FQL: A Query Language for Program Testing

for(syte)

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

White Box Testing Targets

TECHNISCHE UNIVERSITÄT DARMSTADT

- Structural coverage (basic block, condition, decision, paths)
- Data flow coverage (def-use pairs)
- Variable valuations

- Specific program executions
- Any of these within a fragment of the program

- Independent of test case generation technique

White Box Testing Targets

- Structural c[…]ths)
- Data flow c[…]
- Variable va[…]

- Informal
- No pre-existing formalism to describe all these coverage criteria
- Coverage analysis tools apply contradictory definitions

- Specific program executions
- Any of these within a fragment of the program

- Independent of test case generation technique

White Box Testing Targets

TECHNISCHE UNIVERSITÄT DARMSTADT

- Informal
- No pre-existing formalism to describe all these coverage criteria
- Coverage analysis tools apply contradictory definitions

- Structural c … ths)
- Data flow c
- Variable va
- Specific program exec

- Ad-hoc solutions

- Any of these within a … program

- Independent of test case generation technique

White Box Testing Targets

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Structural coverage (basic block, condition, decision, paths)
- Data flow coverage (def-use pairs)
- Variable valuations
- Speci
- Any of ... the program
- Independent of test case generation technique

- Informal
- No pre-existing formalism to describe all these coverage criteria
- Coverage

- Database analogue!
- Easy-to-use query language building on mathematical core
- Efficient backend (CAV'08, VMCAI'09)

Related Work

TECHNISCHE UNIVERSITÄT DARMSTADT

Related Work

TECHNISCHE UNIVERSITÄT DARMSTADT

- Coverage formalizations
  - Using Z (Vilkomir, Bowen FORTEST’08)
  - Temporal logics (Hong et al. TACAS’02)

Related Work

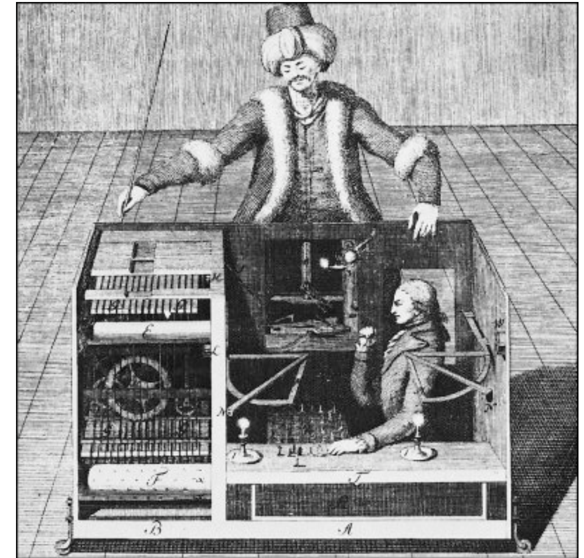
TECHNISCHE UNIVERSITÄT DARMSTADT

- Coverage formalizations
  - Using Z (Vilkomir, Bowen FORTEST'08)
  - Temporal logics (Hong et al. TACAS'02)
- Test case generation techniques
  - Random testing, fuzz testing
  - Directed testing (Godefroid et al. PLDI'05)
  - Model-checking based (Beyer et al. ICSE'04)

Related Work

TECHNISCHE UNIVERSITÄT DARMSTADT

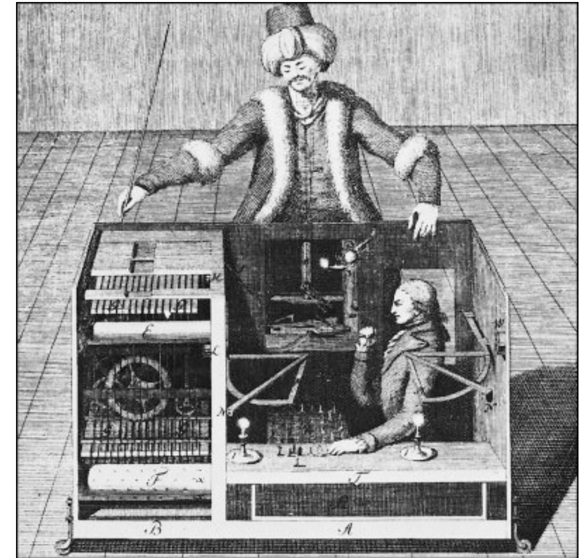
- Coverage formalizations
  - Using Z (Vilkomir, Bowen FORTEST'08)
  - Temporal logics (Hong et al. TACAS'02)
- Test case generation techniques
  - Random testing, fuzz testing
  - Directed testing (Godefroid et al. PLDI'05)
  - Model-checking based (Beyer et al. ICSE'04)



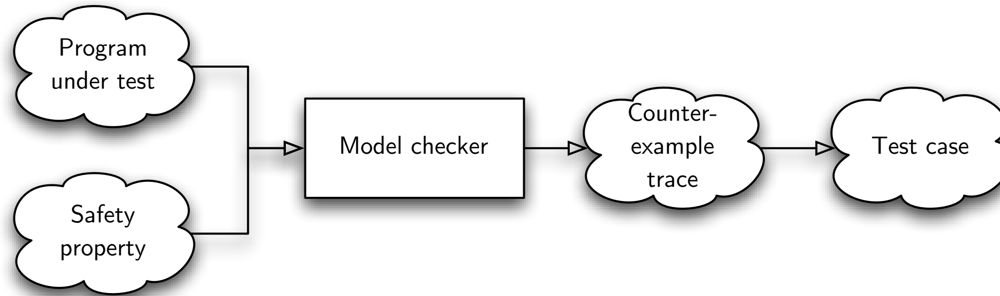
Related Work

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Coverage formalizations
  - Using Z (Vilkomir, Bowen FORTEST'08)
  - Temporal logics (Hong et al. TACAS'02)
- Test case generation techniques
  - Random testing, fuzz testing
  - Directed testing (Godefroid et al. PLDI'05)
  - Model-checking based (Beyer et al. ICSE'04)
- BLAST query language (Beyer et al. SAS'04)

