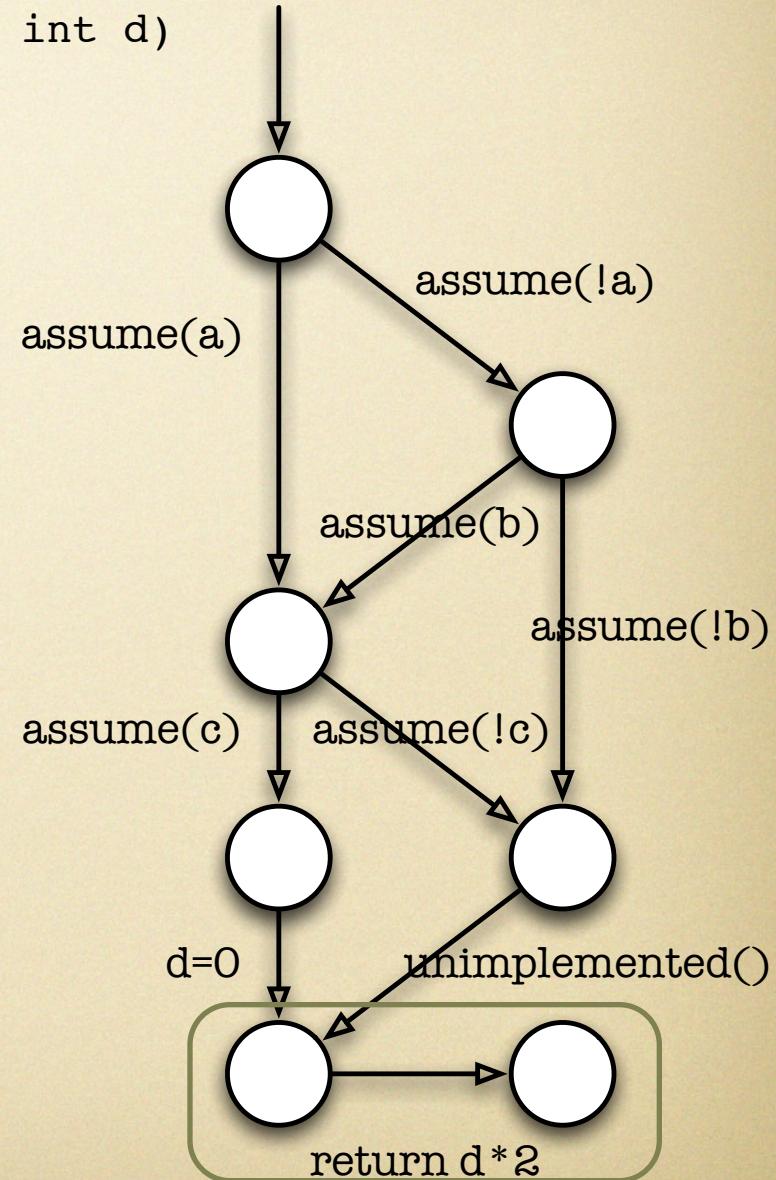
# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



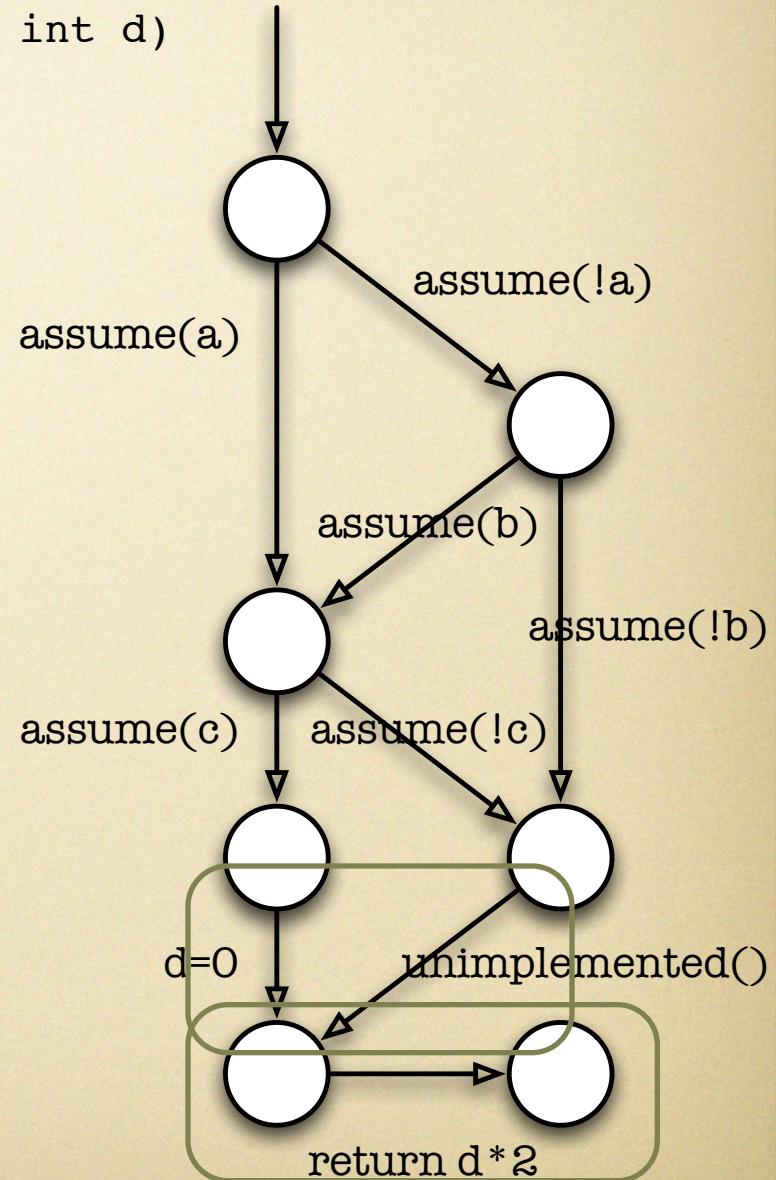
# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# CFA Transformers

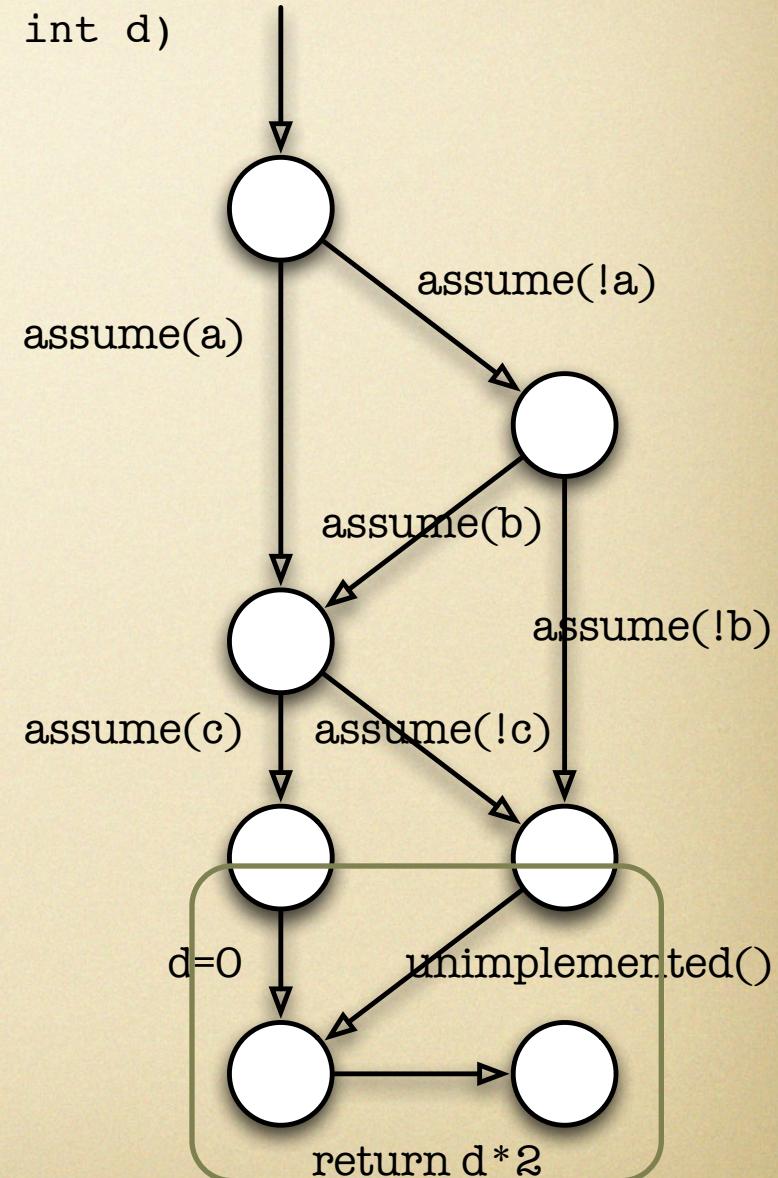
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

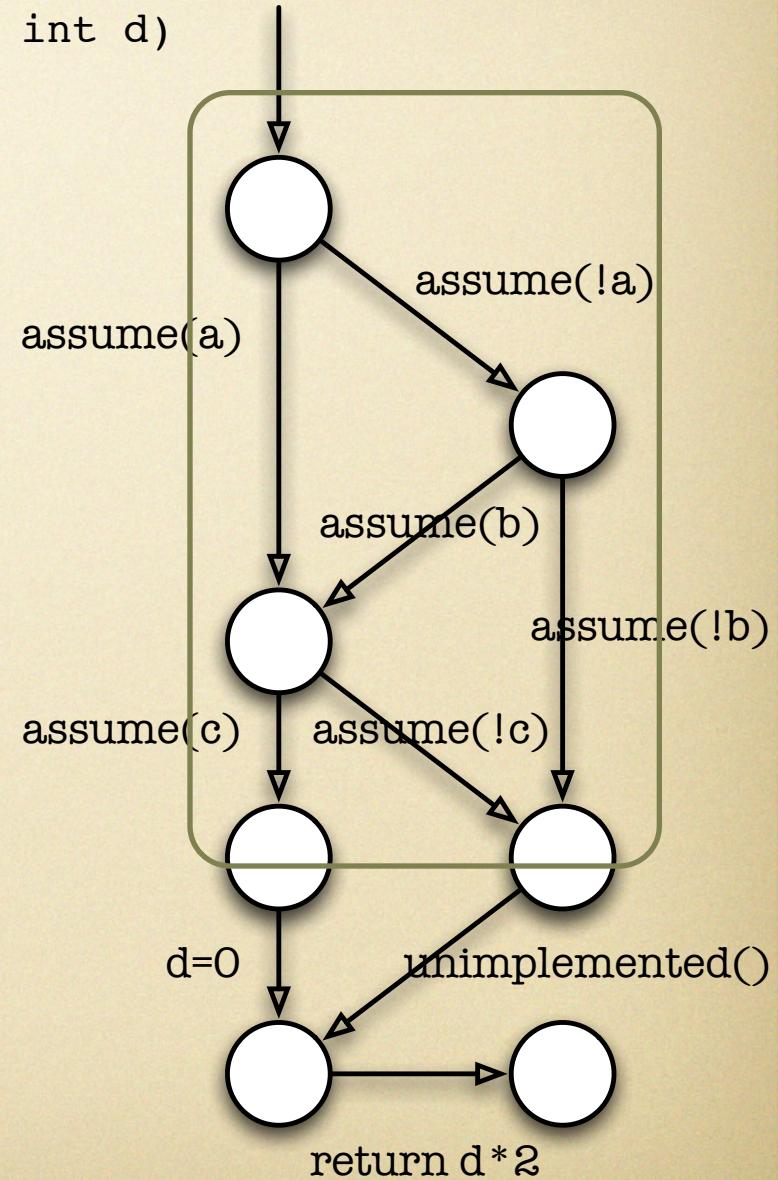
- cover "@7 | @decisionedge"



# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

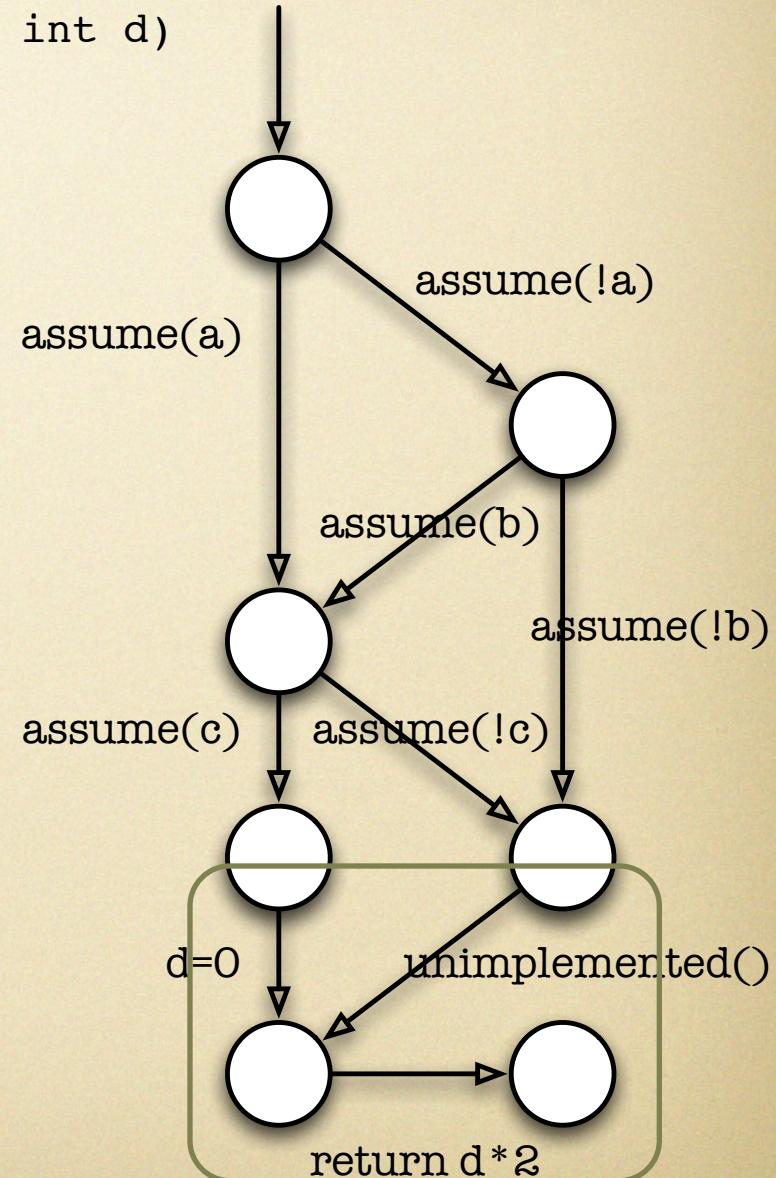
- cover "@7 | @decisionedge"



# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

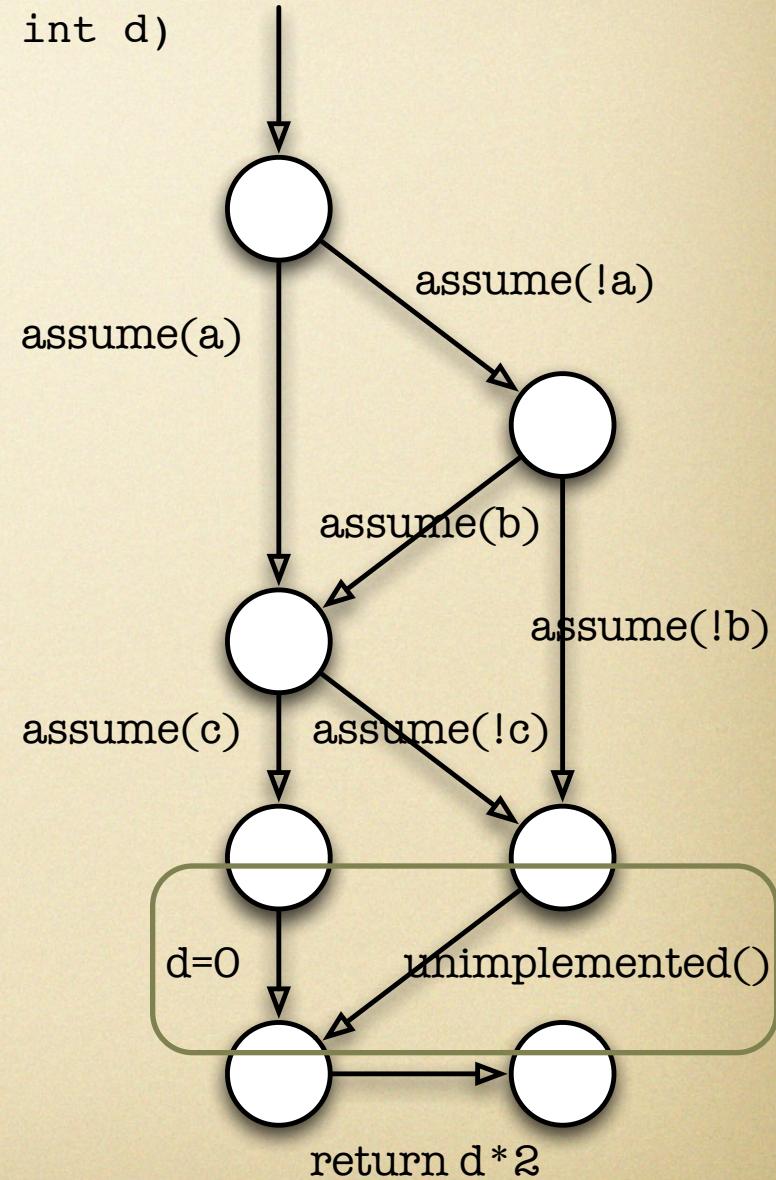
- cover "@7 | @decisionedge"
- cover "not(@3)"



# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

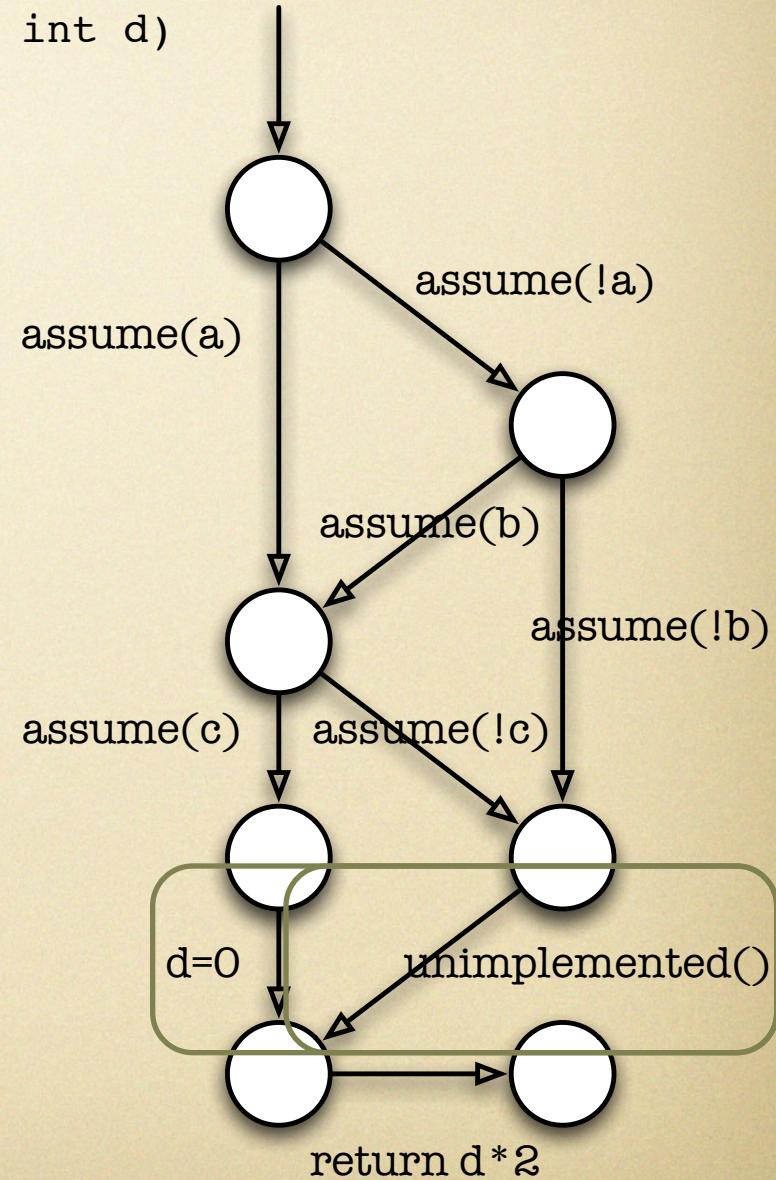
- cover "@7 | @decisionedge"
- cover "not(@3)"



# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

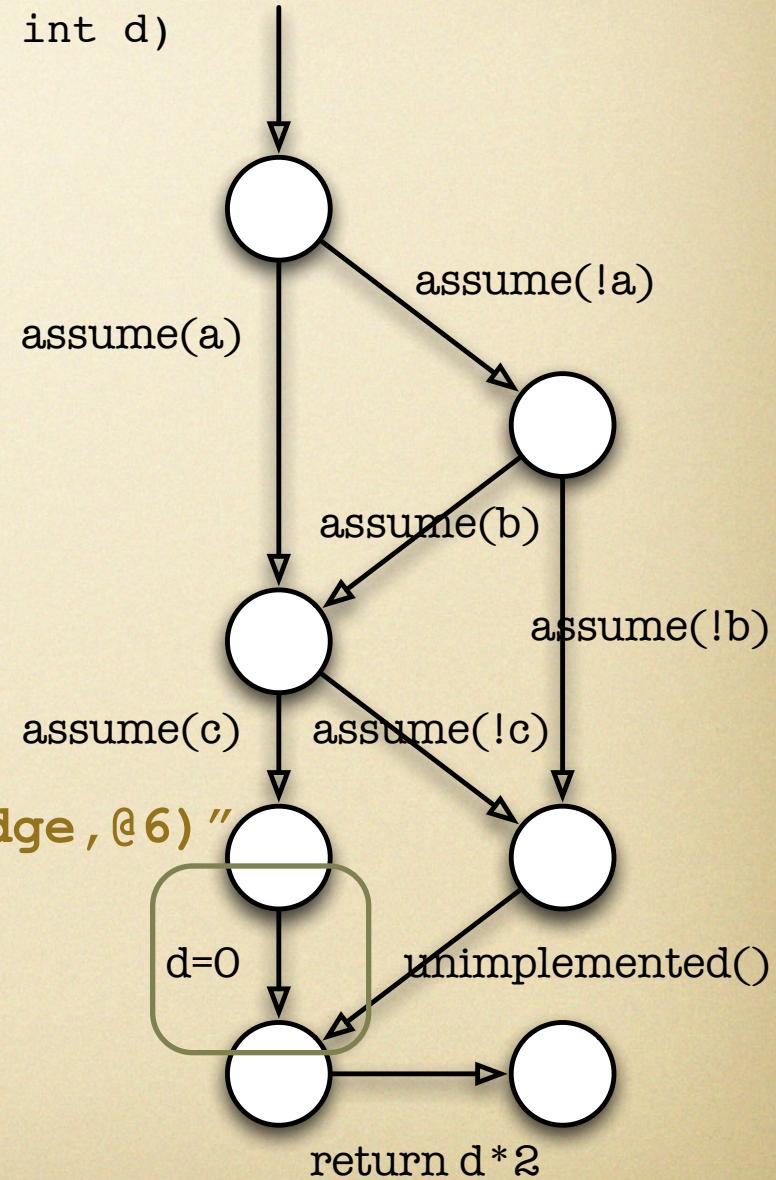
- cover "@7 | @decisionedge"
- cover "not(@3)"



# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

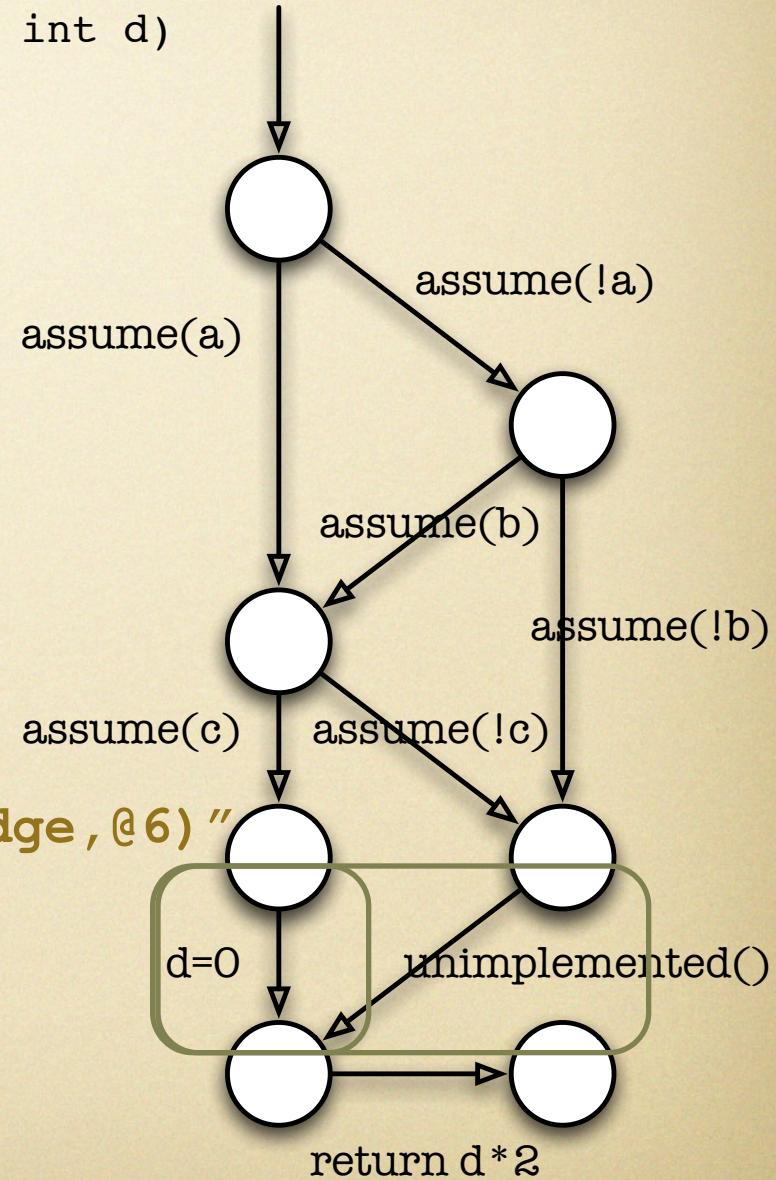
- cover "@7 | @decisionedge"
- cover "not(@3)"
- cover "setminus(@decisionedge, @6)"



# CFA Transformers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

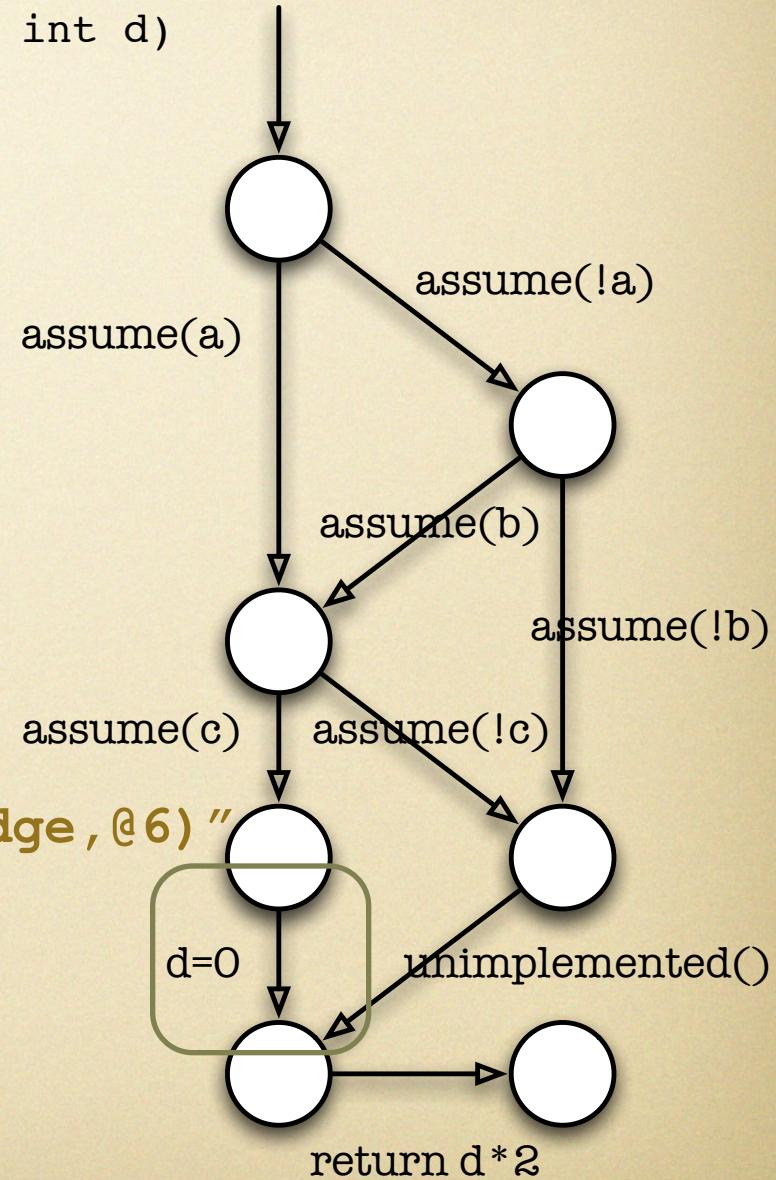
- cover "@7 | @decisionedge"
- cover "not(@3)"
- cover "setminus (@decisionedge, @6)"



# CFA Transformers

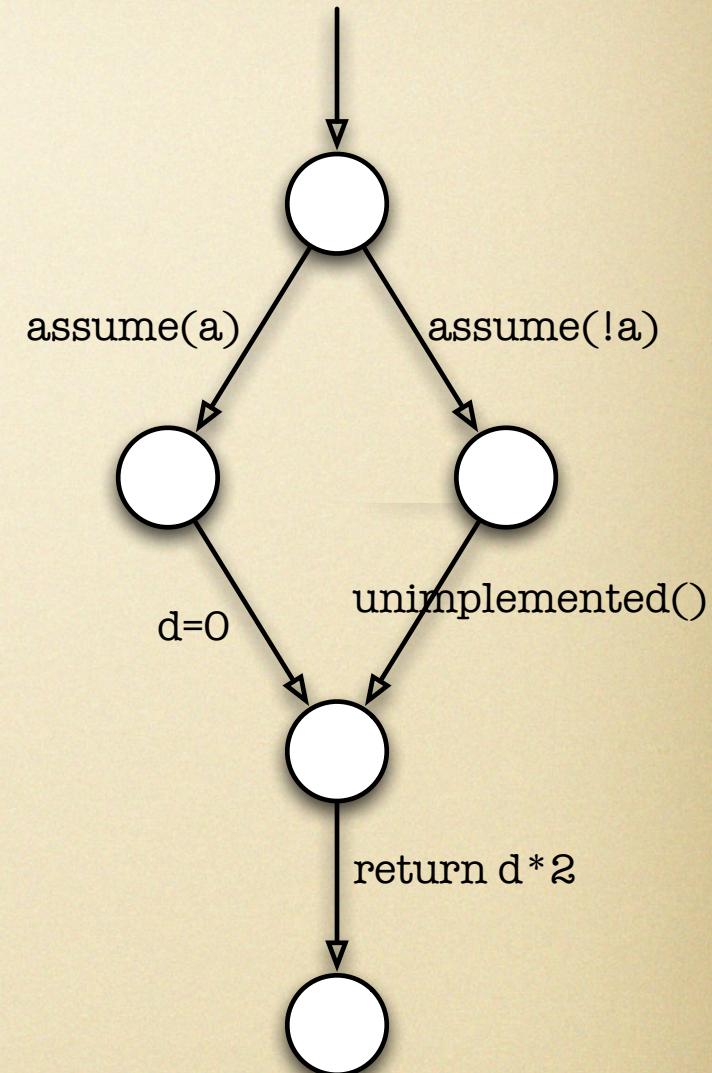
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- cover "@7 | @decisionedge"
- cover "not(@3)"
- cover "setminus(@decisionedge, @6)"
- cover "@decisionedge & @4"



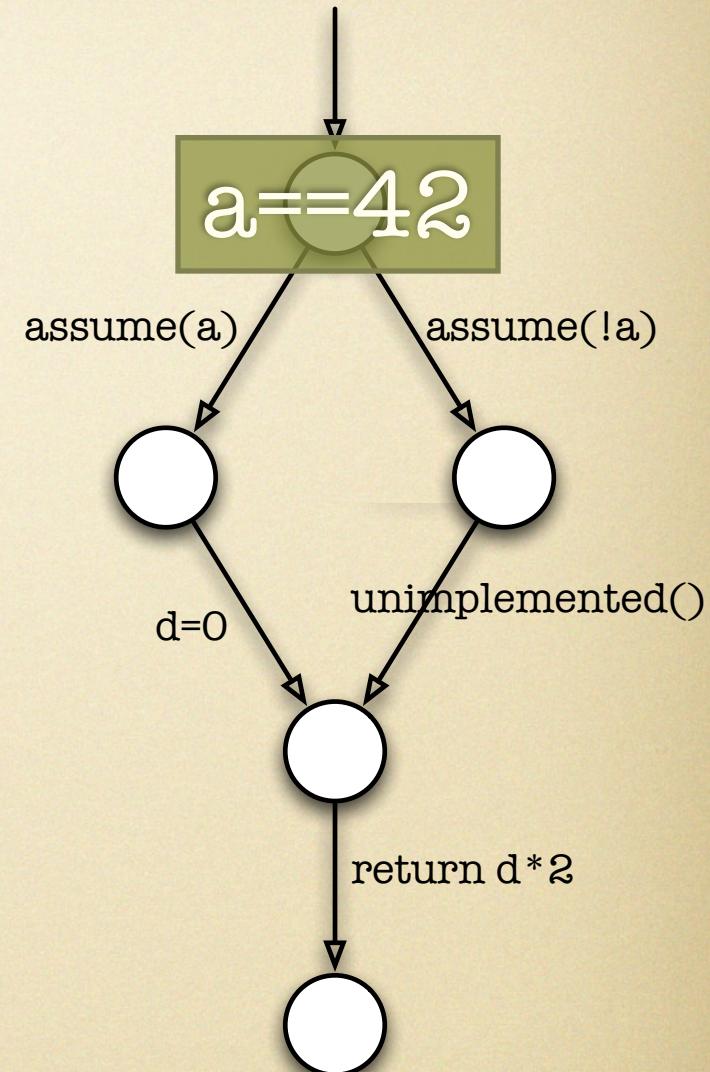
# Program Variables

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Program Variables

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



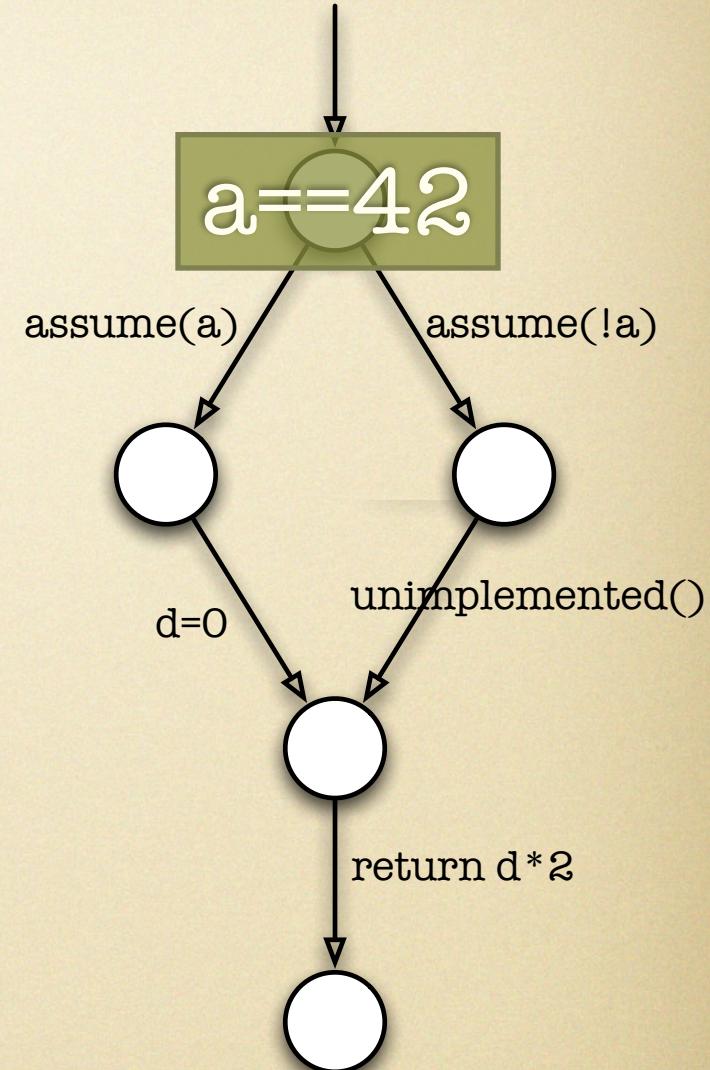
# Program Variables

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- cover “{ $a==42$ } .@3”

IN:

$a=42$   
 $d=0$



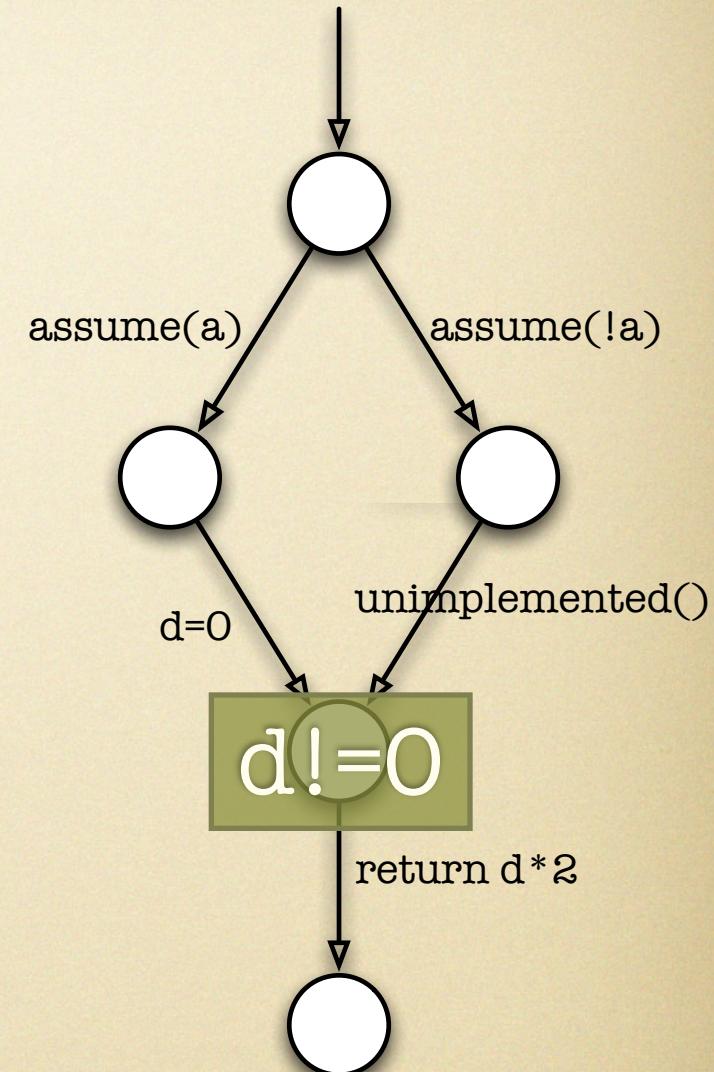
# Program Variables

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- **cover “{a==42} .@3”**

IN:

a=42  
d=0



# Program Variables

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- **cover “{a==42} .@3”**

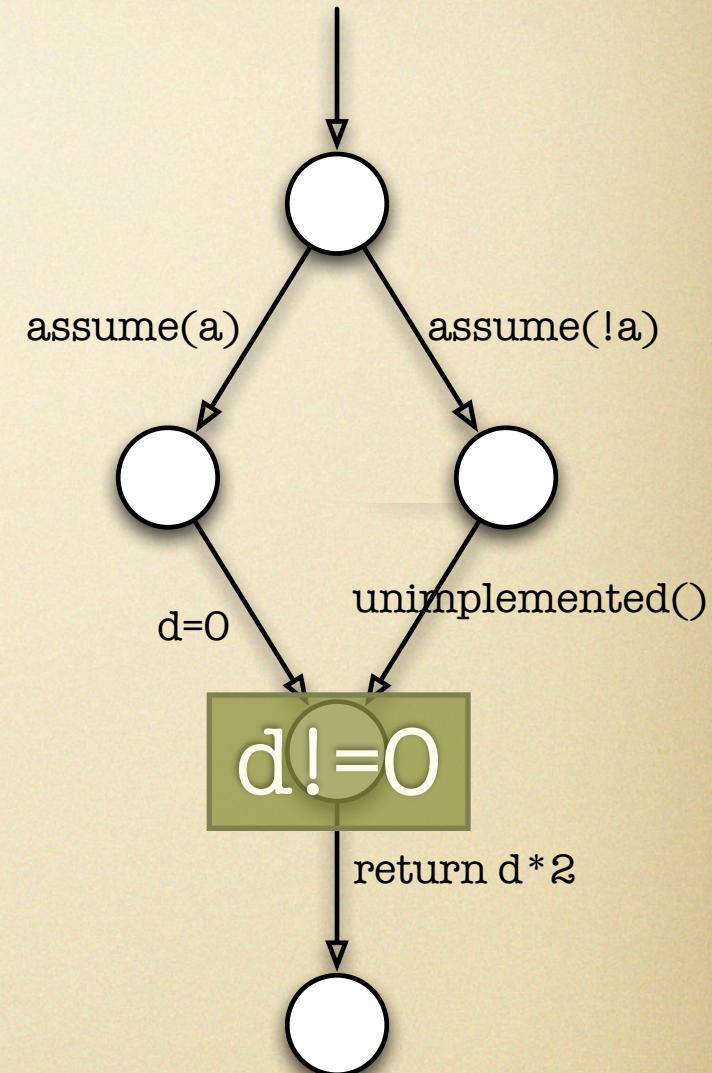
IN:

a=42  
d=0

- **cover “(@4+@6) . {d!=0}”**

IN:

a=0  
d=65536

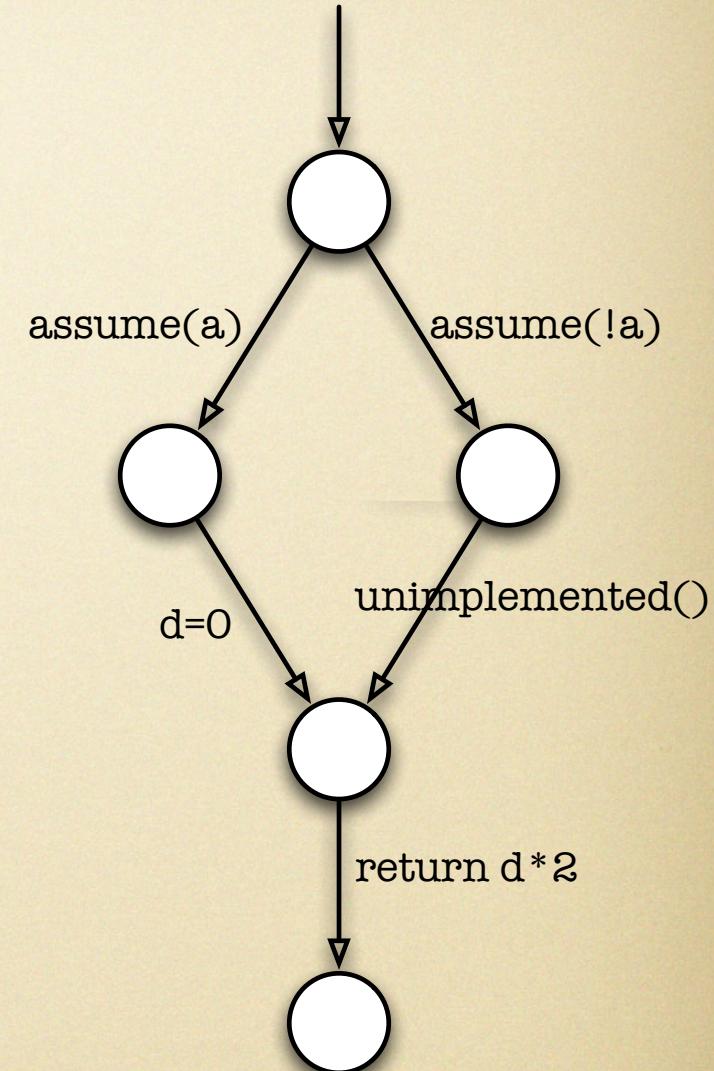


# Software Testing using FQL

- ☐ Program Representation: Control Flow Automata
- ☐ Describing Single Test Cases
  - Describing Test Suites