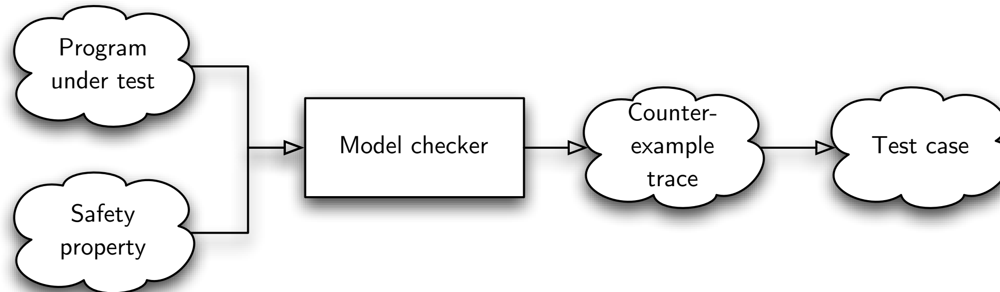


Test Case Generation Using Model Checkers



Test Case Generation Using Model Checkers

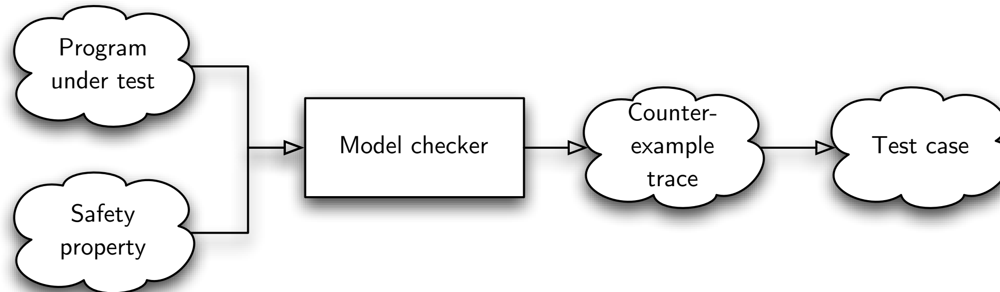
TECHNISCHE UNIVERSITÄT DARMSTADT



- Naive approach requires manual annotation of assertions

Test Case Generation Using Model Checkers

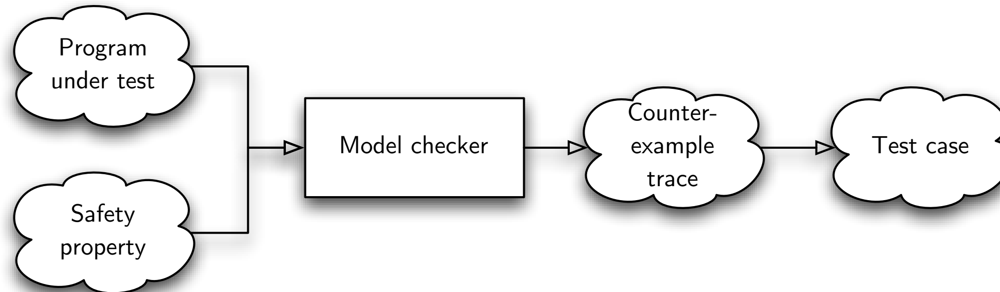
TECHNISCHE
UNIVERSITÄT
DARMSTADT



- Naive approach requires manual annotation of assertions
- MC neither tailored for coverage nor efficient enumeration

Test Case Generation Using Model Checkers

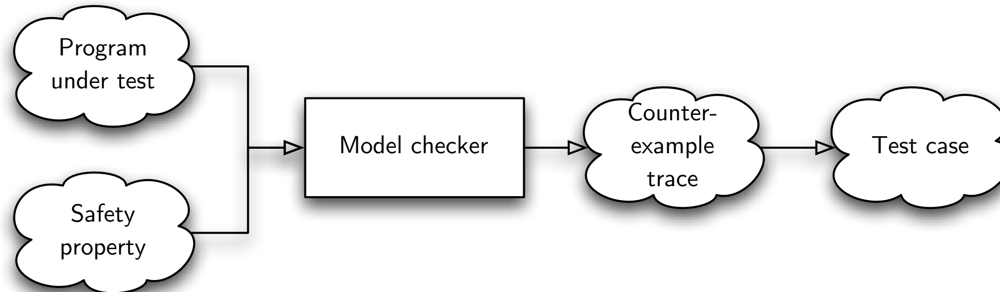
TECHNISCHE UNIVERSITÄT DARMSTADT



- Naive approach requires manual annotation of assertions
- MC neither tailored for coverage nor efficient enumeration
- BLAST 2.0:
  - First automated test suite generation using model checkers
  - Basic block coverage hard coded

Test Case Generation Using Model Checkers

TECHNISCHE UNIVERSITÄT DARMSTADT

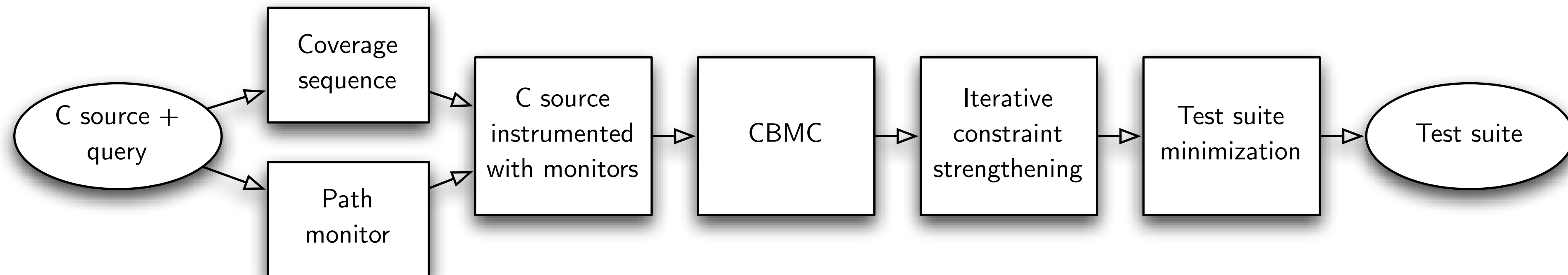


- Naive approach requires manual annotation of assertions
- MC neither tailored for coverage nor efficient enumeration
- BLAST 2.0:
  - First automated test suite generation using model checkers
  - Basic block coverage hard coded
- BLAST query language
  - Convenient specification of reachability queries
  - Description of paths requires observer automata (distinct formalism)
  - Path coverage cannot be expressed easily

FQL: FShell Query Language

TECHNISCHE UNIVERSITÄT DARMSTADT

- FShell: Test case generation based on CBMC code
- Shell-like interface to state FQL queries
- Full ANSI-C support
- Efficient SAT-based enumeration of test cases using incremental constraint strengthening (VMCAI'09)

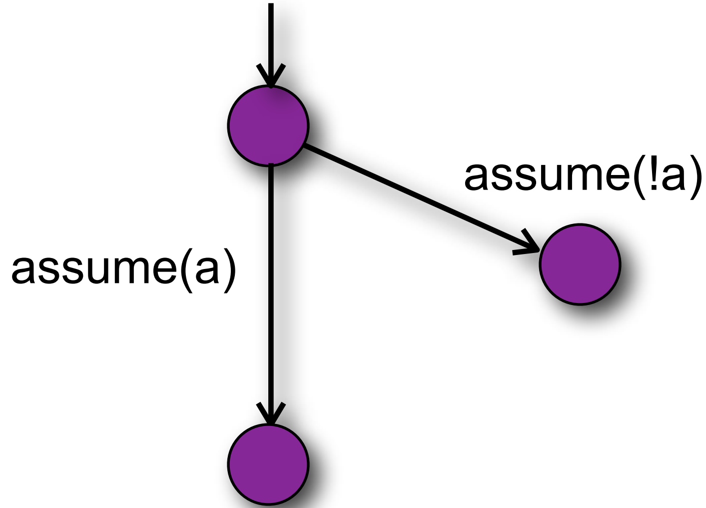


Program Representation: Control Flow Automaton (CFA)

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

Program Representation: Control Flow Automaton (CFA)

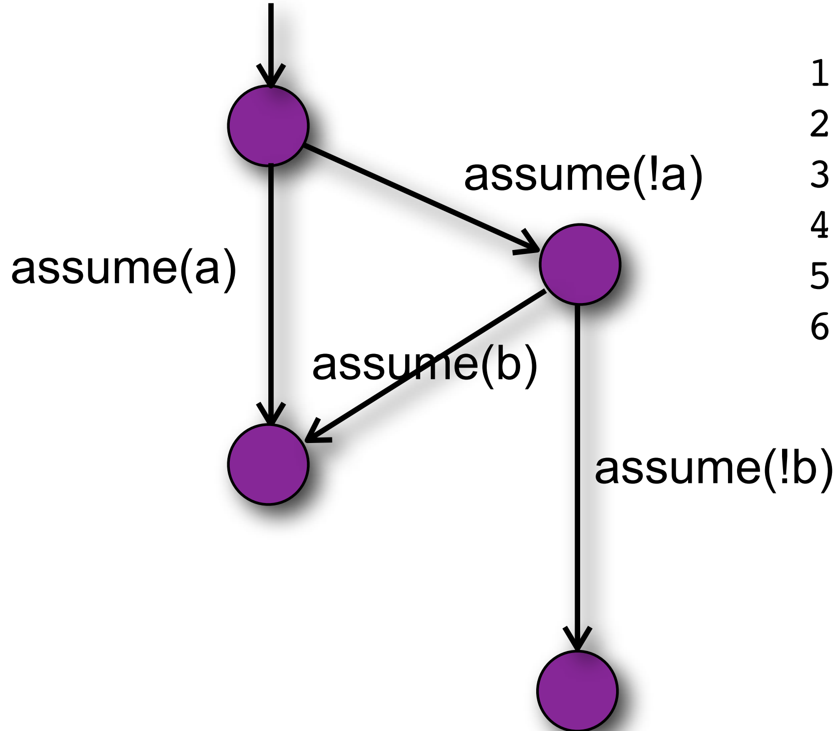
TECHNISCHE UNIVERSITÄT DARMSTADT




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

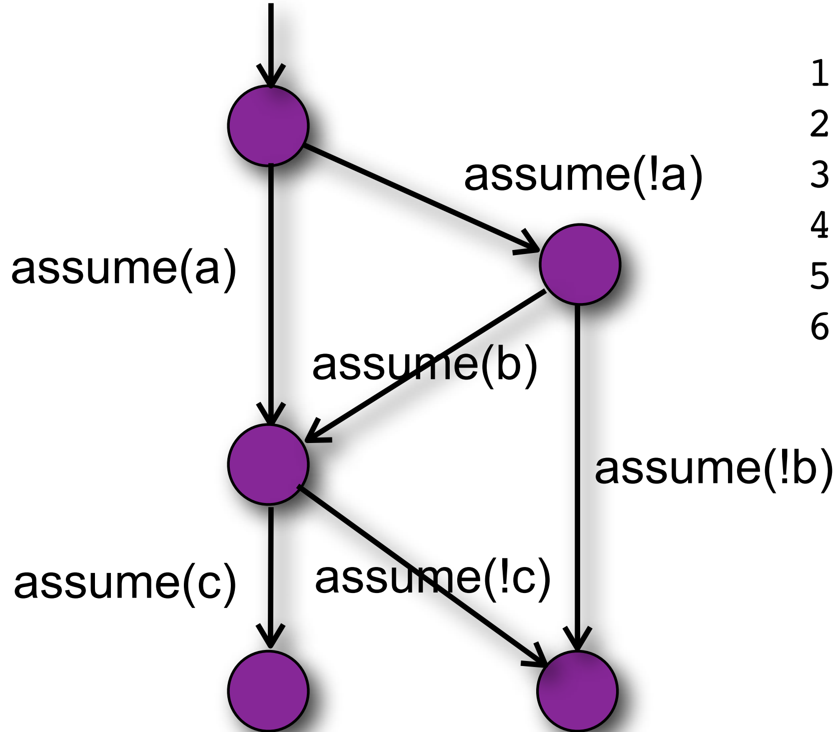
Program Representation: Control Flow Automaton (CFA)