# Coverage Patterns

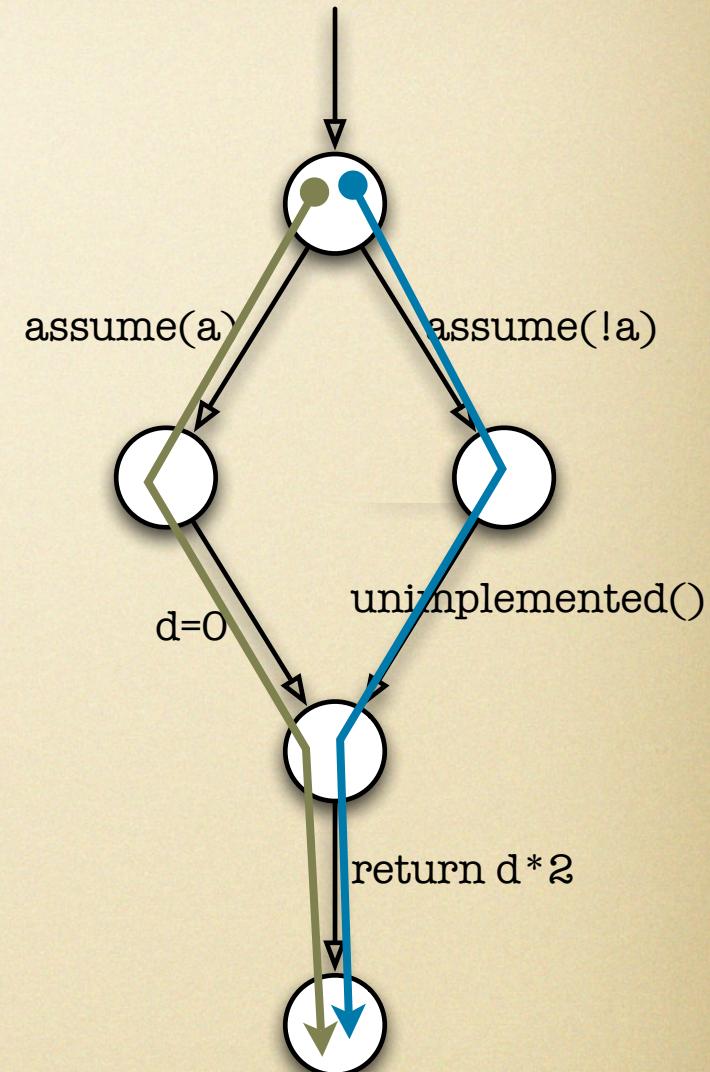
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Coverage Patterns

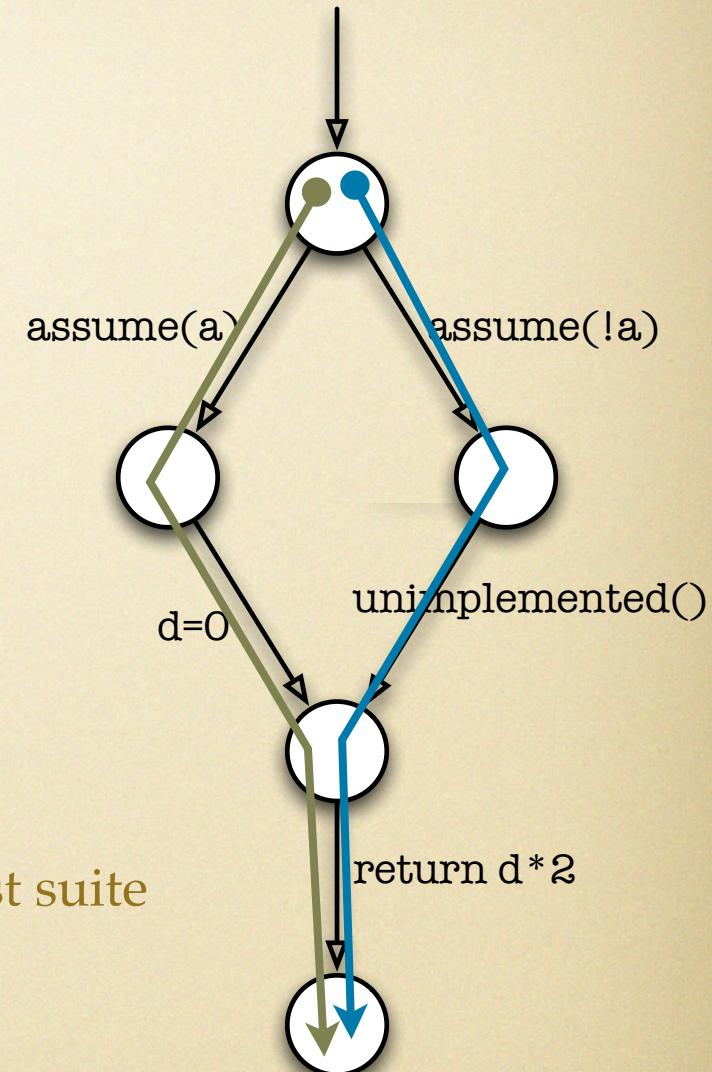
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Cover if and else branch ?



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



- Cover if **and** else branch ?
- We need more than one test case - a test suite

# Coverage Patterns

# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**

# Coverage Patterns

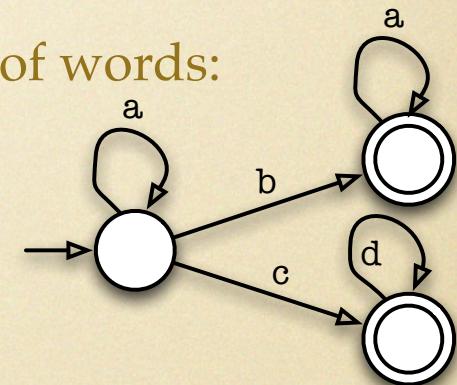
- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals

# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals
- Path patterns may describe **infinite** number of words:
  - $a^* . b . a^* + cd^* = \{ b, c, ab, cd, aba, cdd, aab, aaba, aabaa, \dots \}$

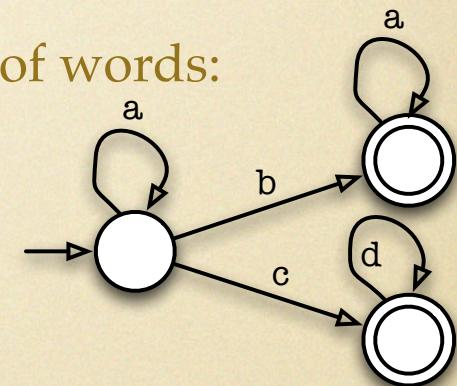
# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals
- Path patterns may describe **infinite** number of words:
  - $a^* . b . a^* + cd^* = \{ b, c, ab, cd, aba, cdd, aab, aaba, aabaa, \dots \}$



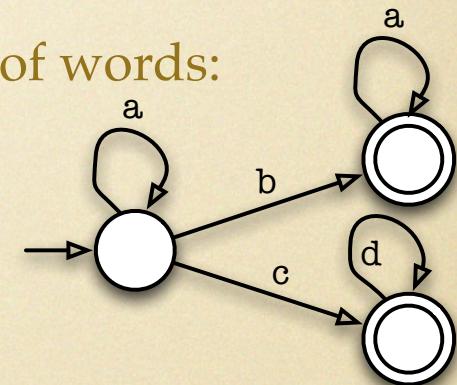
# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals
- Path patterns may describe **infinite** number of words:
  - $a^* . b . a^* + cd^* = \{ b, c, ab, cd, aba, cdd, aab, aaba, aabaa, \dots \}$
- Quoting operator **blocks** expansion:

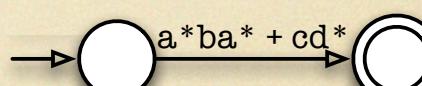


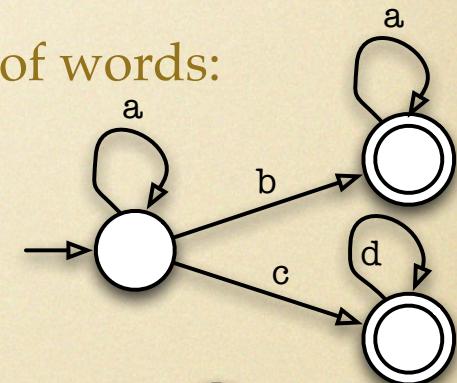
# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals
- Path patterns may describe **infinite** number of words:
  - $a^* . b . a^* + cd^* = \{ b, c, ab, cd, aba, cdd, aab, aaba, aabaa, \dots \}$
- Quoting operator **blocks expansion**:
  - “ $a^* . b . a^* + cd^*$ ” = {  $a^*ba^* + cd^*$  }



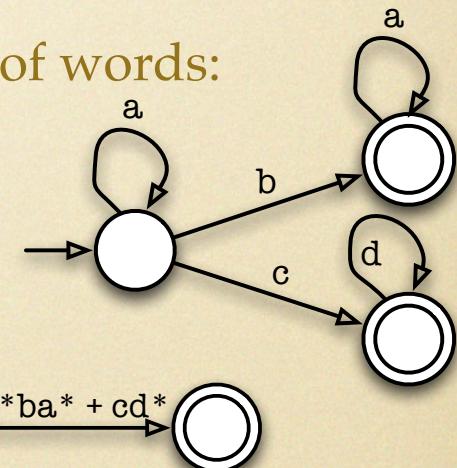
# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals
- Path patterns may describe **infinite** number of words:
  - $a^* . b . a^* + cd^* = \{ b, c, ab, cd, aba, cdd, aab, aaba, aabaa, \dots \}$
- Quoting operator **blocks expansion**:
  - “ $a^* . b . a^* + cd^*$ ” = {  $a^*ba^* + cd^*$  } → 

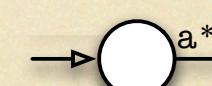


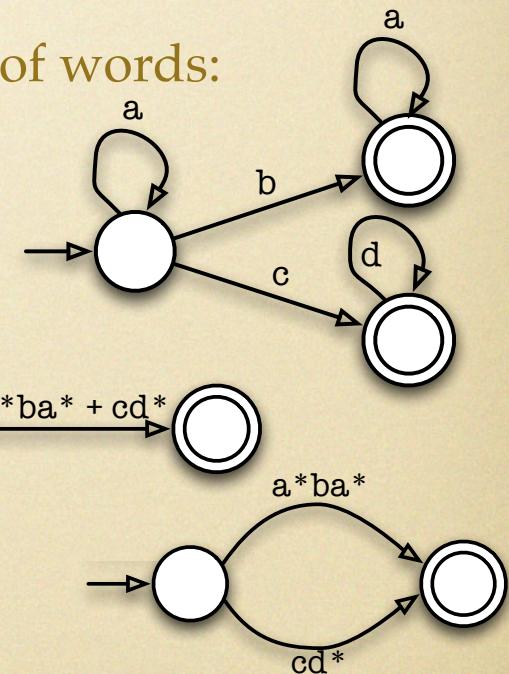
# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals
- Path patterns may describe **infinite** number of words:
  - $a^* . b . a^* + cd^* = \{ b, c, ab, cd, aba, cdd, aab, aaba, aabaa, \dots \}$
- Quoting operator **blocks expansion**:
  - “ $a^* . b . a^*$ ” = {  $a^*ba^*$  } →  $a^*ba^*$
  - “ $a^* . b . a^*$ ” + “ $cd^*$ ” = {  $a^*ba^*$ ,  $cd^*$  }

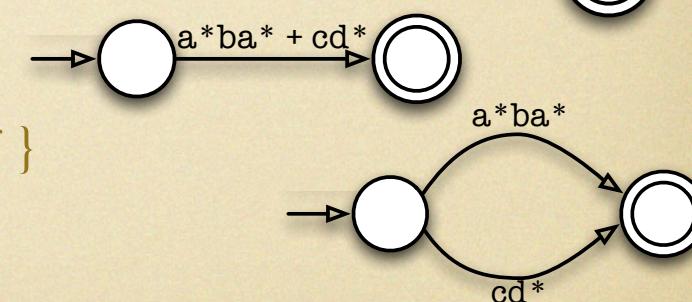


# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals
- Path patterns may describe **infinite** number of words:
  - $a^* . b . a^* + cd^* = \{ b, c, ab, cd, aba, cdd, aab, aaba, aabaa, \dots \}$
- Quoting operator **blocks expansion**:
  - “ $a^* . b . a^* + cd^*$ ” = {  $a^*ba^* + cd^*$  } → A Non-deterministic Finite Automaton (NFA) with two states. The start state has a self-loop labeled 'a'. It also has transitions labeled 'b' to a second state and 'c' to another state. The second state has a self-loop labeled 'd'. The third state has a self-loop labeled 'a'. Both transitions from the start state to the second and third states are labeled  $a^*ba^* + cd^*$ . The final states are the second and third states.
$$a^*ba^* + cd^* = \{ a^*ba^*, cd^* \}$$
  - “ $a^* . b . a^*$ ” + “ $cd^*$ ” = {  $a^*ba^*$ ,  $cd^*$  }

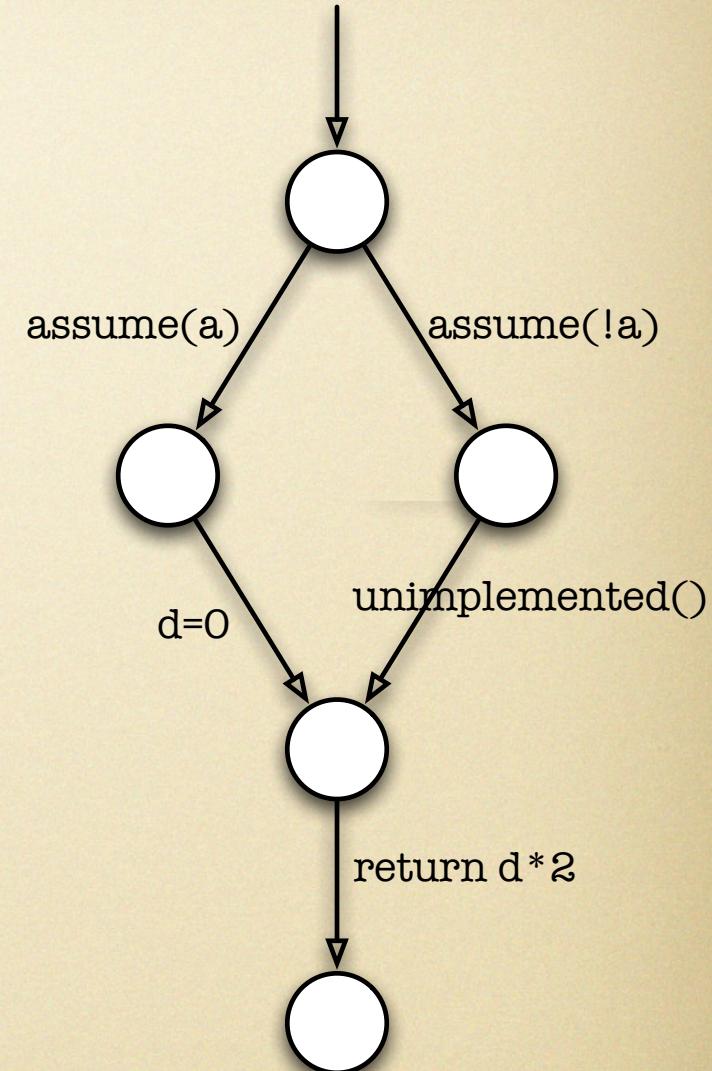


# Coverage Patterns

- Like path patterns, coverage patterns describe a language
- Each word is a path pattern
- This language denotes the set of **test goals**
- We need a **finite** set of test goals
- Path patterns may describe **infinite** number of words:
  - $a^* . b . a^* + cd^* = \{ b, c, ab, cd, aba, cdd, aab, aaba, aabaa, \dots \}$
- Quoting operator **blocks expansion**:
  - “ $a^* . b . a^* + cd^*$ ” = {  $a^*ba^* + cd^*$  } → 
  - “ $a^* . b . a^*$ ” + “ $cd^*$ ” = {  $a^*ba^*$ ,  $cd^*$  }
- In coverage patterns: Kleene star only within quotes

# Coverage Patterns

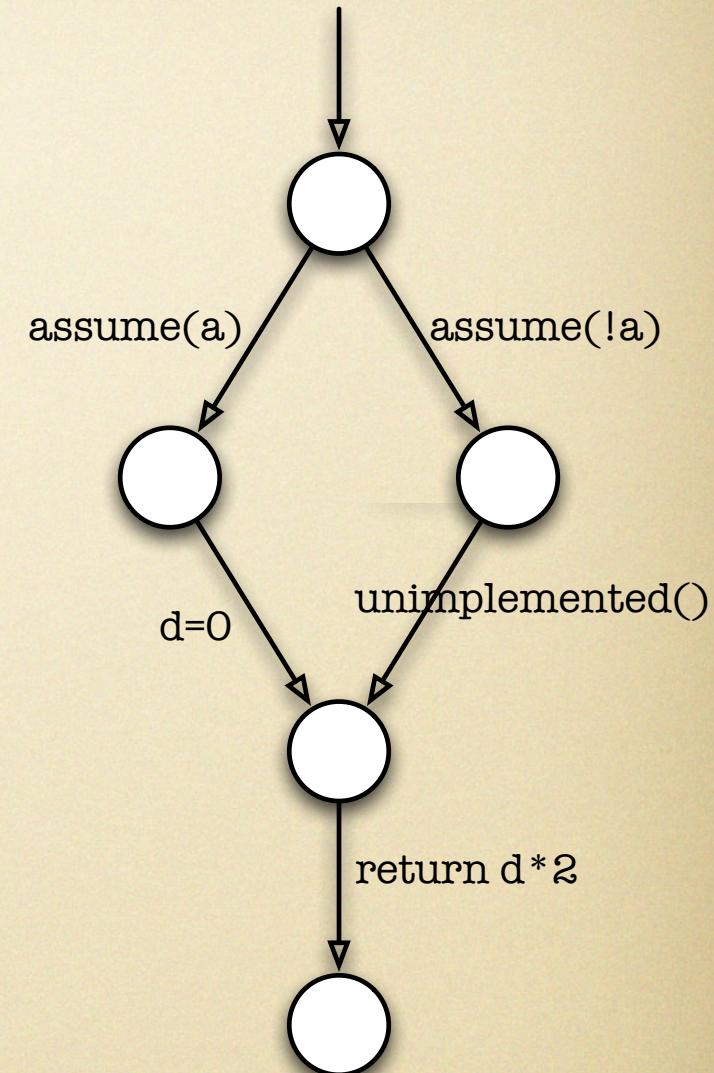
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

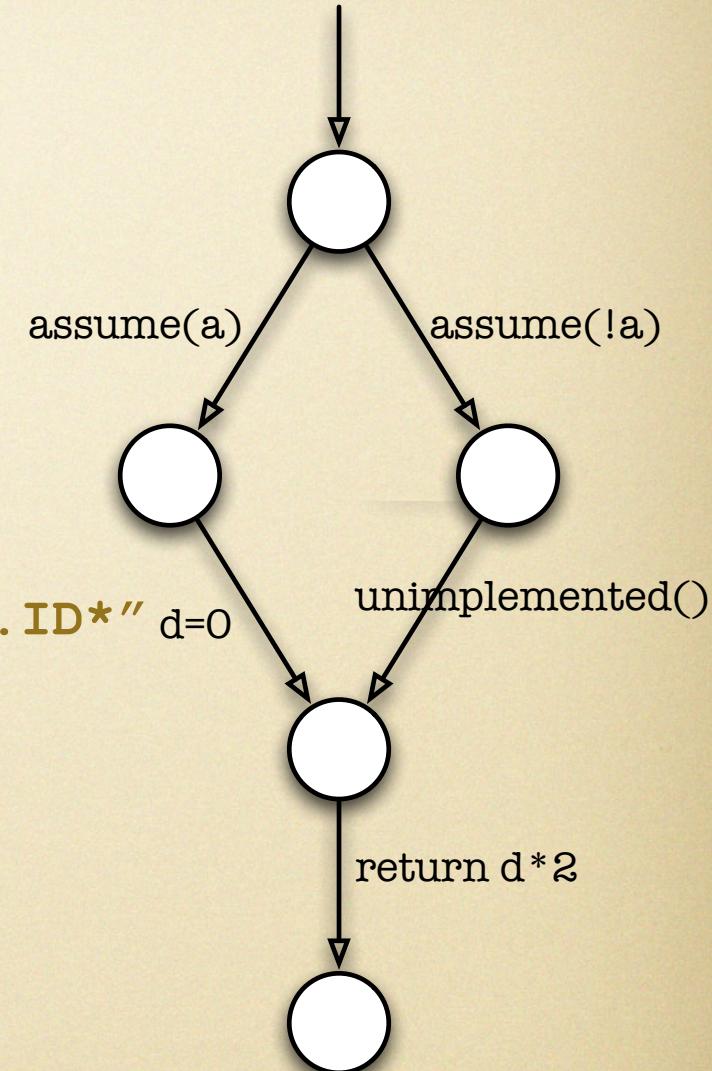
- Cover if and else branch



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

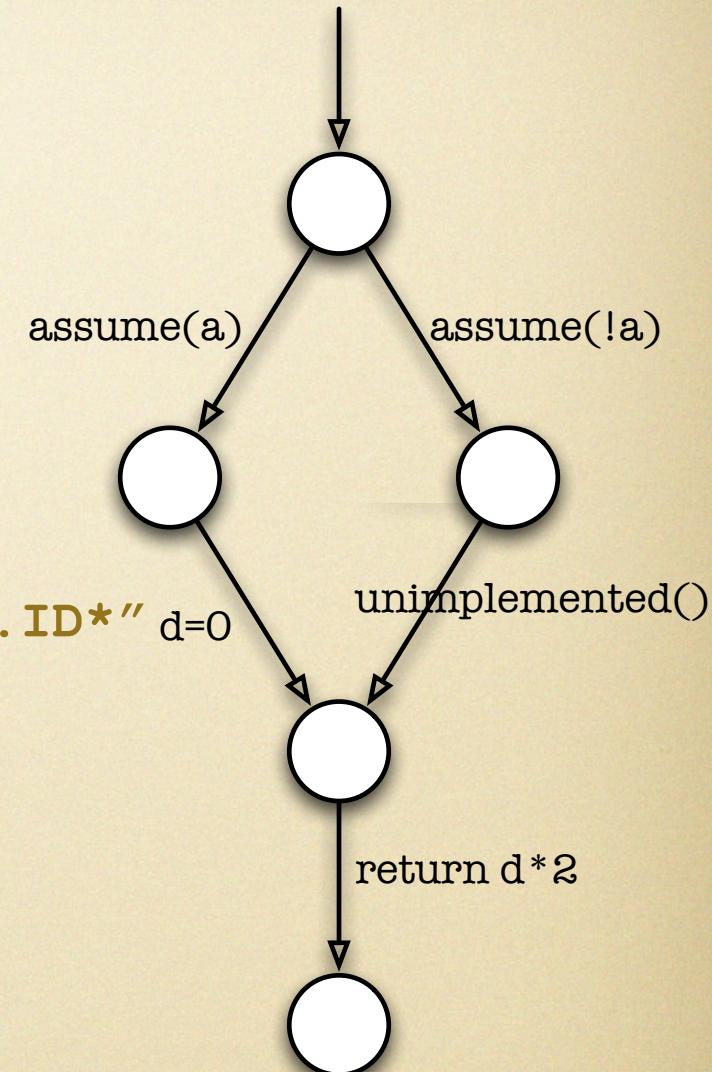
- Cover if and else branch
- **cover “ID\*.@4.ID\*” + “ID\*.@6.ID\*” d=0**