TECHNISCHE UNIVERSITÄT DARMSTADT




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

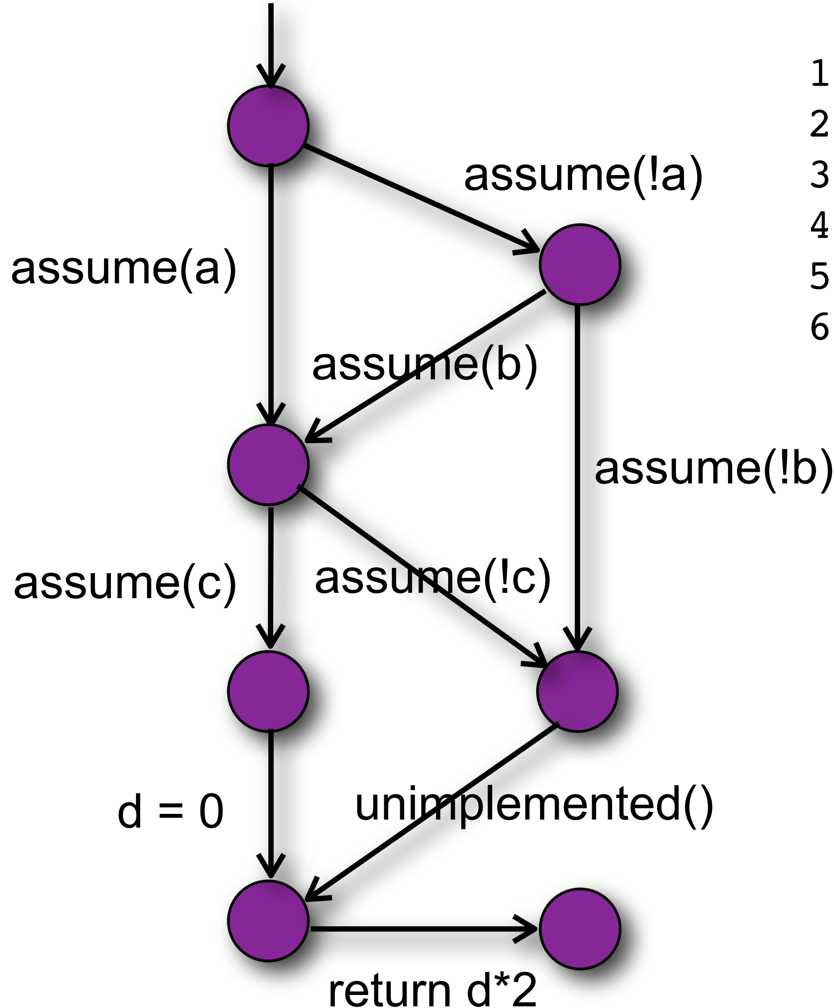
Program Representation: Control Flow Automaton (CFA)




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

Program Representation: Control Flow Automaton (CFA)

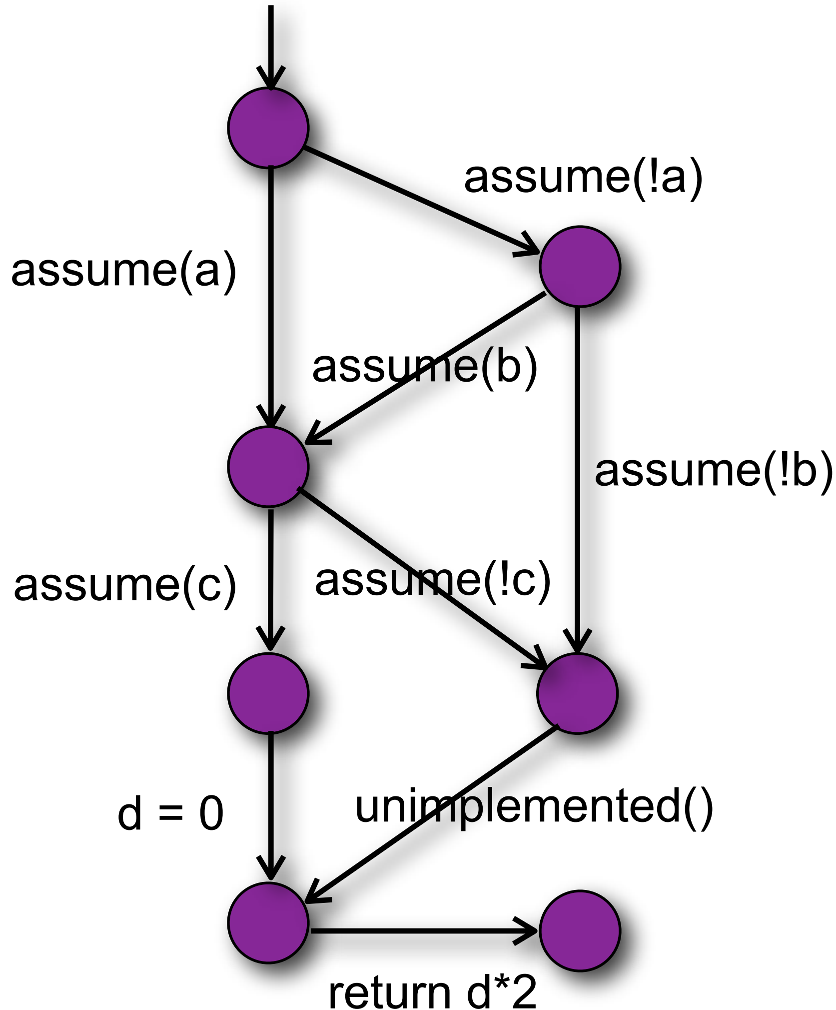
TECHNISCHE UNIVERSITÄT DARMSTADT



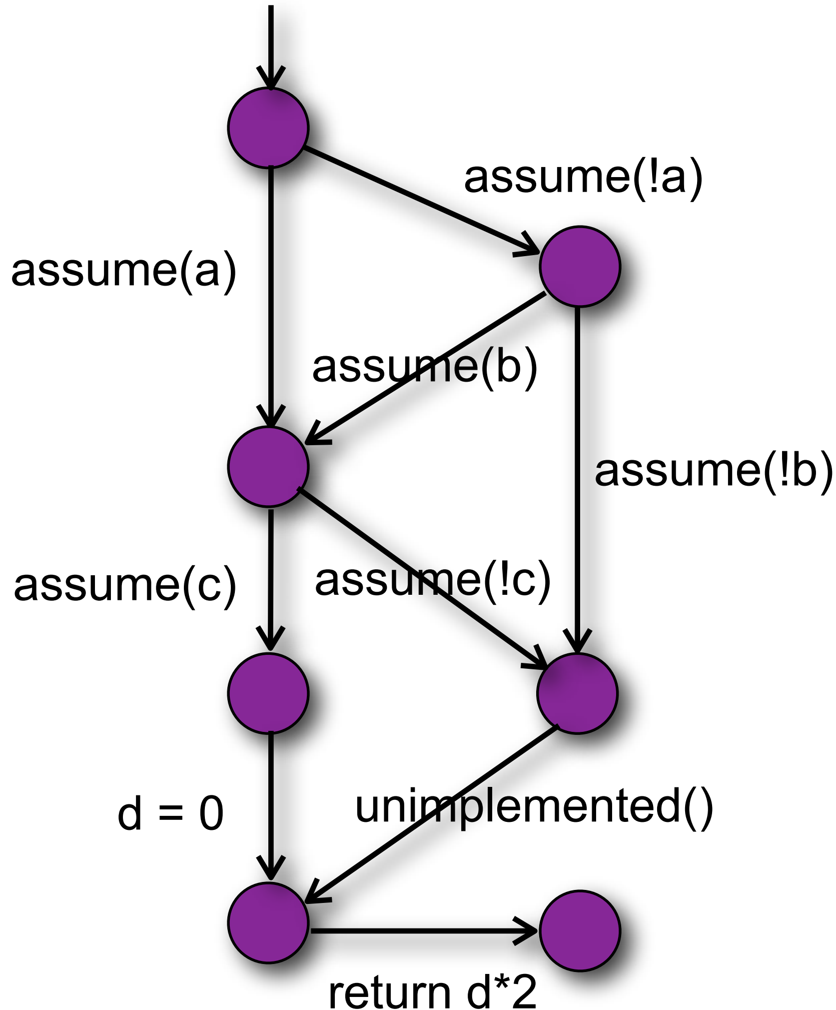

```
1   int test(int a, int b, int c, int d)
2   {
3     if ((a || b) && c) d = 0;
4     else unimplemented();
5     return d*2;
6   }
```

Filter Functions and Target Graphs

TECHNISCHE
UNIVERSITÄT
DARMSTADT

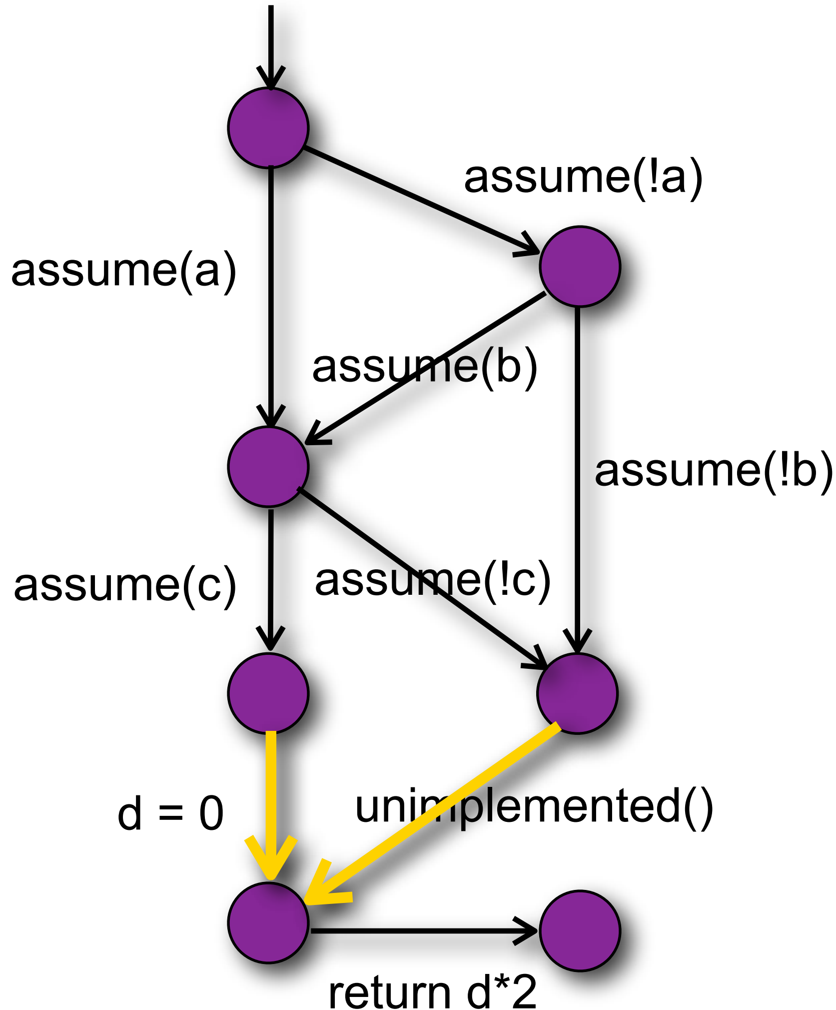


Filter Functions and Target Graphs



- Testing targets

Filter Functions and Target Graphs

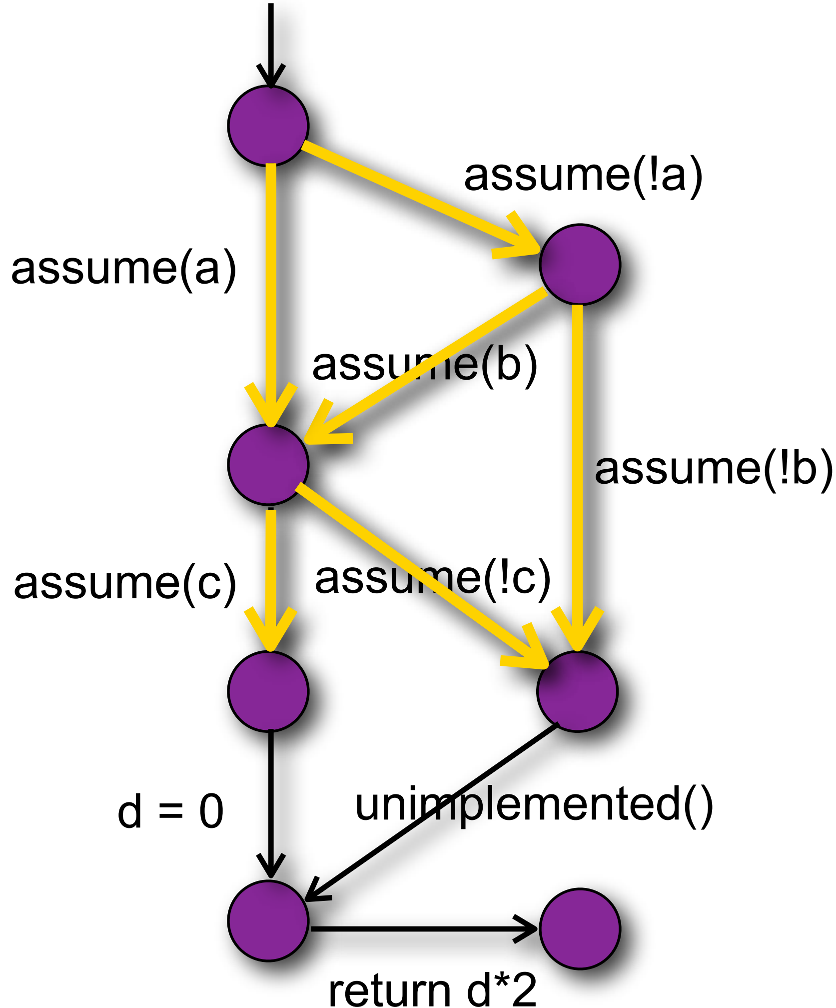


- Testing targets
  - cover decision edges

    `@decisionedge`

TECHNISCHE UNIVERSITÄT DARMSTADT
for(syte)