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Cover if and else branch
- **cover "ID\*.@4.ID\*"** + **"ID\*.@6.ID\*" d=0**
- 2 words - 2 test goals



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

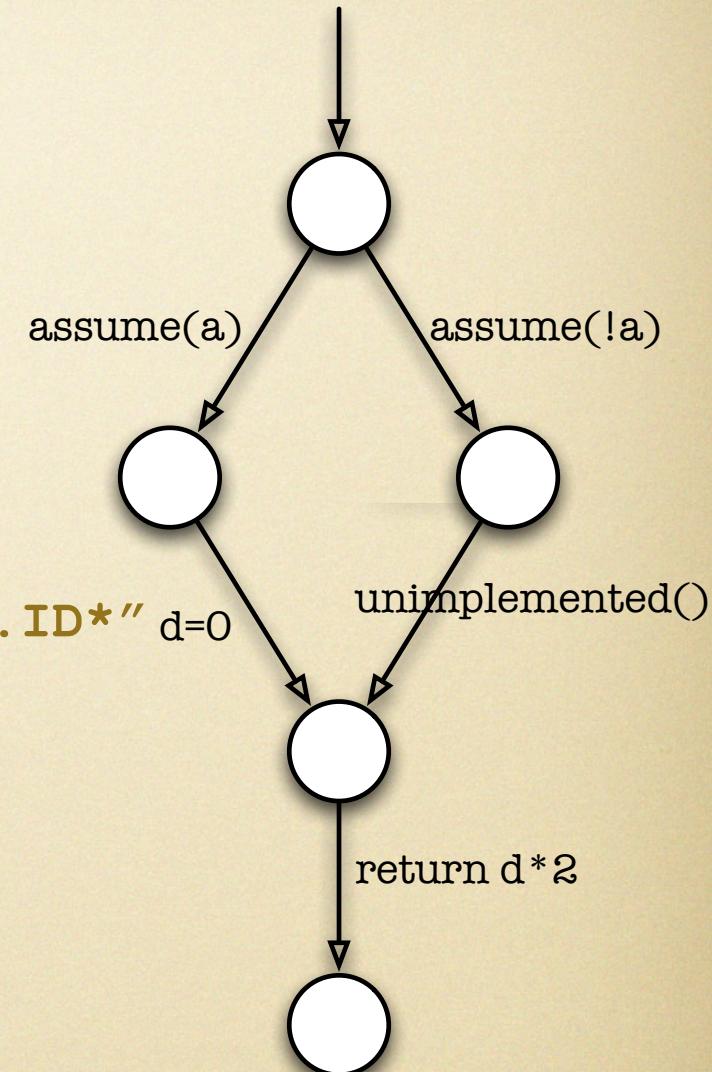
- Cover if and else branch
- cover "ID\*.@4.ID\*" + "ID\*.@6.ID\*" d=0
- 2 words - 2 test goals

IN:

a=2  
d=0

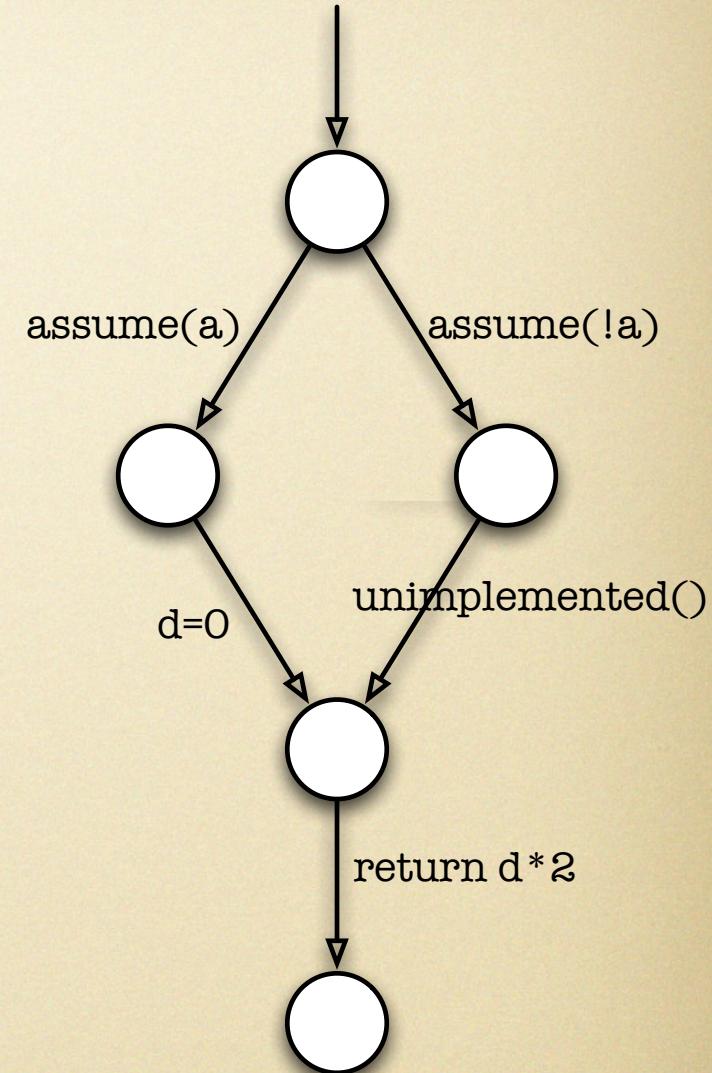
IN:

a=0  
d=0



# Coverage Patterns

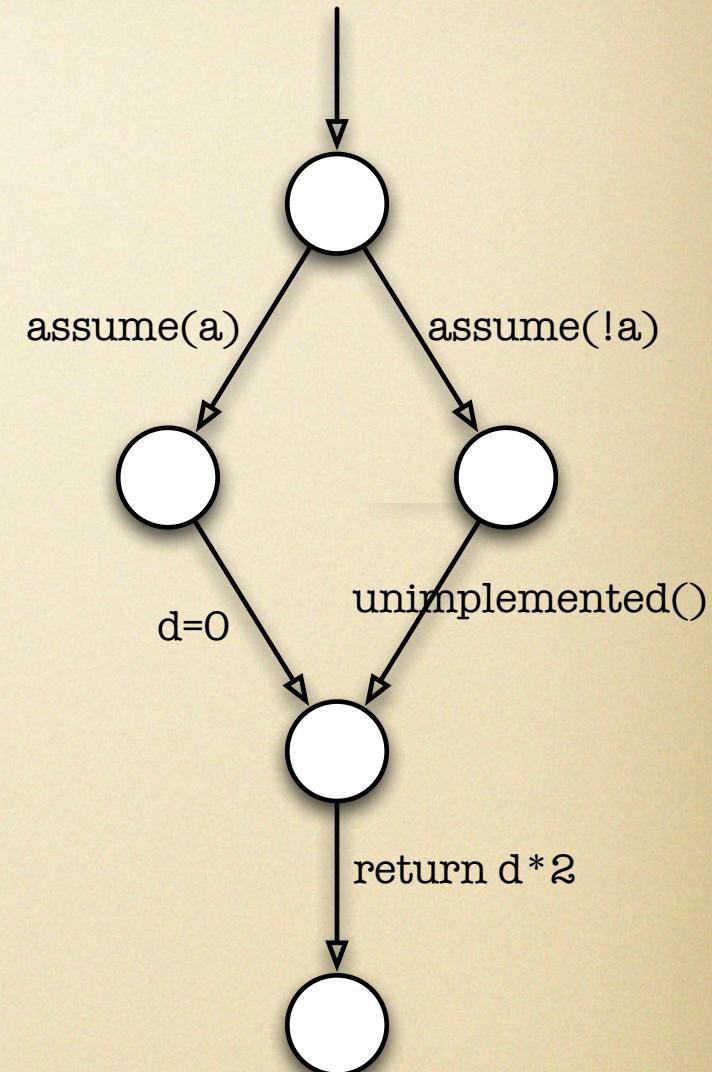
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

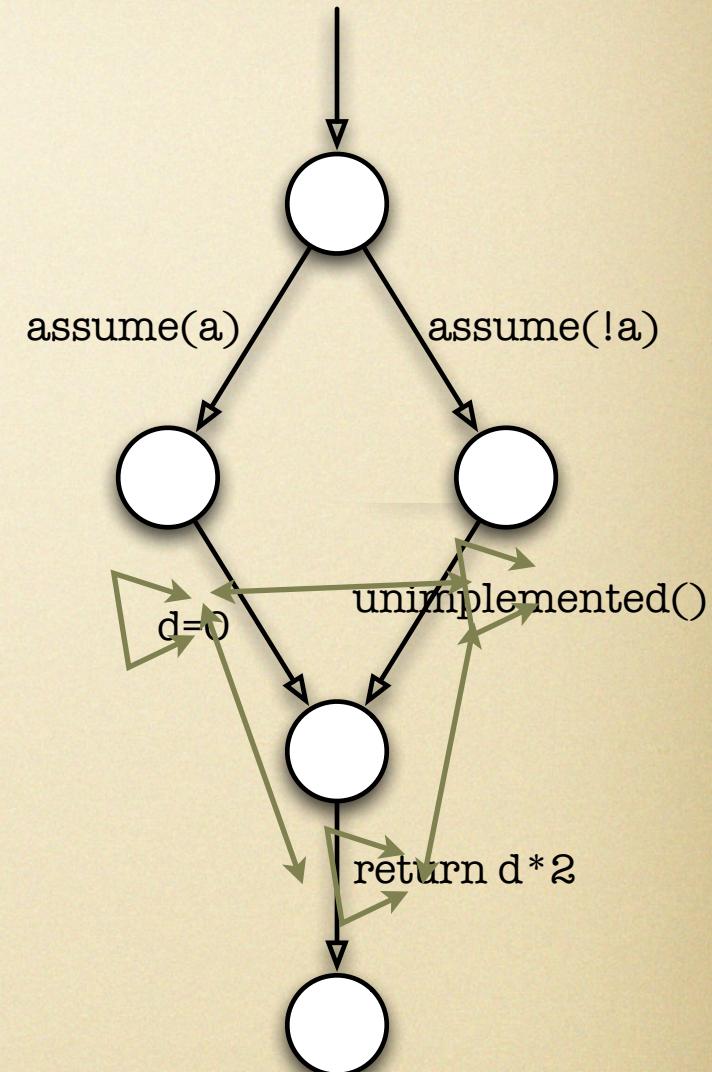
- Can we cover pairs of lines?



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

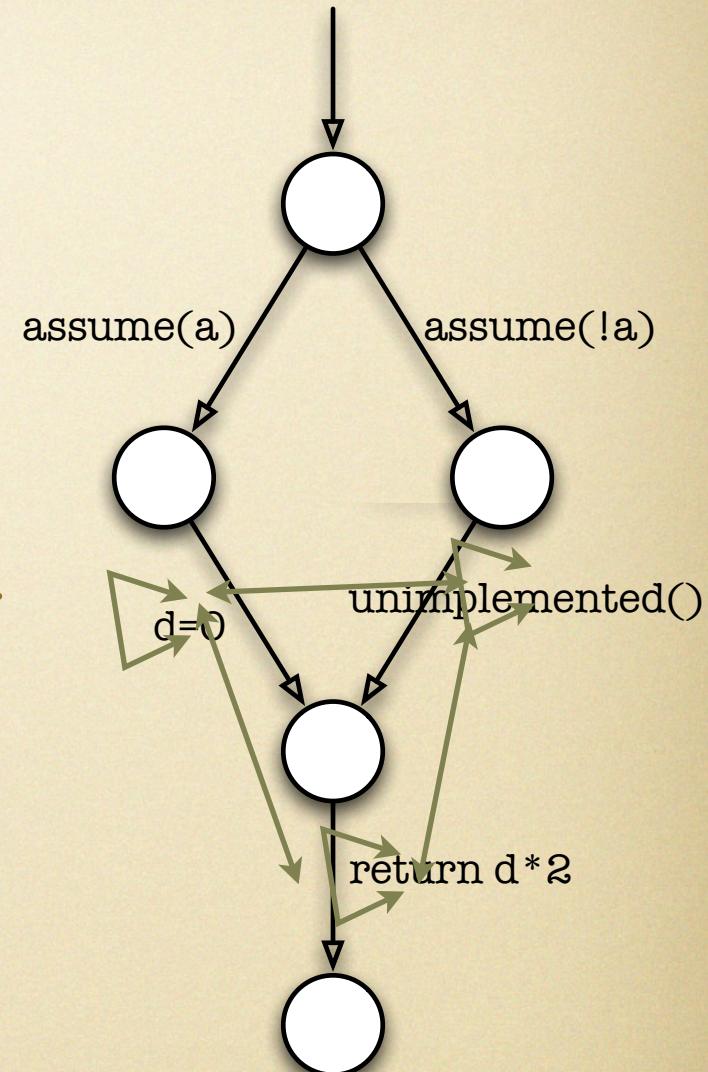
- Can we cover pairs of lines?



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

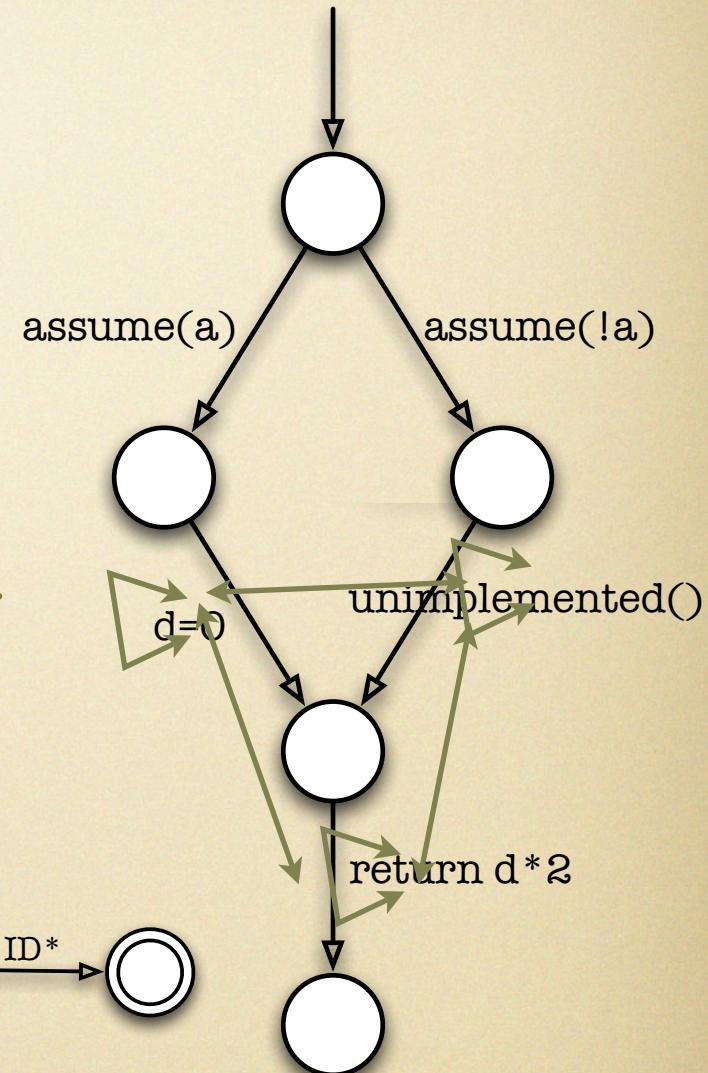
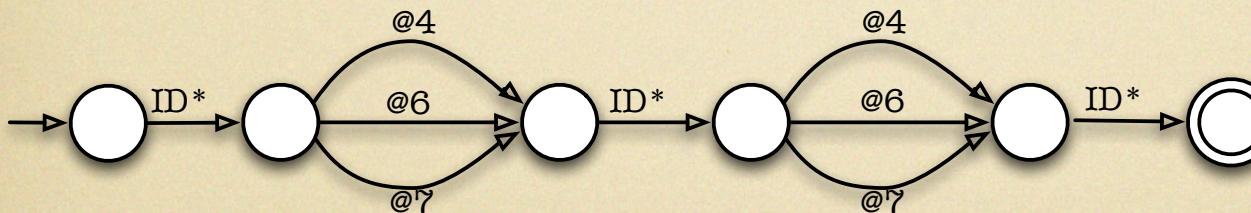
- Can we cover pairs of lines?
- **cover "ID\*".(@4+@6+@7)."ID\*".(@4+@6+@7)."ID\*"**



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

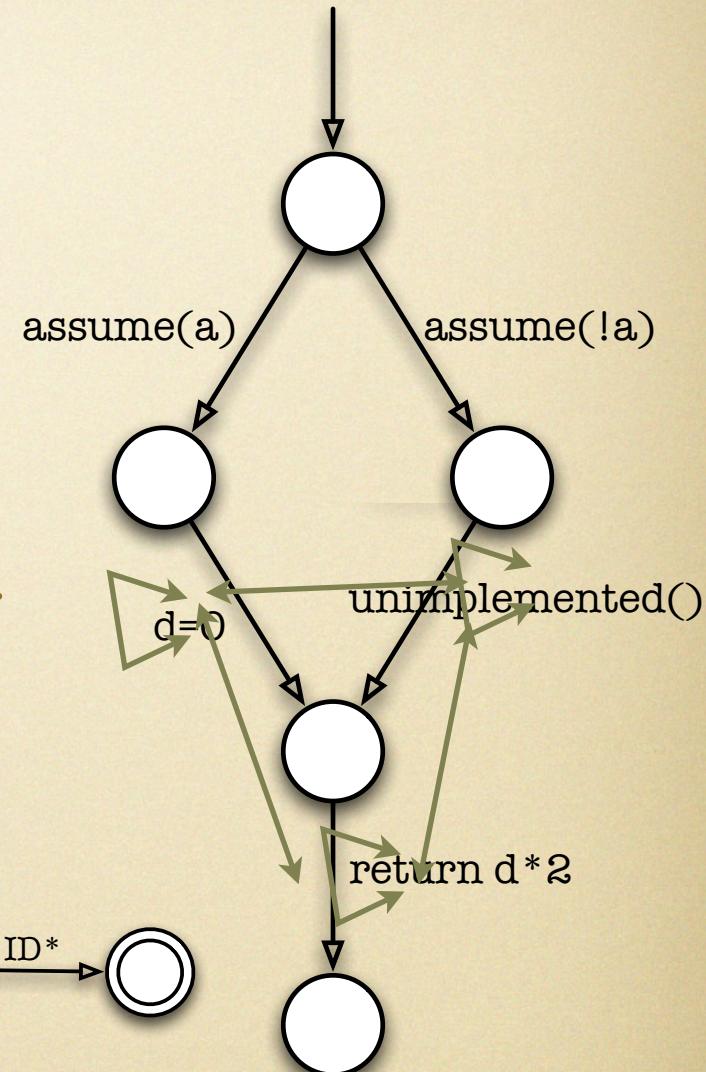
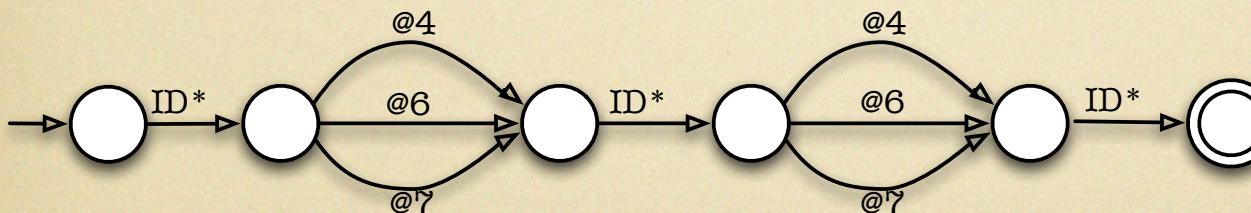
- Can we cover pairs of lines?
- **cover "ID\*".(@4+@6+@7)."ID\*".(@4+@6+@7)."ID\*"**