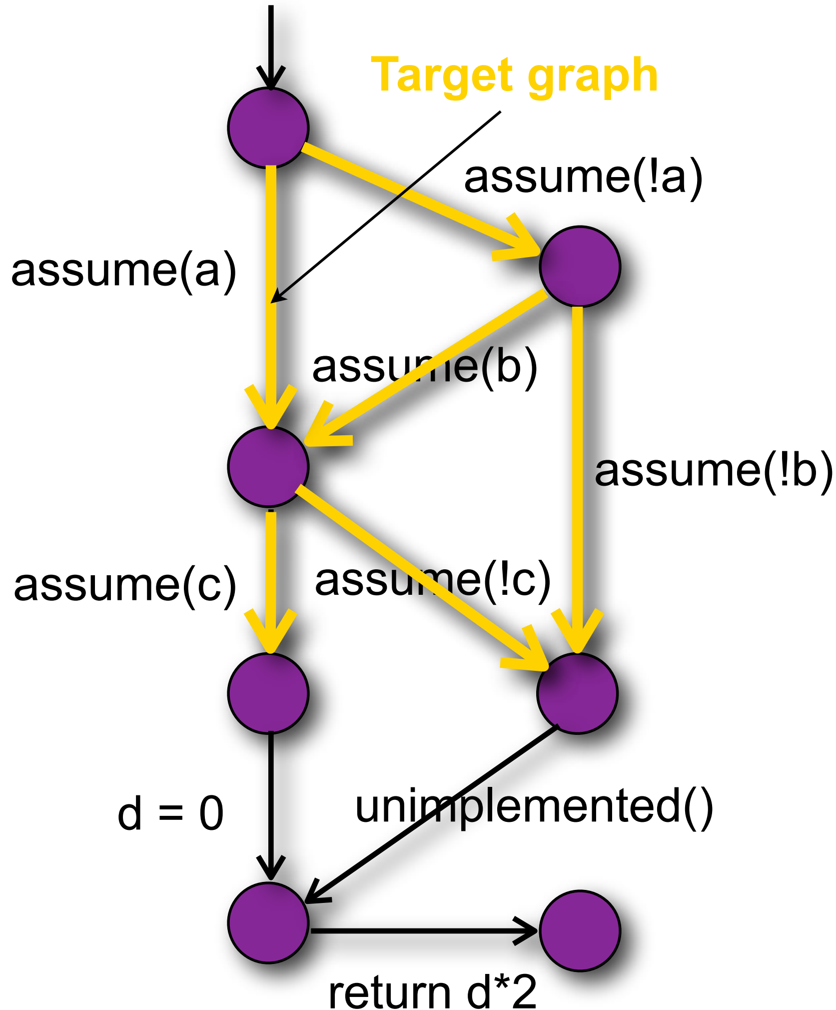
Filter Functions and Target Graphs



- Testing targets
  - cover decision edges
    `@decisionedge`
  - cover condition edges
    `@conditionedge`
    `@conditiongraph`

Filter Functions and Target Graphs

TECHNISCHE UNIVERSITÄT DARMSTADT

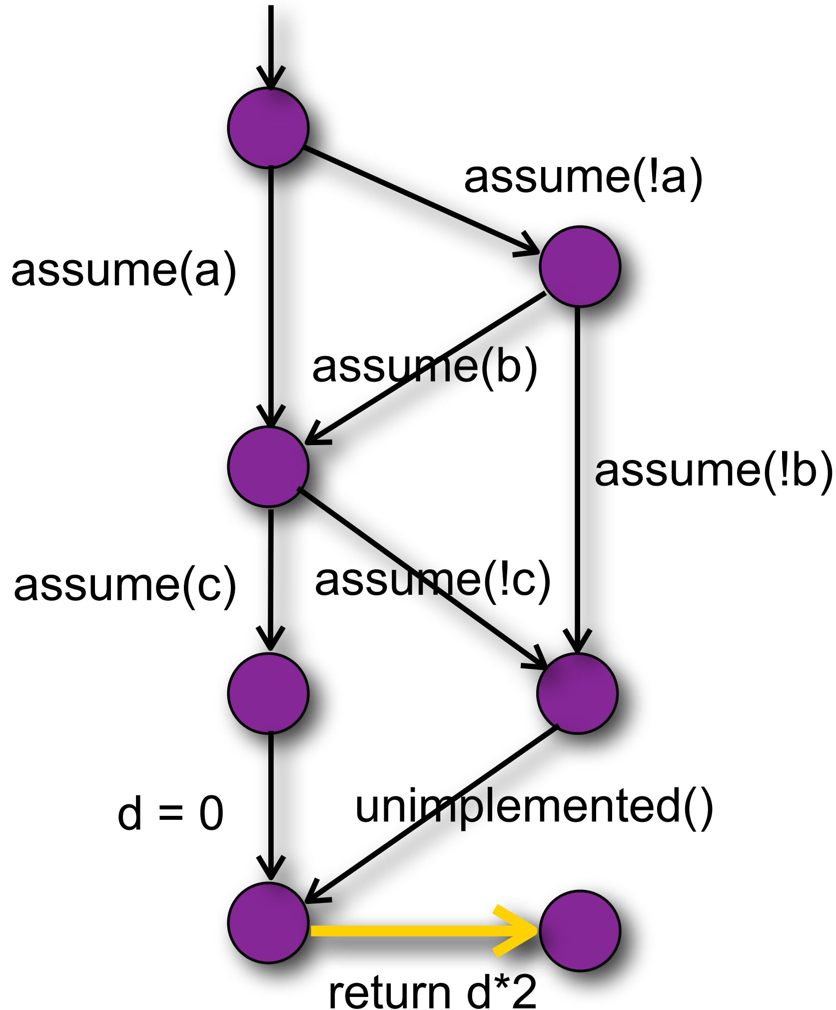


- Testing targets
  - cover decision edges
    `@decisionedge`
  - cover condition edges
    `@conditionedge`
    `@conditiongraph`