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Can we cover pairs of lines?
- **cover "ID\*".(@4+@6+@7)."ID\*".(@4+@6+@7)."ID\*"**
- Coverage pattern describes 9 test goals



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

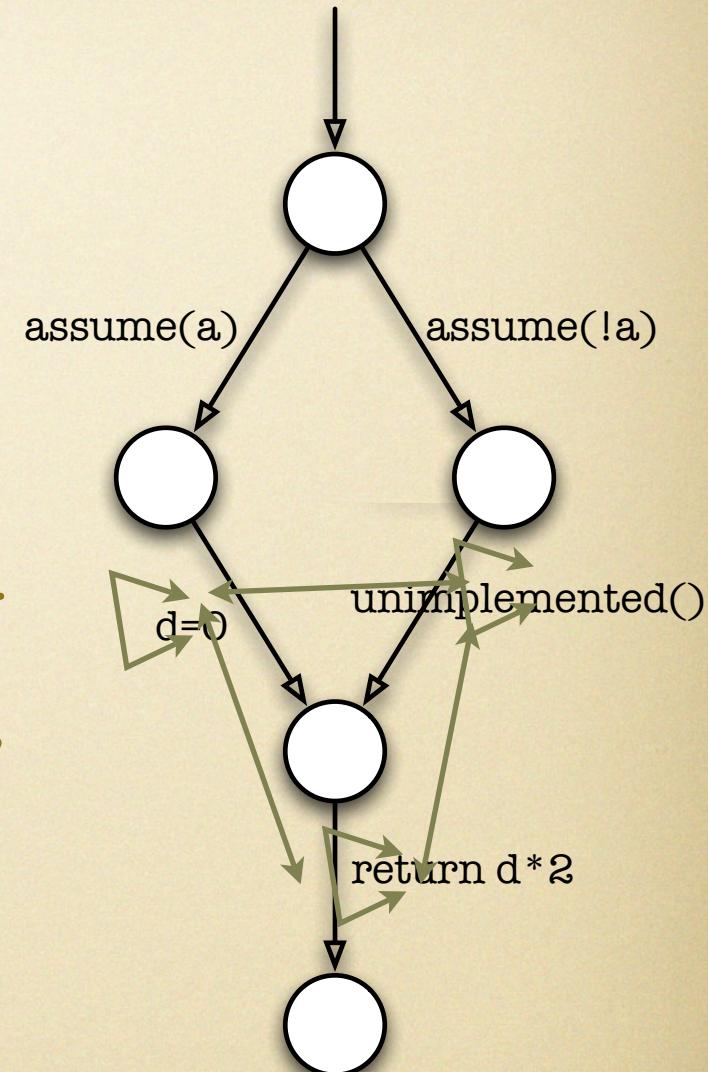
- Can we cover pairs of lines?
- **cover "ID\*".(@4+@6+@7)."ID\*".(@4+@6+@7)."ID\***
- Coverage pattern describes 9 test goals

IN:

a=67108864  
d=0

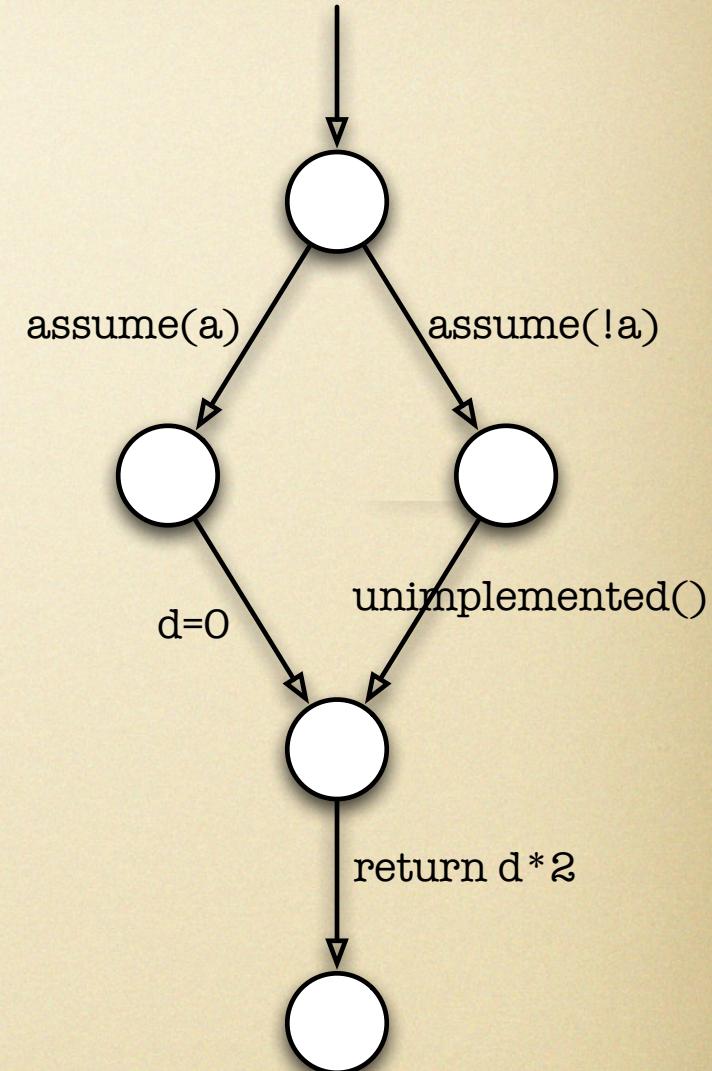
IN:

a=0  
d=0



# Coverage Patterns

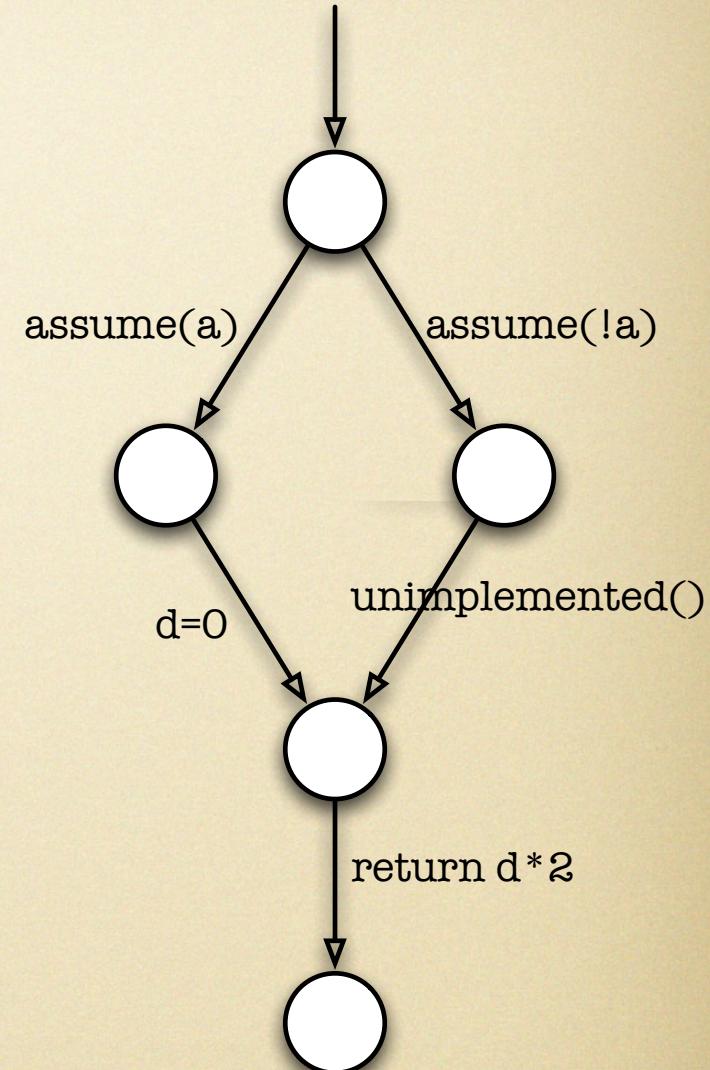
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

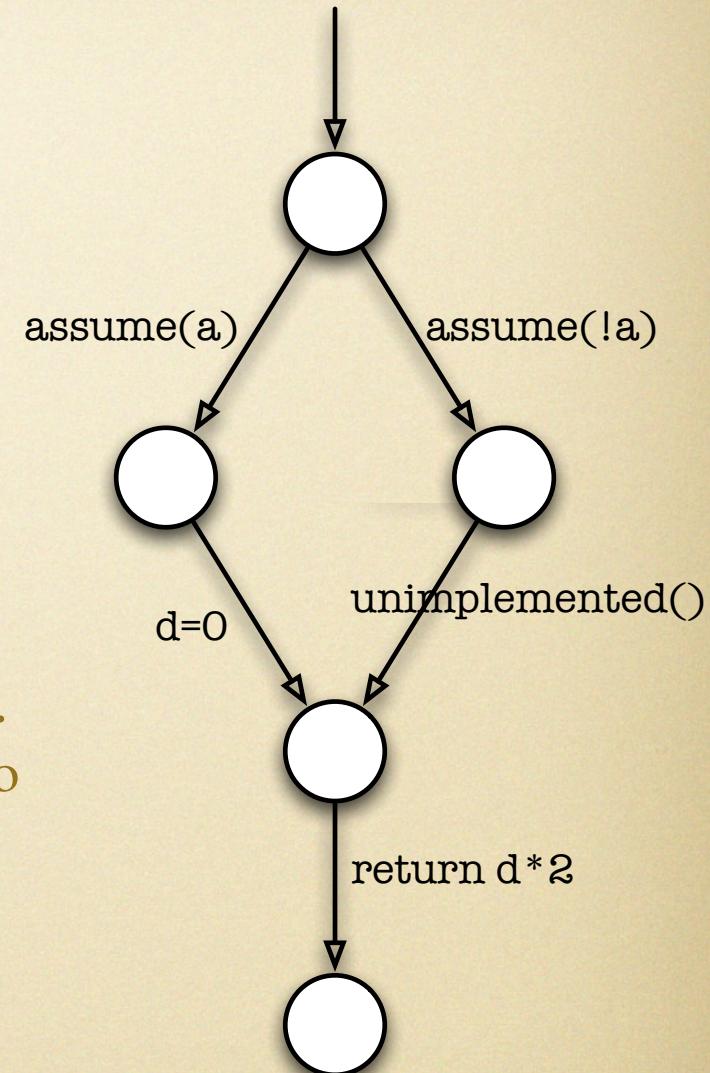
- Earlier: “For this example  
**@basicblockentry** is tantamount to  
**@4+@6+@7”**



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

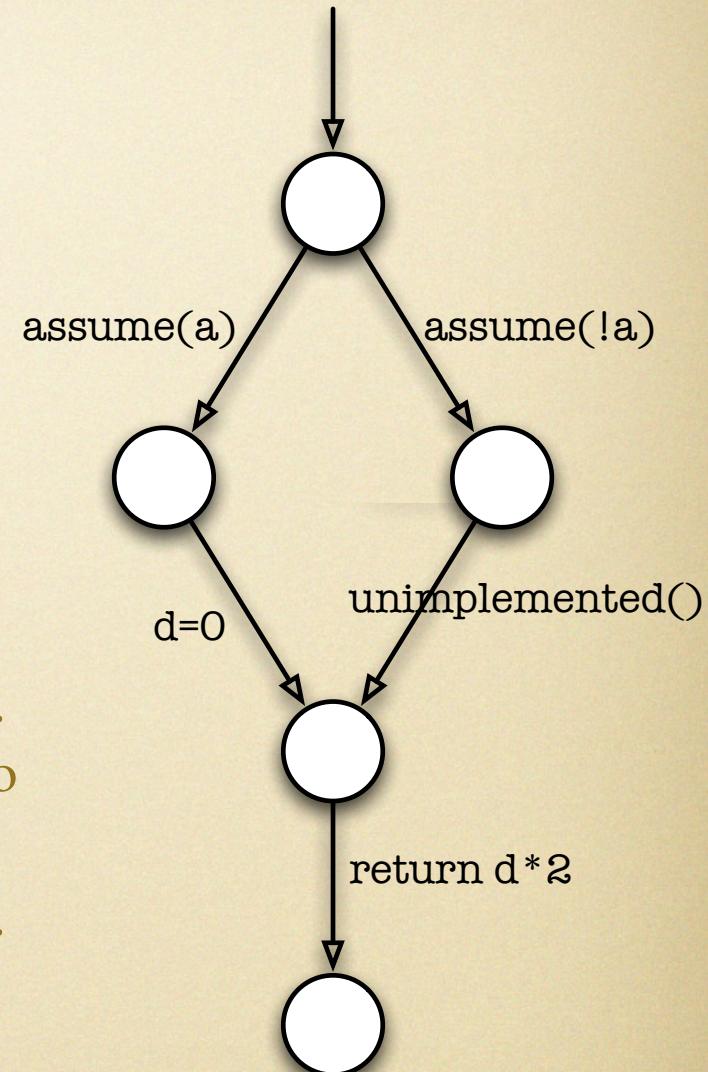
- Earlier: “*For this example @basicblockentry is tantamount to @4+@6+@7*”
- **cover “ID\*”.(@4+@6+@7).“ID\*”.(@4+@6+@7).“ID\*”** is tantamount to



# Coverage Patterns

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

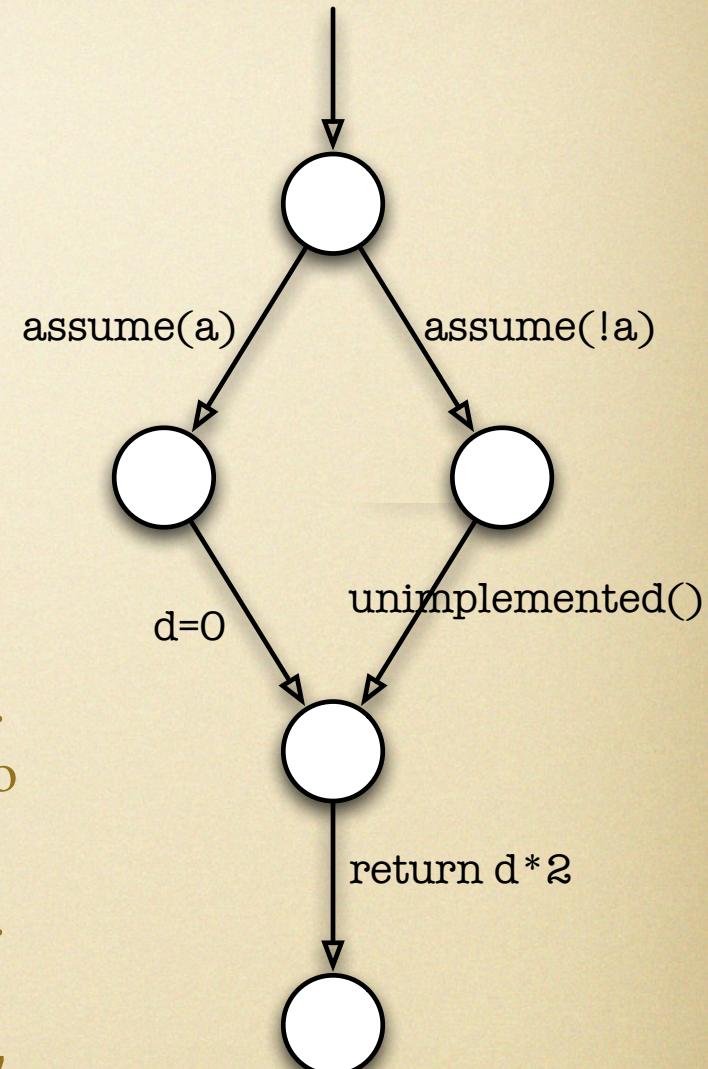
- Earlier: “*For this example @basicblockentry is tantamount to @4+@6+@7*”
- **cover “ID\*”.(@4+@6+@7).“ID\*”.** (@4+@6+@7).“ID\*” is tantamount to
- **cover “ID\*”.@basicblockentry.“ID\*”.** @basicblockentry.“ID\*”



# Coverage Patterns

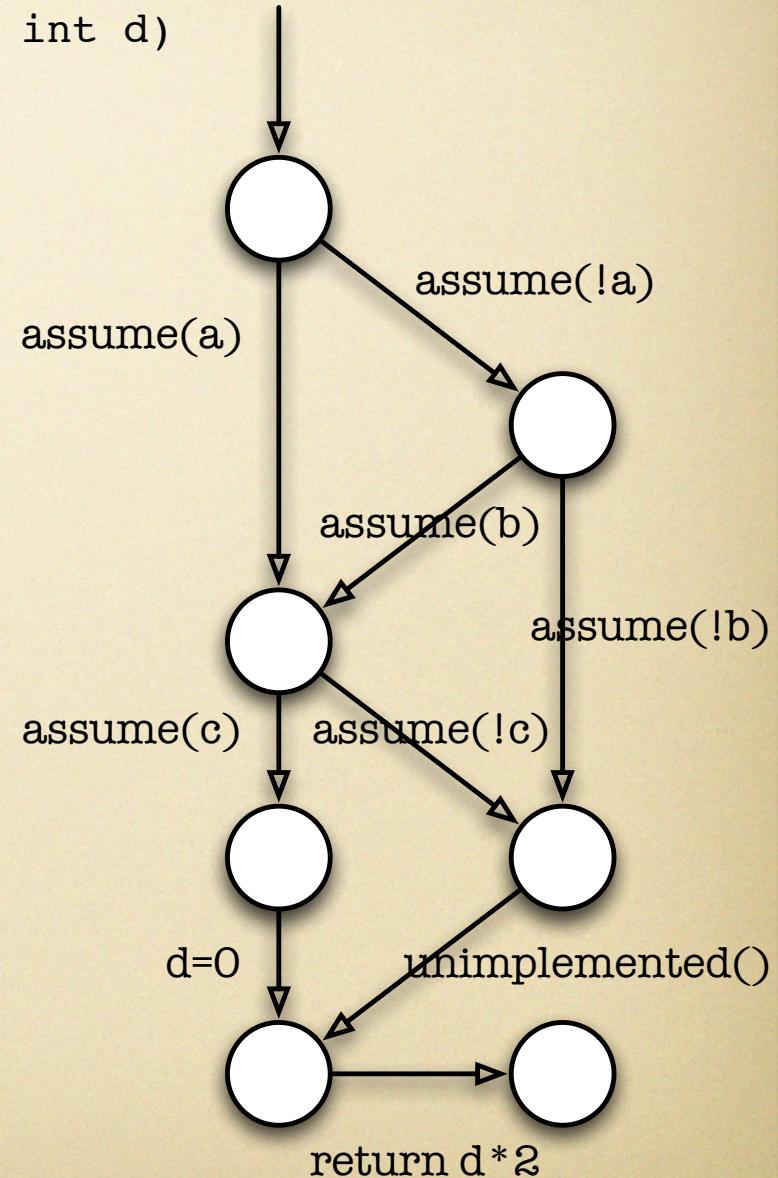
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- Earlier: “*For this example @basicblockentry is tantamount to @4+@6+@7*”
- **cover “ID\*”.(@4+@6+@7).“ID\*”.** (@4+@6+@7).“ID\*” is tantamount to
- **cover** “ID\*”.@basicblockentry.“ID\*”.@basicblockentry.“ID\*”
- **@basicblockentry** and **@4+@6+@7** describe the same language of CFA edges



# Edges and Beyond

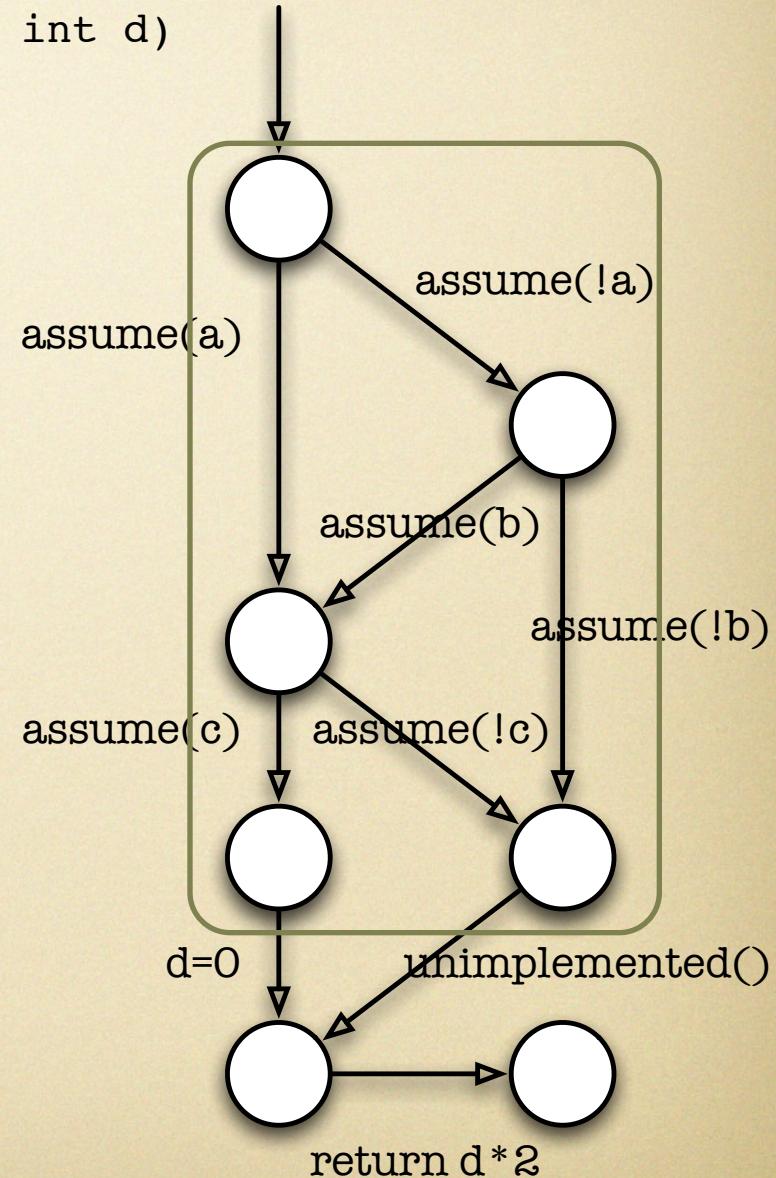
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