Filter function

Filter Functions and Target Graphs

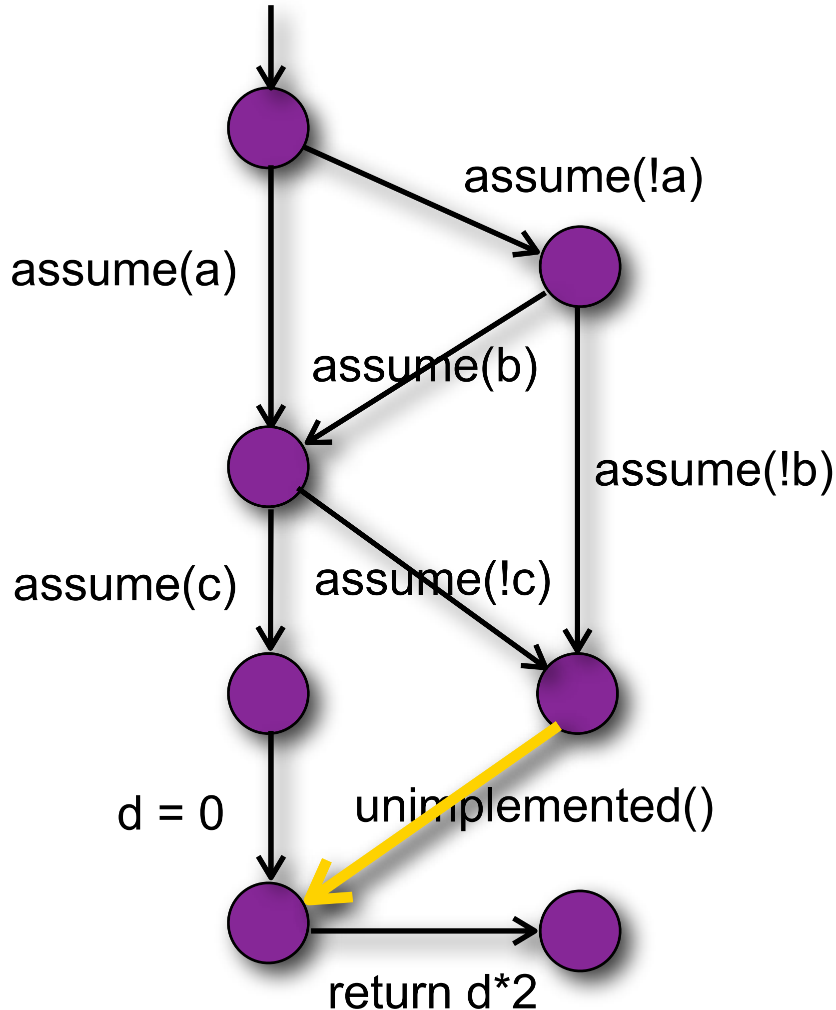
TECHNISCHE UNIVERSITÄT DARMSTADT



- Testing targets
  - cover decision edges
    `@decisionedge`
  - cover condition edges
    `@conditionedge`
    `@conditiongraph`
  - cover line 5
    `@line(5)` or `@5`

Filter Functions and Target Graphs

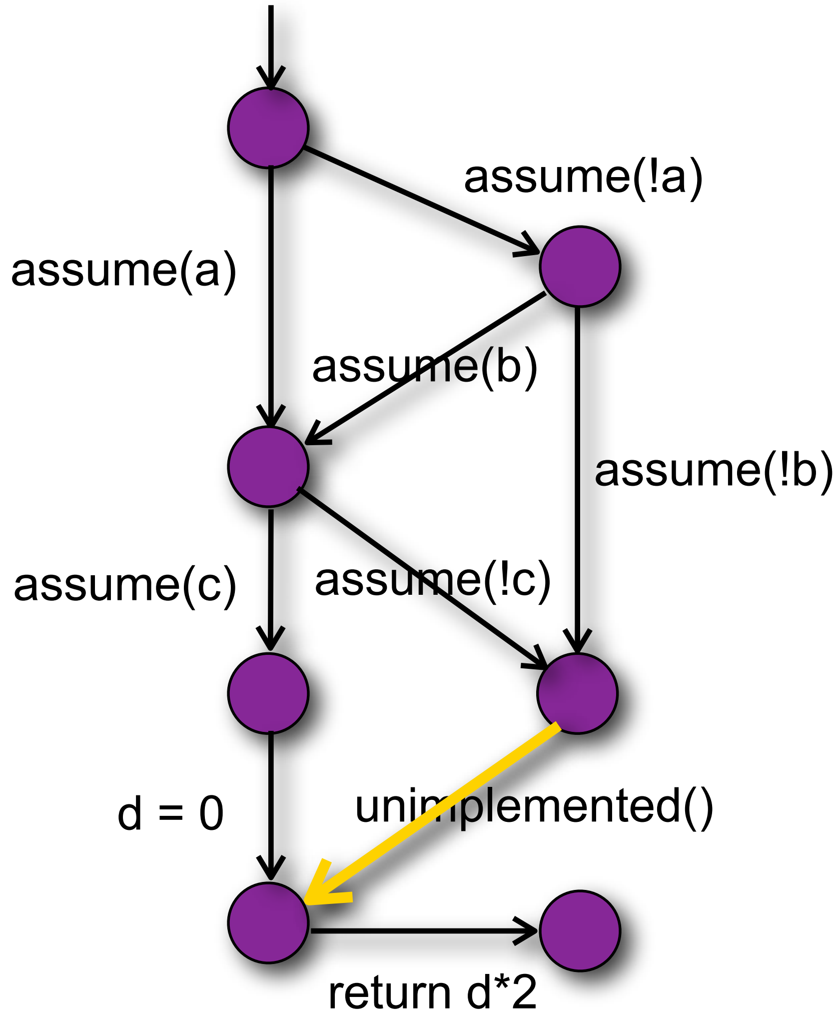
TECHNISCHE UNIVERSITÄT DARMSTADT



- Testing targets
  - cover decision edges
    `@decisionedge`
  - cover condition edges
    `@conditionedge`
    `@conditiongraph`
  - cover line 5
    `@line(5)` or `@5`
  - cover function calls
    `@calls`
    `@call(unimplemented)`