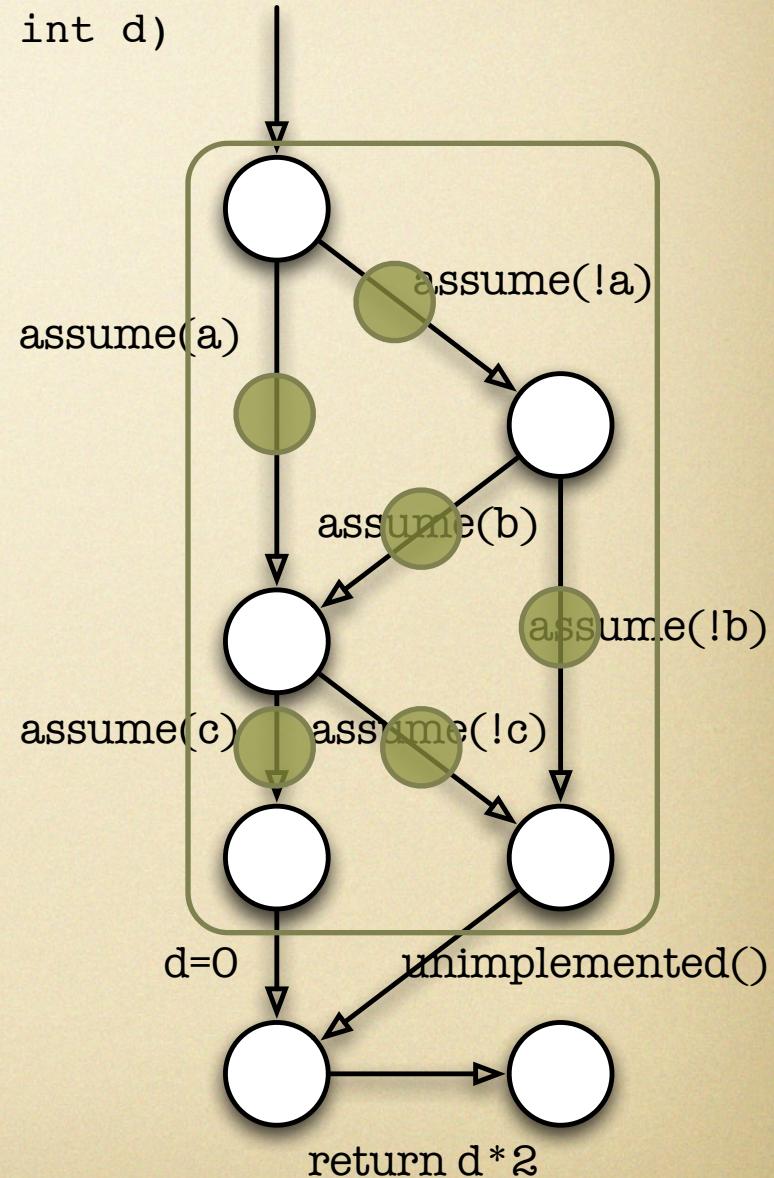
- cover @conditiongraph



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

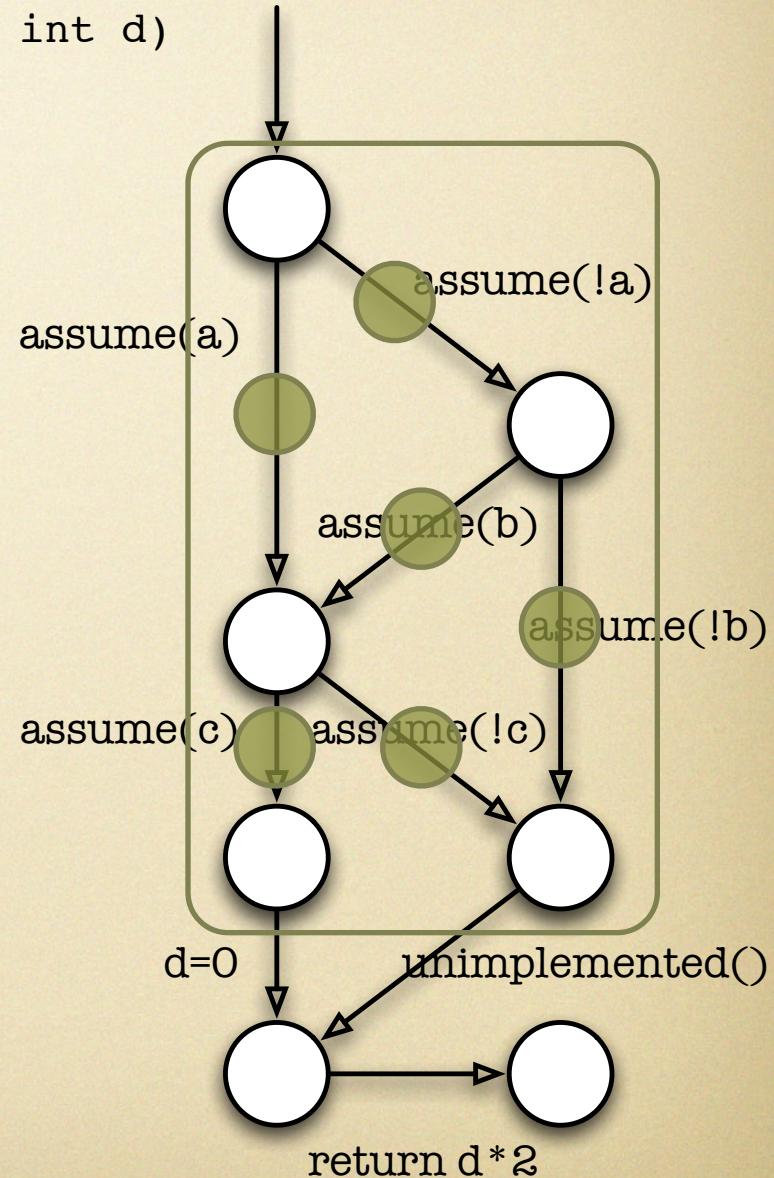
- cover @conditiongraph
- cover EDGES (@conditiongraph)



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

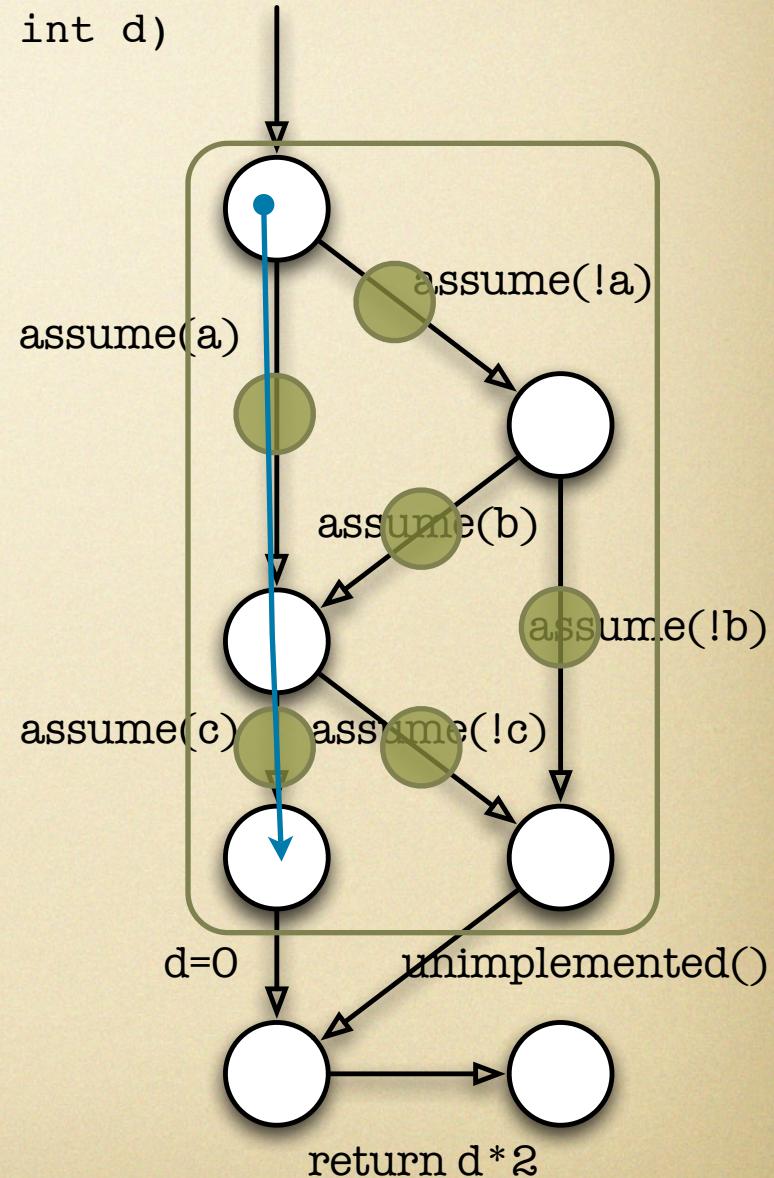
- cover @conditiongraph
- cover EDGES (@conditiongraph)
- 6 test goals, 3 test cases suffice:



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

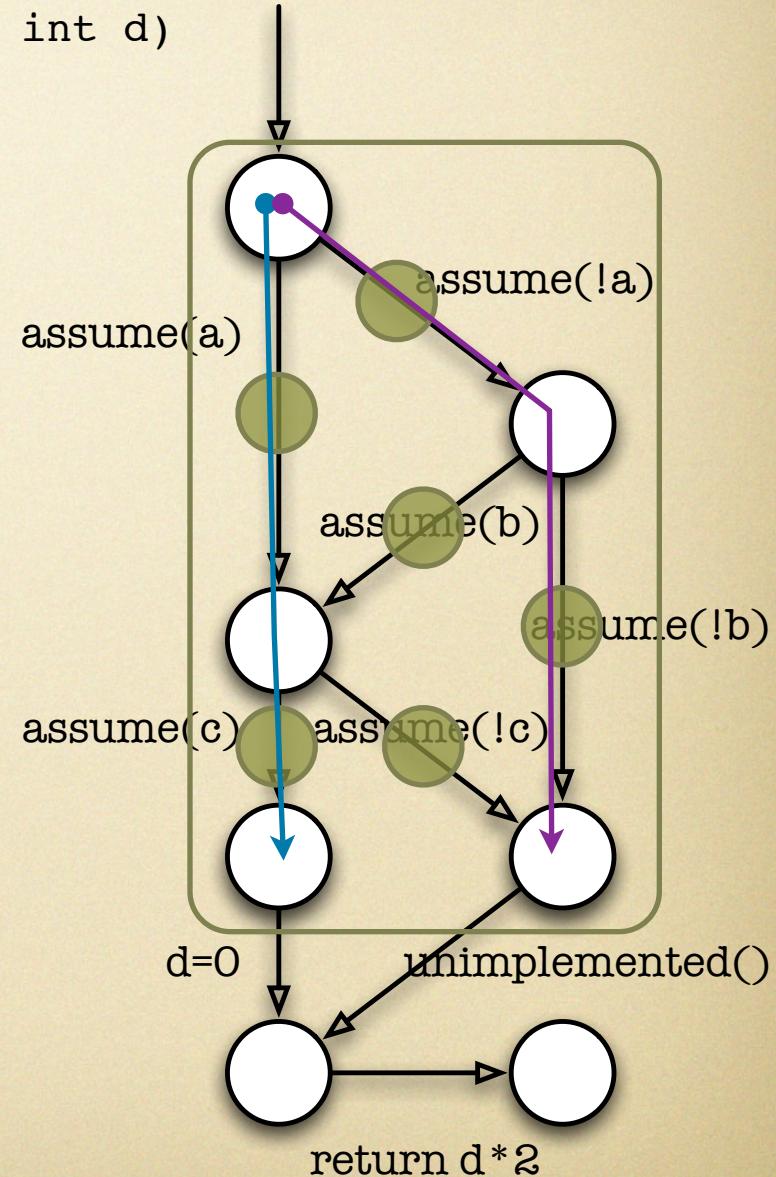
- cover @conditiongraph
- cover EDGES (@conditiongraph)
- 6 test goals, 3 test cases suffice:
  - $a = 1, c = 1$



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

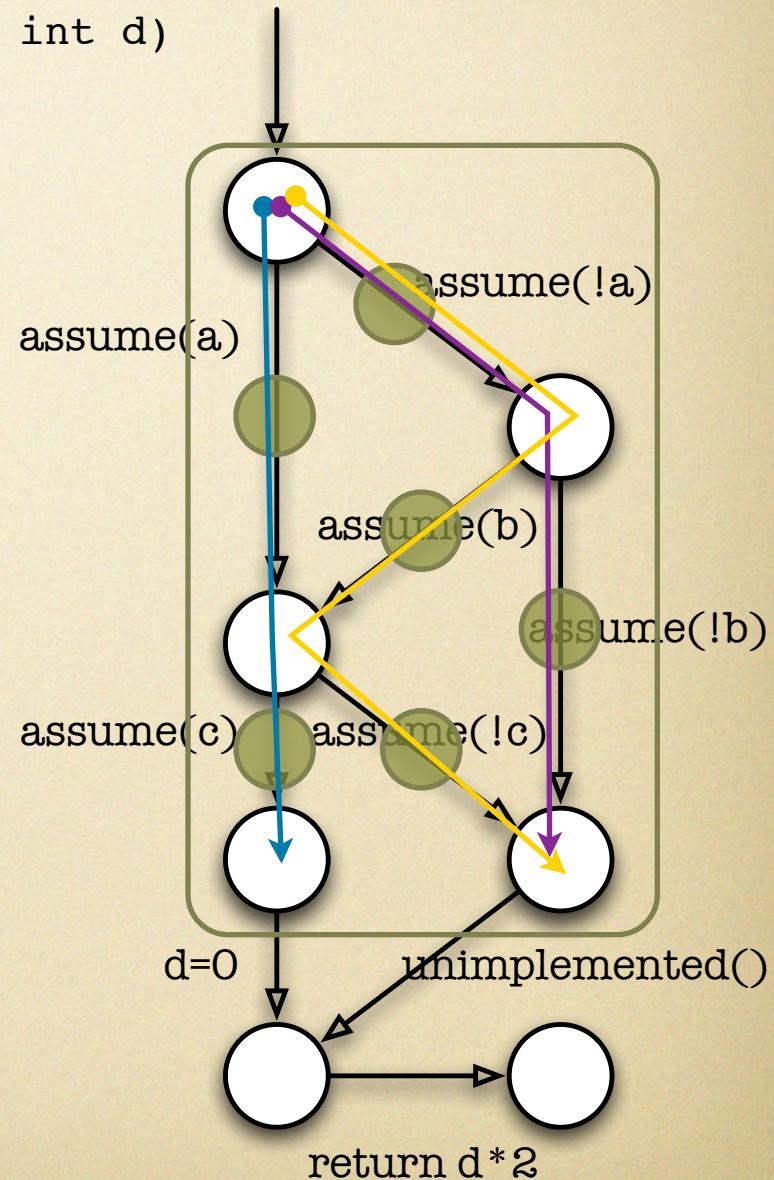
- cover @conditiongraph
- cover EDGES (@conditiongraph)
- 6 test goals, 3 test cases suffice:
  - a = 1, c = 1
  - a = 0, b = 0



# Edges and Beyond

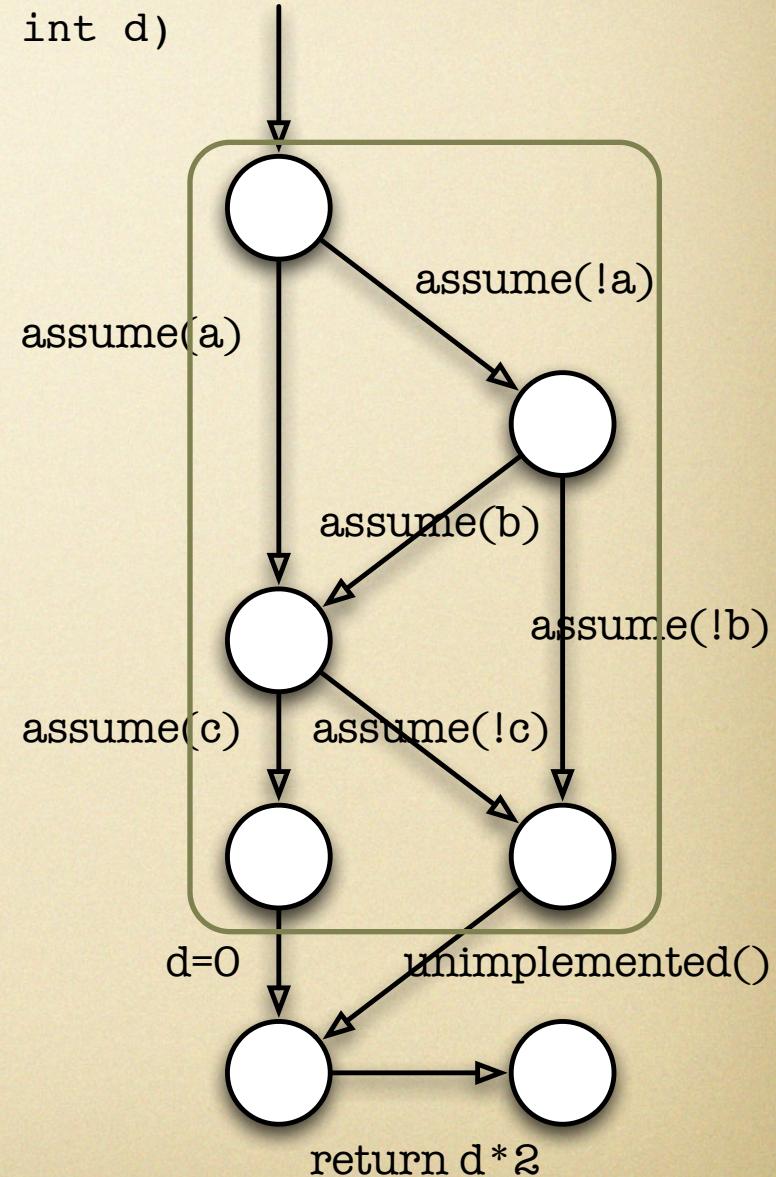
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- cover @conditiongraph
- cover EDGES (@conditiongraph)
- 6 test goals, 3 test cases suffice:
  - a = 1, c = 1
  - a = 0, b = 0
  - a = 0, b = 1, c = 0



# Edges and Beyond

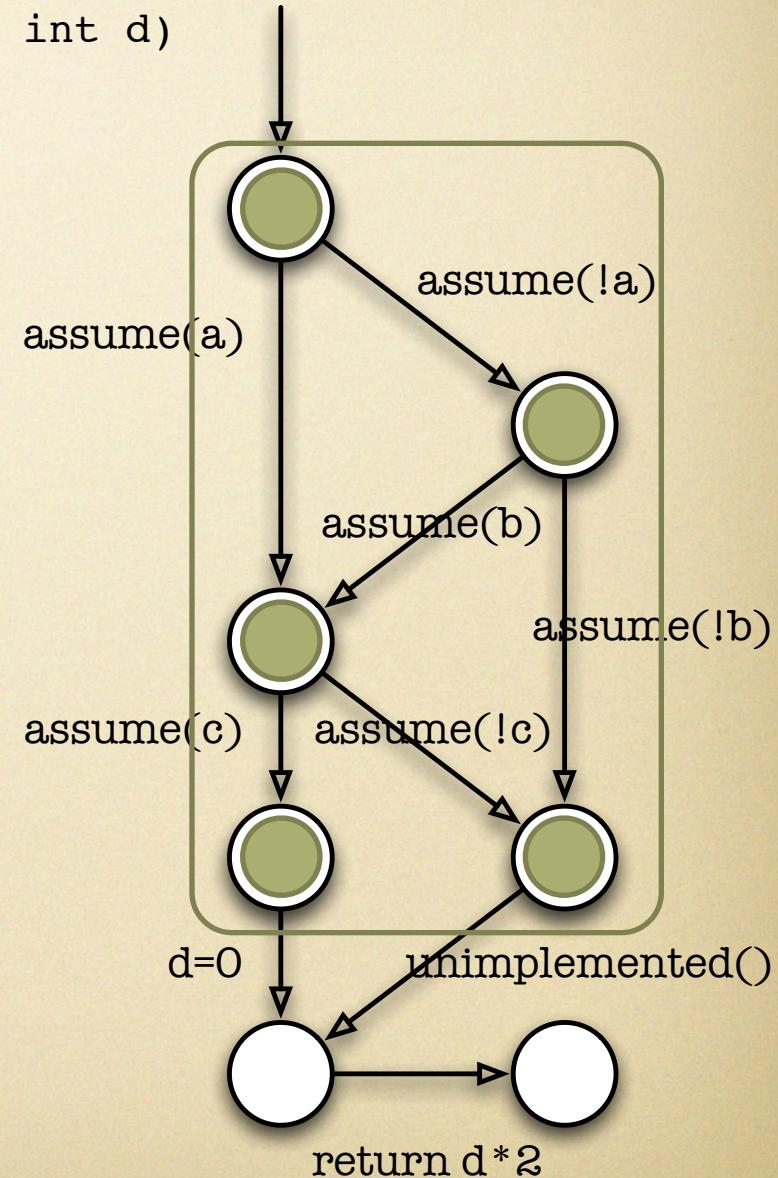
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

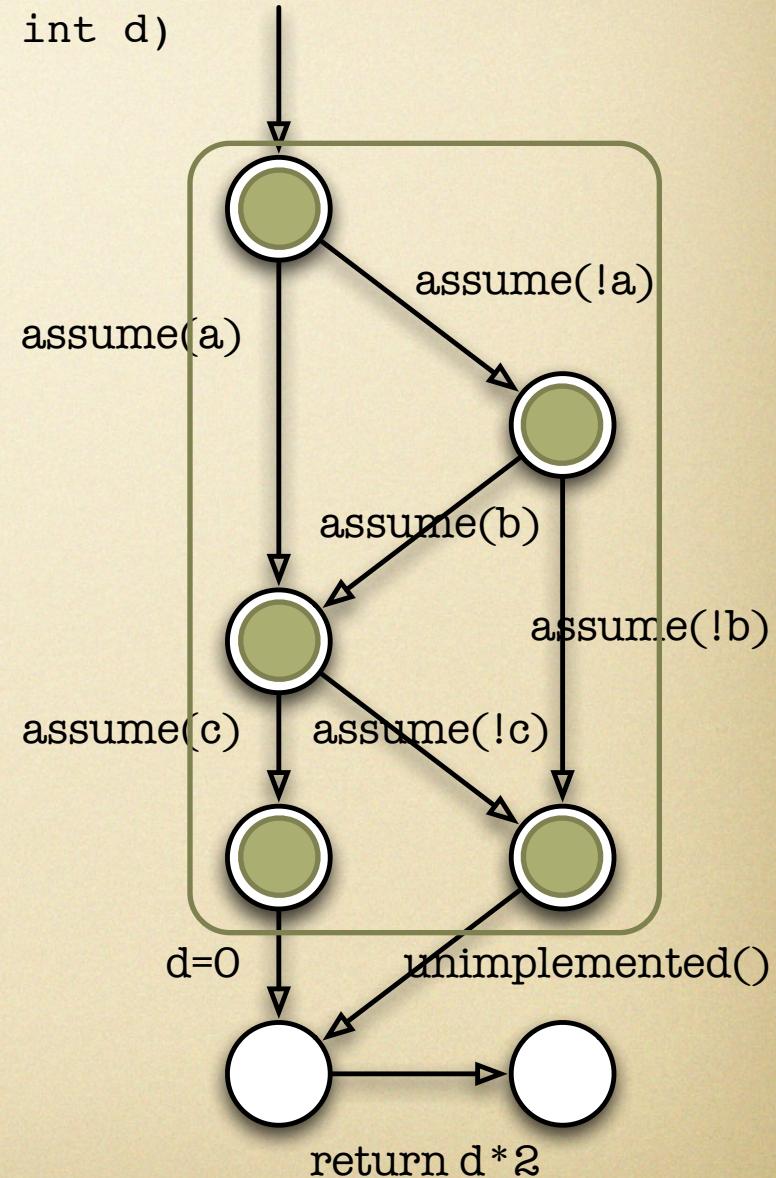
- cover NODES (@conditiongraph)



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

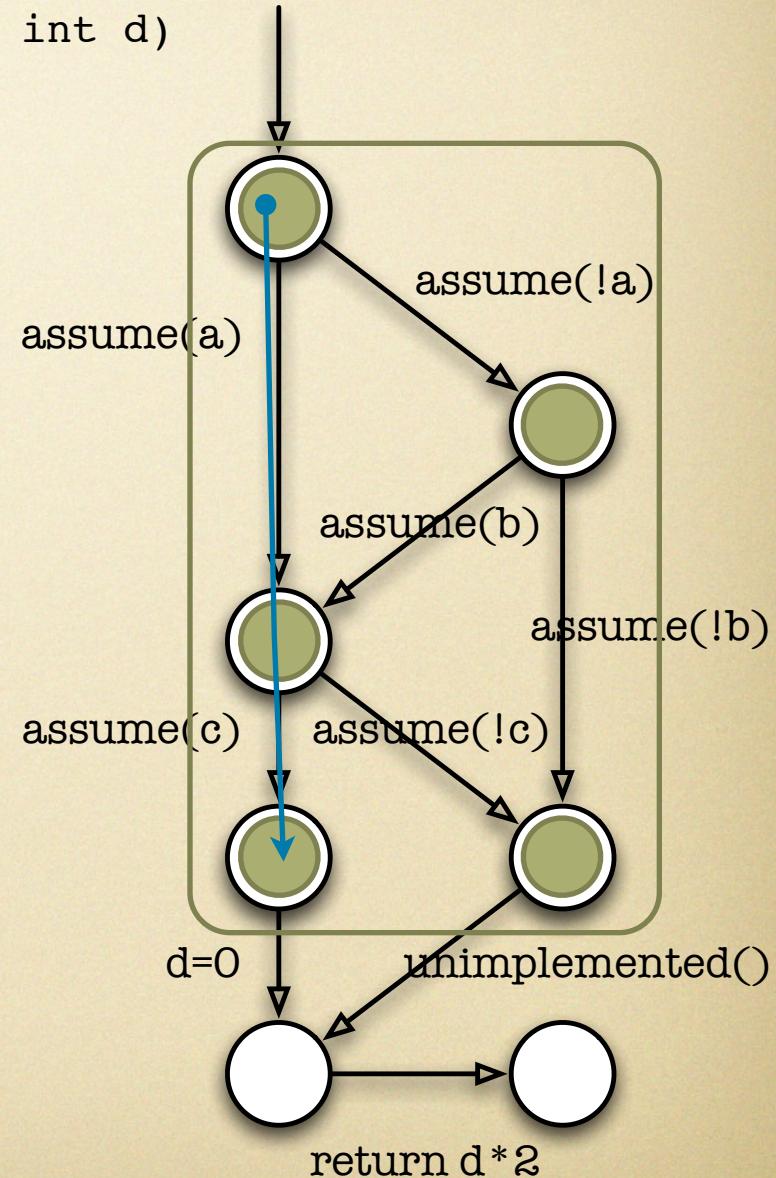
- cover NODES (@conditiongraph)
- 5 test goals, 2 test cases suffice:



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

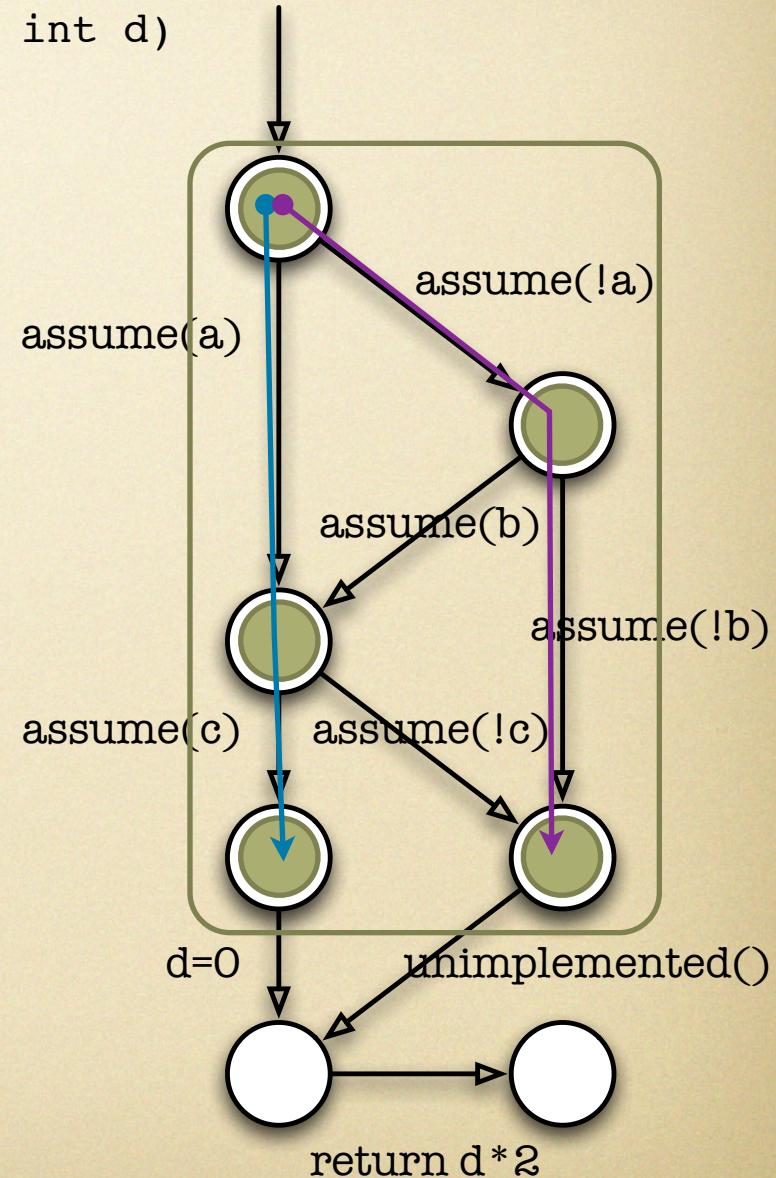
- cover NODES (@conditiongraph)
- 5 test goals, 2 test cases suffice:
  - $a = 1, c = 1$



# Edges and Beyond

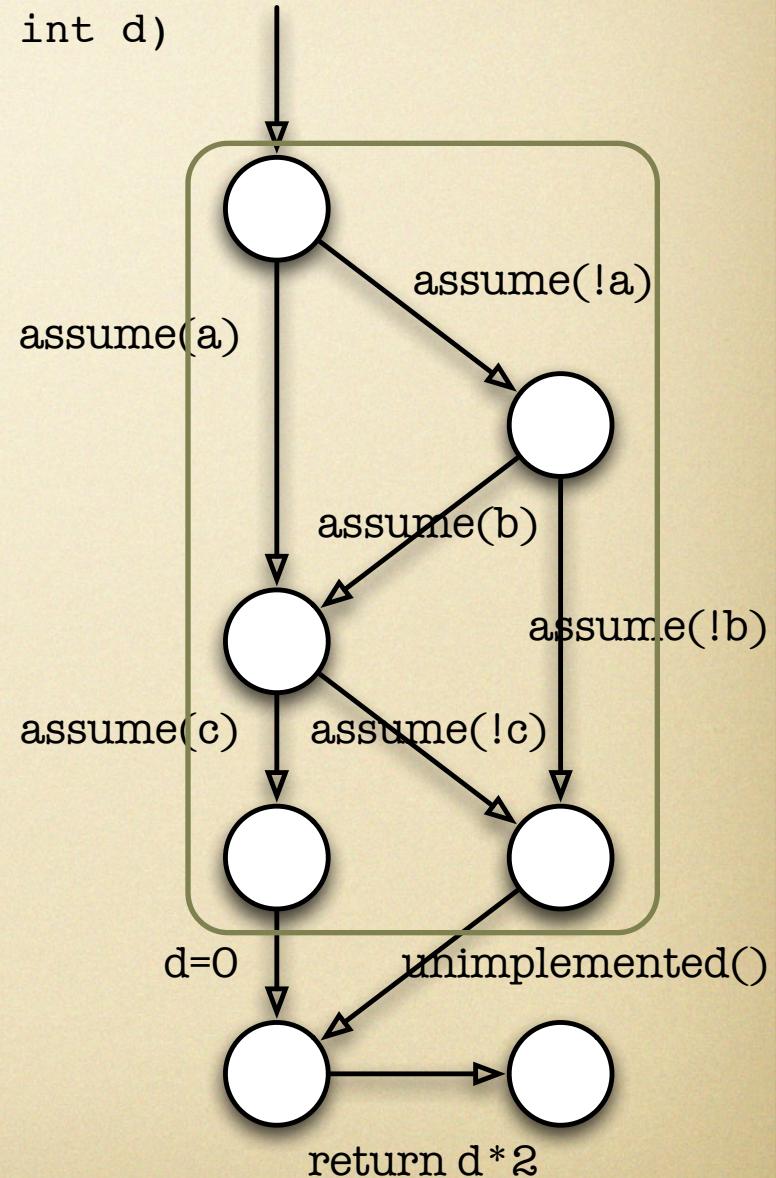
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- cover NODES (@conditiongraph)
- 5 test goals, 2 test cases suffice:
  - $a = 1, c = 1$
  - $a = 0, b = 0$



# Edges and Beyond

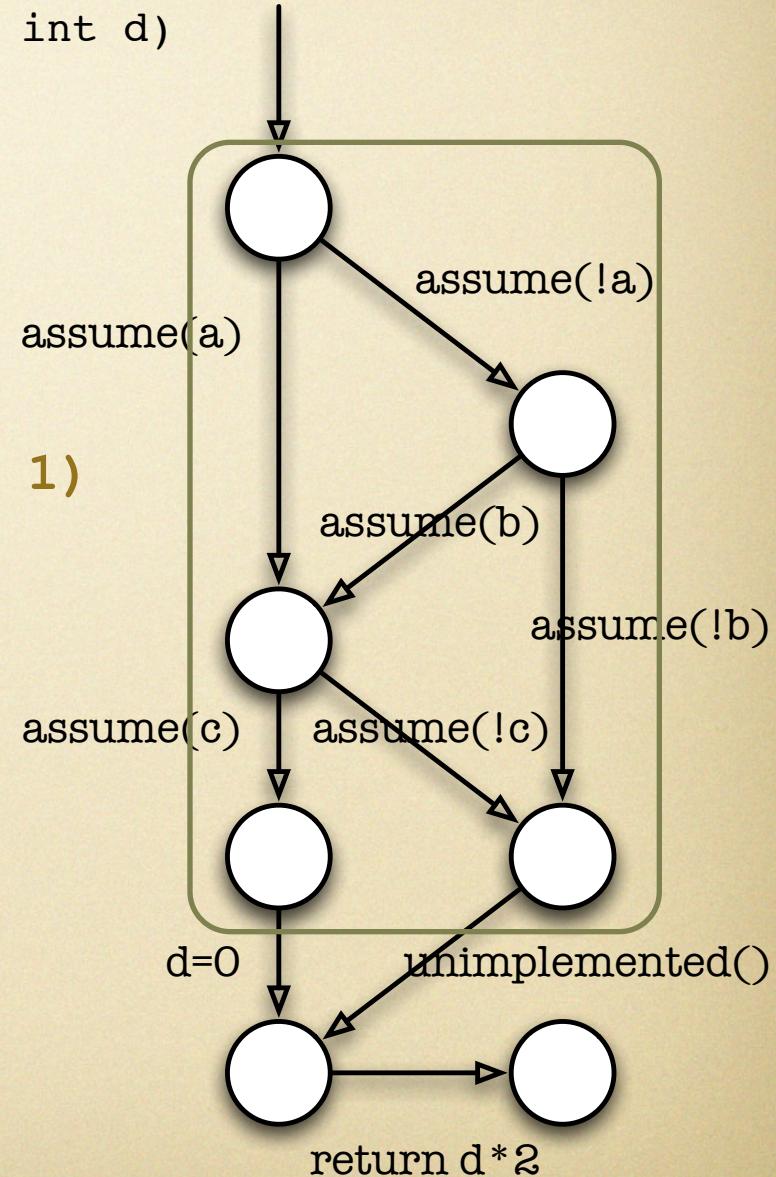
```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

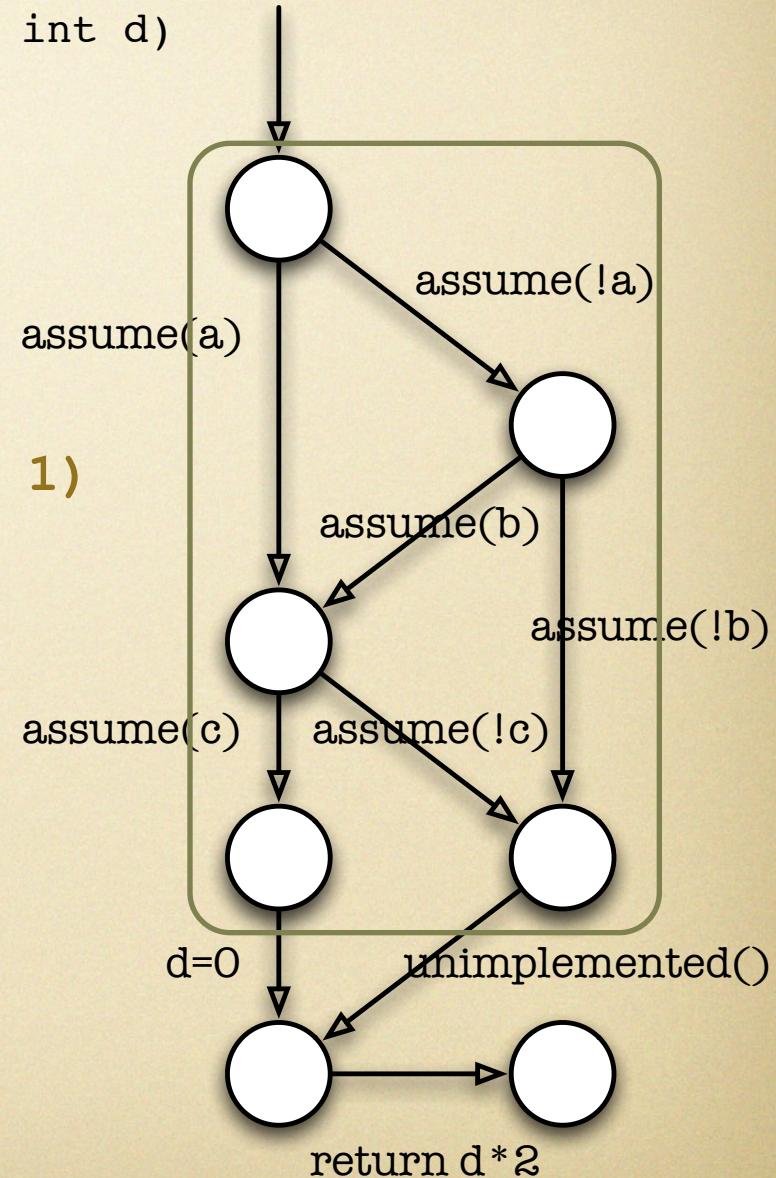
- cover PATHS (@conditiongraph, 1)
- Includes repetition bound (finiteness)



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

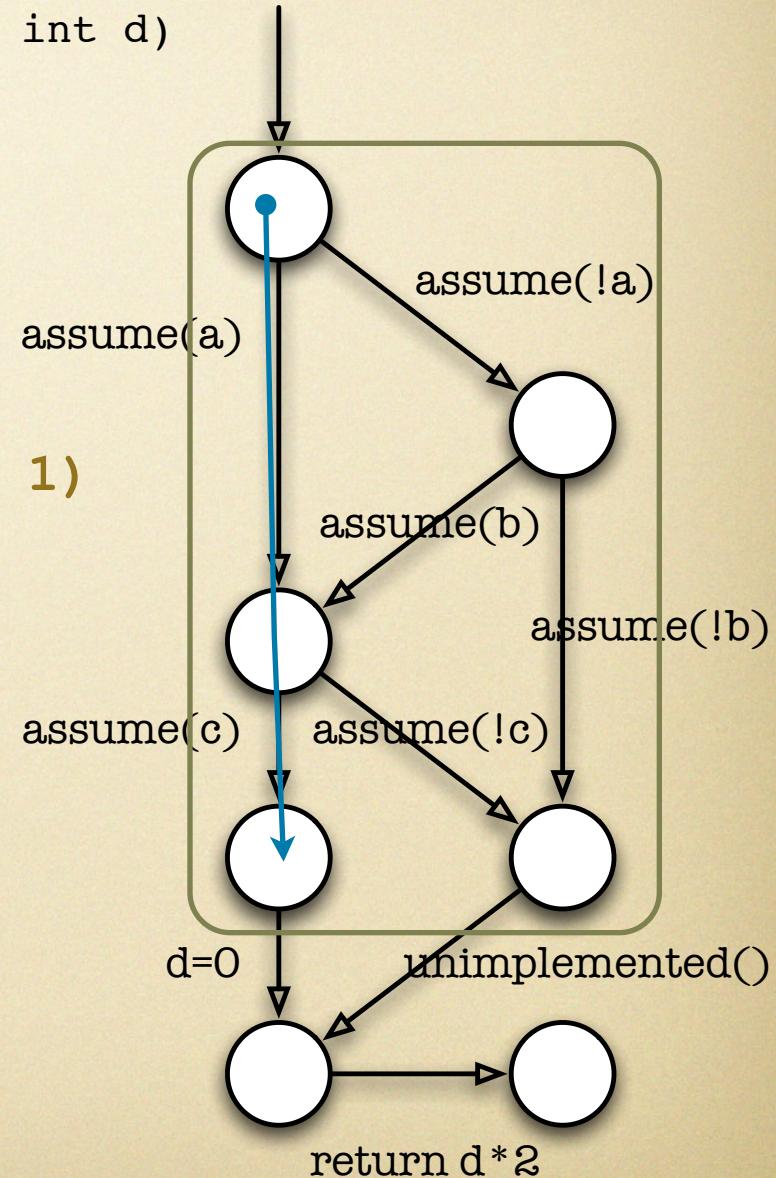
- cover PATHS (@conditiongraph, 1)
- Includes repetition bound (finiteness)
- 5 test goals, 5 test cases required:



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

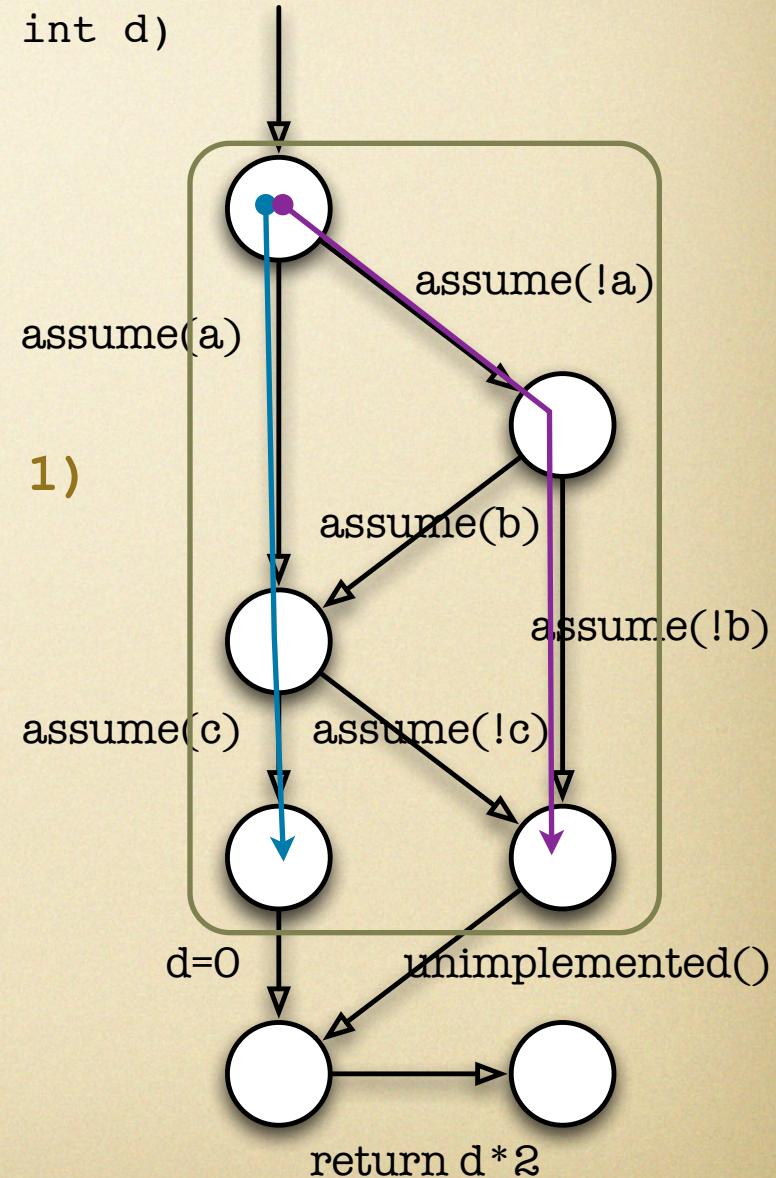
- cover PATHS (@conditiongraph, 1)
- Includes repetition bound (finiteness)
- 5 test goals, 5 test cases required:
  - $a = 1, c = 1$



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

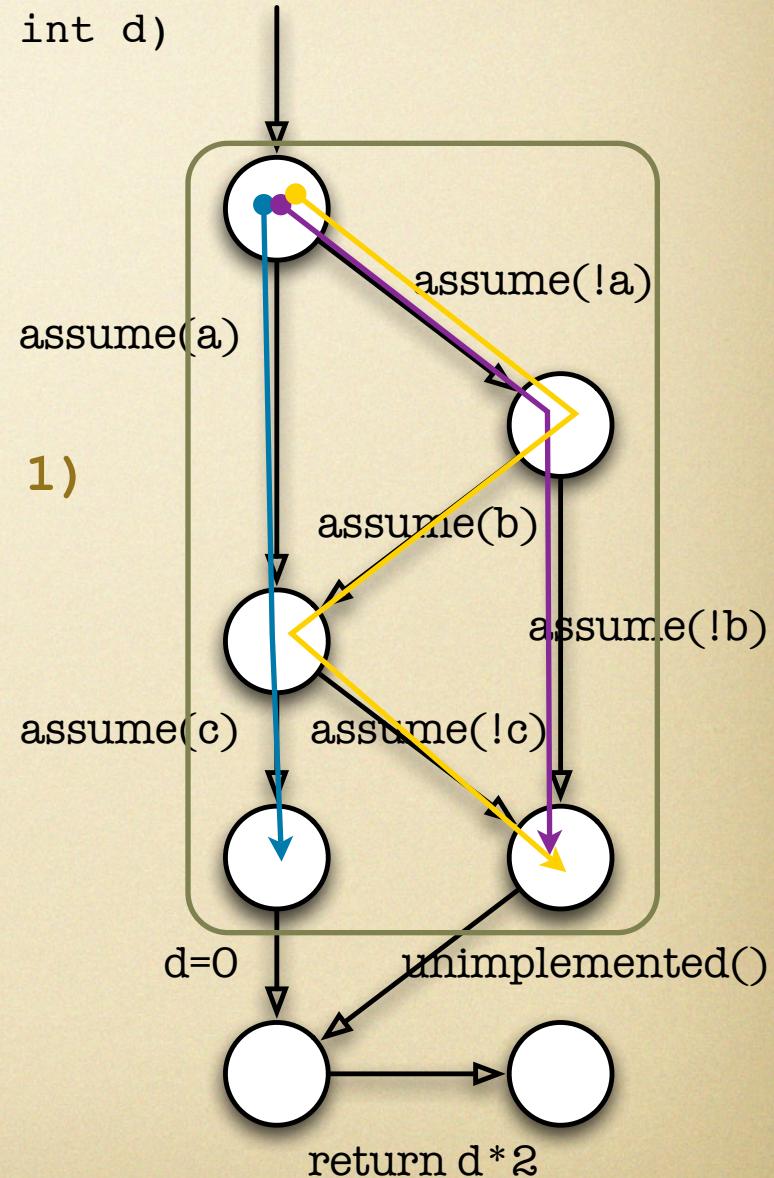
- cover PATHS (@conditiongraph, 1)
- Includes repetition bound (finiteness)
- 5 test goals, 5 test cases required:
  - $a = 1, c = 1$
  - $a = 0, b = 0$



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

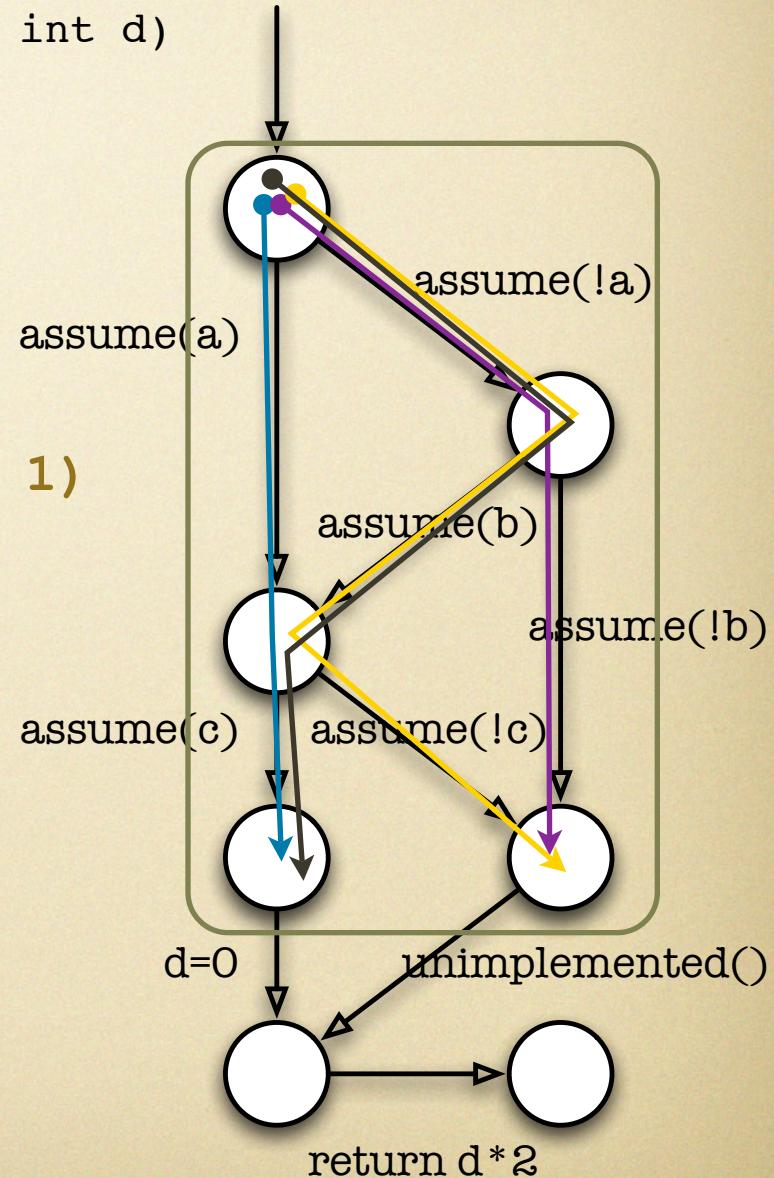
- cover PATHS (@conditiongraph, 1)
- Includes repetition bound (finiteness)
- 5 test goals, 5 test cases required:
  - $a = 1, c = 1$
  - $a = 0, b = 0$
  - $a = 0, b = 1, c = 0$



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

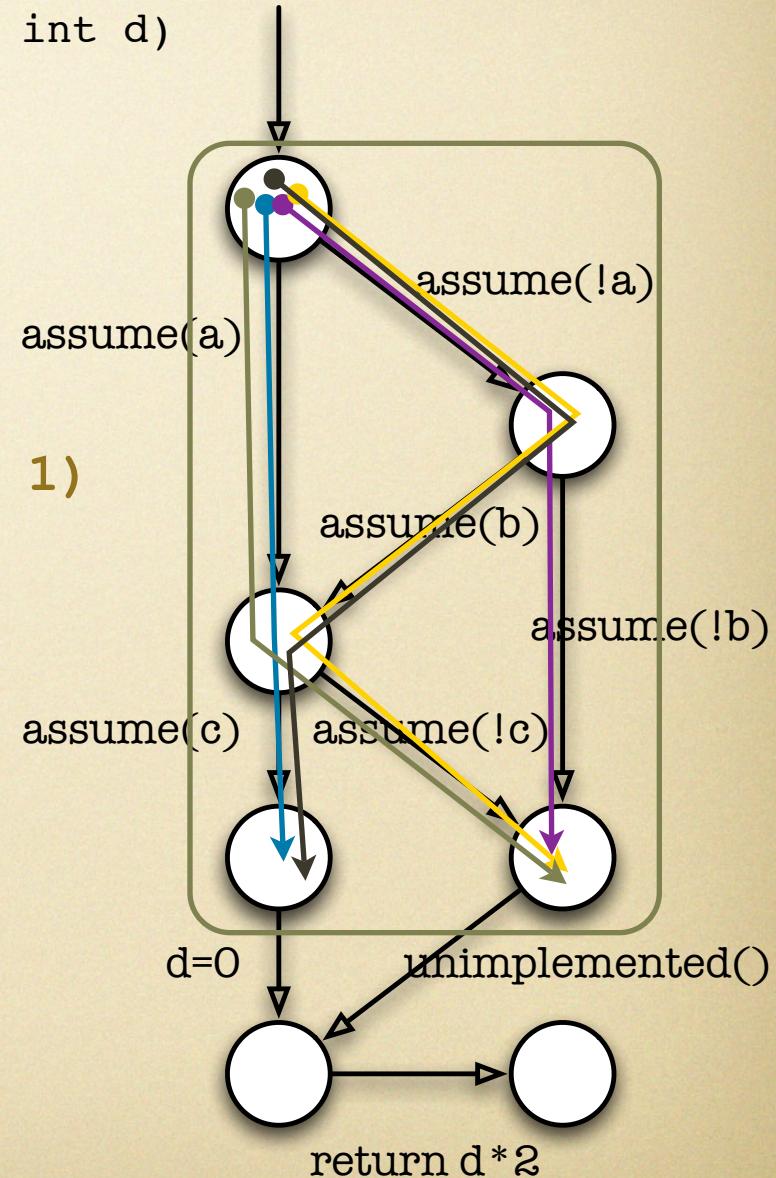
- cover PATHS (@conditiongraph, 1)
- Includes repetition bound (finiteness)
- 5 test goals, 5 test cases required:
  - $a = 1, c = 1$
  - $a = 0, b = 0$
  - $a = 0, b = 1, c = 0$
  - $a = 0, b = 1, c = 1$



# Edges and Beyond

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

- cover PATHS (@conditiongraph, 1)
- Includes repetition bound (finiteness)
- 5 test goals, 5 test cases required:
  - $a = 1, c = 1$
  - $a = 0, b = 0$
  - $a = 0, b = 1, c = 0$
  - $a = 0, b = 1, c = 1$
  - $a = 1, c = 0$



# FQL Queries

# FQL Queries

**in scope cover goals passing constraints**

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer/filter function

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer/filter function
- *goals*: coverage pattern

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer / filter function
- *goals*: coverage pattern
- *constraints*: path pattern

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer / filter function
- *goals*: coverage pattern
- *constraints*: path pattern

“For each test goal inside scope, provide a test case which satisfies the passing constraints.”

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer / filter function
- *goals*: coverage pattern
- *constraints*: path pattern

“For each test goal inside scope, provide a test case which satisfies the passing constraints.”

Condition coverage in function *partition* with test cases that reach *line 7* at least once.

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer/filter function
- *goals*: coverage pattern
- *constraints*: path pattern

“For each test goal inside scope, provide a test case which satisfies the passing constraints.”

Condition coverage in function *partition* with test cases that reach *line 7* at least once.

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer / filter function
- *goals*: coverage pattern
- *constraints*: path pattern

“For each test goal inside scope, provide a test case which satisfies the passing constraints.”

Condition coverage in function *partition* with test cases that reach *line 7* at least once.

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer / filter function
- *goals*: coverage pattern
- *constraints*: path pattern

“For each test goal inside scope, provide a test case which satisfies the passing constraints.”

Condition coverage in function *partition* with test cases that reach line 7 at least once.

# FQL Queries

**in scope cover goals passing constraints**

- *scope*: CFA transformer / filter function
- *goals*: coverage pattern
- *constraints*: path pattern

“For each test goal inside scope, provide a test case which satisfies the passing constraints.”

```
in @func(partition) cover @conditionedge passing @7
```

Condition coverage in function *partition* with test cases that reach line 7 at least once.

# Standard Coverage Criteria

```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

cover @basicblockentry

## Basic Block Coverage

Requires a test suite such that each basic block is executed at least once

# Standard Coverage Criteria

```
1 int partition(int a[], int left, int right) {  
2     int v = a[right], i = left - 1, j = right, t;  
3     for (;;) {  
4         while (a[++i] < v) ;  
5         while (j > left && a[--j] > v) ;  
6         if (i >= j) break;  
7         t = a[i]; a[i] = a[j]; a[j] = t;  
8     }  
9     t = a[i]; a[i] = a[right]; a[right] = t;  
10    return i;  
11 }
```

cover @basicblockentry

## Basic Block Coverage

Requires a test suite such that each basic block is executed at least once

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

cover @decisionedge

## Decision Coverage

Conditional statements evaluate to true / false

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a) | (b) && (c))
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

cover @conditionedge

## Condition Coverage

Atomic conditions evaluate to true/false

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a) | (b) && (c))
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

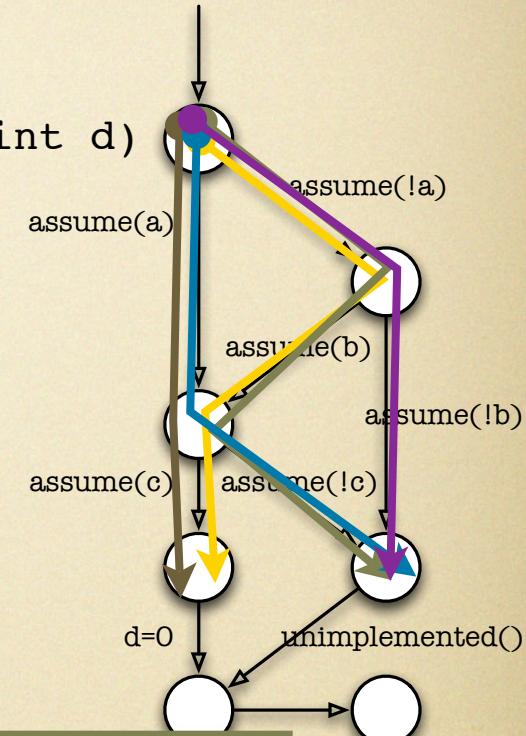
**cover @conditionedge+@decisionedge**

## Condition/Decision Coverage

Union of condition and decision coverage

# Standard Coverage Criteria

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a) | (b) && (c))
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```



```
cover paths (@conditiongraph, 1)
```

## Multiple Condition Coverage

All Boolean combinations of atomic conditions in complex conditions

# Standard Coverage Criteria

```
1  int example(int a, int d)
2  {
3      if (a)
4          d = 0;
5      else
6          unimplemented();
7      return d*2;
8  }
```

```
cover @def(d) ."not(@def(d)) *".@use(d)
```

## Def-Use Coverage

Cover all definition/use pairs of a variable  $d$

# Beyond Standard Criteria

```
1 void insert(int * a, int pos) {  
2     ...  
3 }  
4  
5 int main(int argc, char * argv[ ]) {  
6     int i;  
7     int A[100];  
8  
9     for (i=0; i<100; ++i)  
10    insert(A, i);  
11  
12    return 0;  
13 }
```

```
cover paths(@func(main) | @func(insert),2)
```

## Loop-Bounded Path Coverage

- All paths through *main* and *insert*
- Pass each statement two times

# Beyond Standard Criteria

```
1 int partition(int a[], int left, int right) {  
2     int v = a[right], i = left - 1, j = right, t;  
3     for (;;) {  
4         while (a[++i] < v) ;  
5         while (j > left && a[--j] > v) ;  
6         if (i >= j) break;  
7         t = a[i]; a[i] = a[j]; a[j] = t;  
8     }  
9     t = a[i]; a[i] = a[right]; a[right] = t;  
10    return i;
```

cover<sup>11</sup>@basicblockentry->@basicblockentry+  
@basicblockentry->@basicblockentry->  
@basicblockentry

## Cartesian Block Coverage

All pairs and triples of basic blocks in function *partition*

# Specs for Working Programmers

```
1 int example(int a, int b, int c, int d)
2 {
3     if ((a || b) && c)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

cover @4+@6

Cover Specific Lines of Code

Lines 4 and 6

# Specs for Working Programmers

```
1 void eval(int * a, int first, int last) {  
2     if (first > last) return;  
3     printf("a[%d]=%d\n", first, a[first]);  
4     eval(a+1, first+1, last);  
5     return;  
6 }  
7  
8 int main(int argc, char * argv[ ]) {  
9     int i;  
10    int A[100];  
11  
12    for (i=0; i<100; ++i)  
13        insert(A, i);  
14  
15    eval(A, 0, 100);  
16    return 0;  
17}
```

**cover @basicblockentry & @func(eval)**

## Restricted Scope of Analysis

Basic block coverage in function *eval* only

# Specs for Working Programmers

```
1 void sort(int * a, int len) {  
2     int i, t;  
3     for (i=1; i<len; ++i) {  
4         if (compare(a[i-1], a[i])) continue;  
5         t=a[i]; a[i]=a[i-1]; a[i-1]=t;  
6     }  
7 }  
8  
9 void eval(int * a, int len) {  
10    int i;  
11    for (i=0; i < 3; ++i)  
12        printf("a[%d]=%d\n", i, a[i]);  
13 }  
14  
15 int main(int argc, char * argv[]) {  
16    int i; int A[100];  
17    for (i=0; i < 100; ++i) sort(A, 100);  
18    eval(A, 100);  
19 }  
20 }
```

cover (@conditionedge & @func(sort)) ->  
(@basicblockentry & @func(eval))

## Interaction Coverage

Cover all pairs of conditions in *sort* and basic blocks in *eval*

# Specs for Working Programmers

```
1 void sort(int * a, int len) {  
2     int i, t;  
3     for (i=1; i<len; ++i) {  
4         if (compare(a[i-1], a[i])) continue;  
5         t=a[i];  
6         a[i]=a[i-1];  
7         a[i-1]=t;  
8     }  
9     return;  
10 }
```

```
cover {len>=2} . {len<=15} .  
@basicblockentry & @func(sort)
```

## Constrained Inputs

- Basic block coverage in *sort*
- Each test case shall use list of 2 to 15 elements

# Specs for Working Programmers

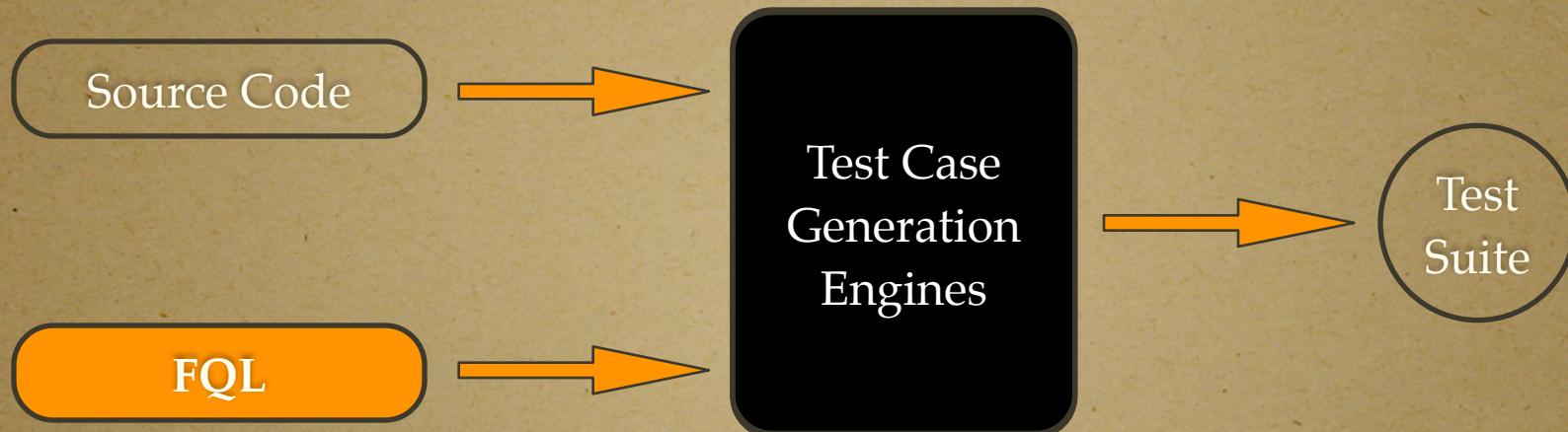
```
1 int example(int a, int d)
2 {
3     if (a)
4         d = 0;
5     else
6         unimplemented();
7     return d*2;
8 }
```

```
cover @basicblockentry & @func(example)
passing ^not(@call(unimplemented)) *$
```

## Avoid Unfinished Code

- Basic block coverage in *example*
- Never call *unimplemented*

# Conclusions and Current Work



- Specification language based on quoted regular expressions to precisely specify where to go
- Two test input generation engines
- Currently evaluating with partners from automotive and avionics industry
- Extending our work to model-based testing



**<http://code.forsyte.de/fshell>**

<http://code.forsyte.de/fshell>

# Experimental Results

Source	SLOC	BB (Q1)		CC (Q2)		BB <sup>2</sup> (Q18)		
		#goals	#tc	#goals	#tc	#goals	#tc	#inf
coreutils/cat.c	27	16	4	10	4	224	6	39
coreutils/echo.c	161	27	8	20	11	675	93	198
coreutils/nohup.c	33	17	6	12	5	255	13	133
coreutils/seq.c	37	28	7	20	7	728	23	394
coreutils/tee.c	73	21	9	16	8	399	30	127
kbfiltr.c	3507	250	2	196	2	62000	2	61911
pseudo-vfs.c	553	10	3	6	3	80	3	44
matlab.c	3444	30	6	22	6	840	10	441
memman.c	245	53	8	40	8	2756	29	1749
PicoSAT	6592	191	43	153	39	36099	417	26352