Filter Functions and Target Graphs

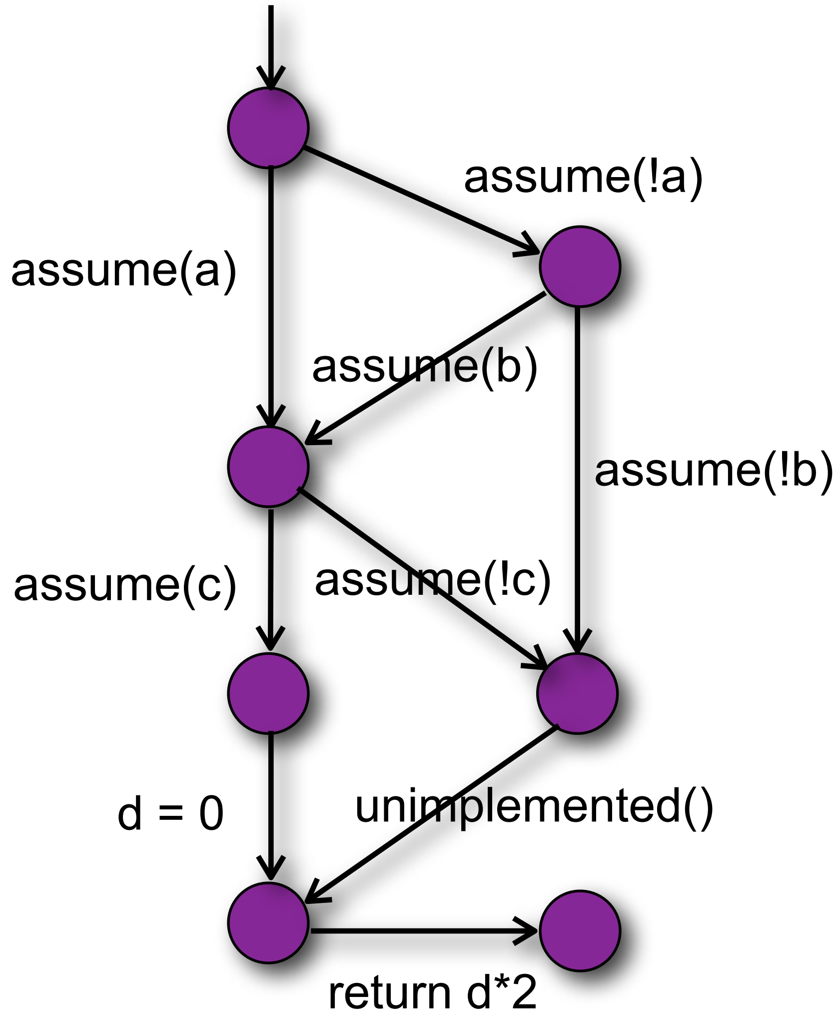
TECHNISCHE UNIVERSITÄT DARMSTADT



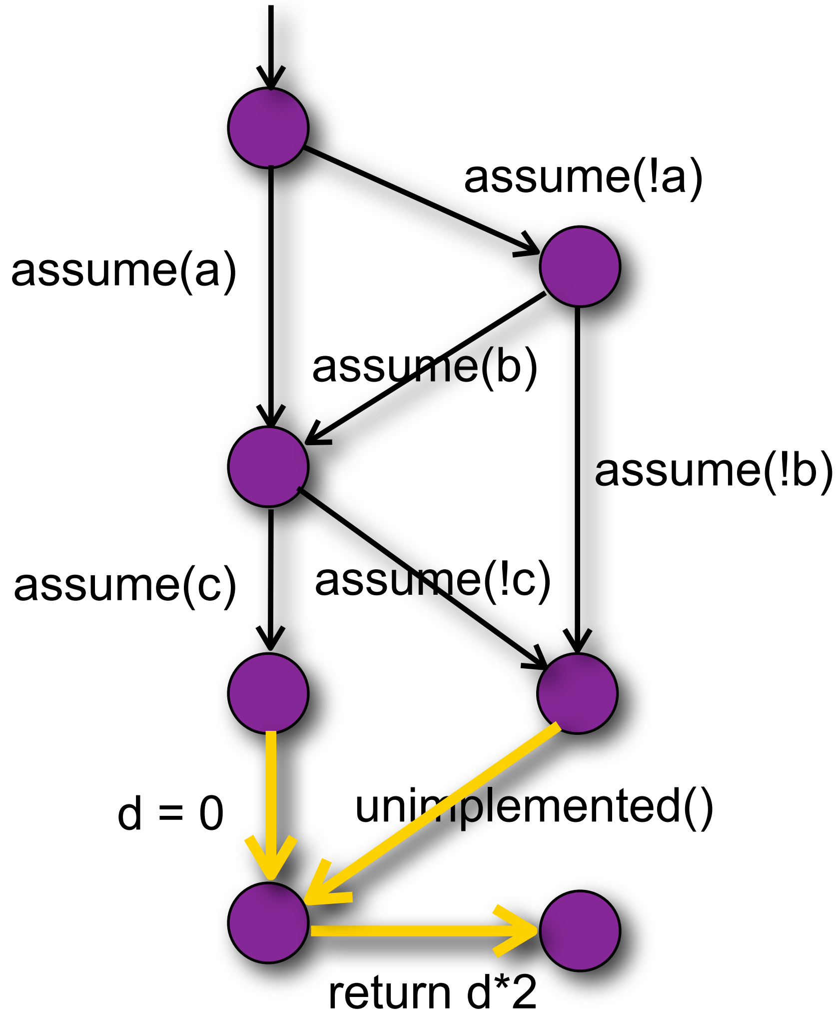
- Testing targets
  - cover decision edges
    `@decisionedge`
  - cover condition edges
    `@conditionedge`
    `@conditiongraph`
  - cover line 5
    `@line(5)` or `@5`
  - cover function calls
    `@calls`
    `@call(unimplemented)`
- Interface to programming language

Filter Functions and Target Graphs

TECHNISCHE UNIVERSITÄT DARMSTADT



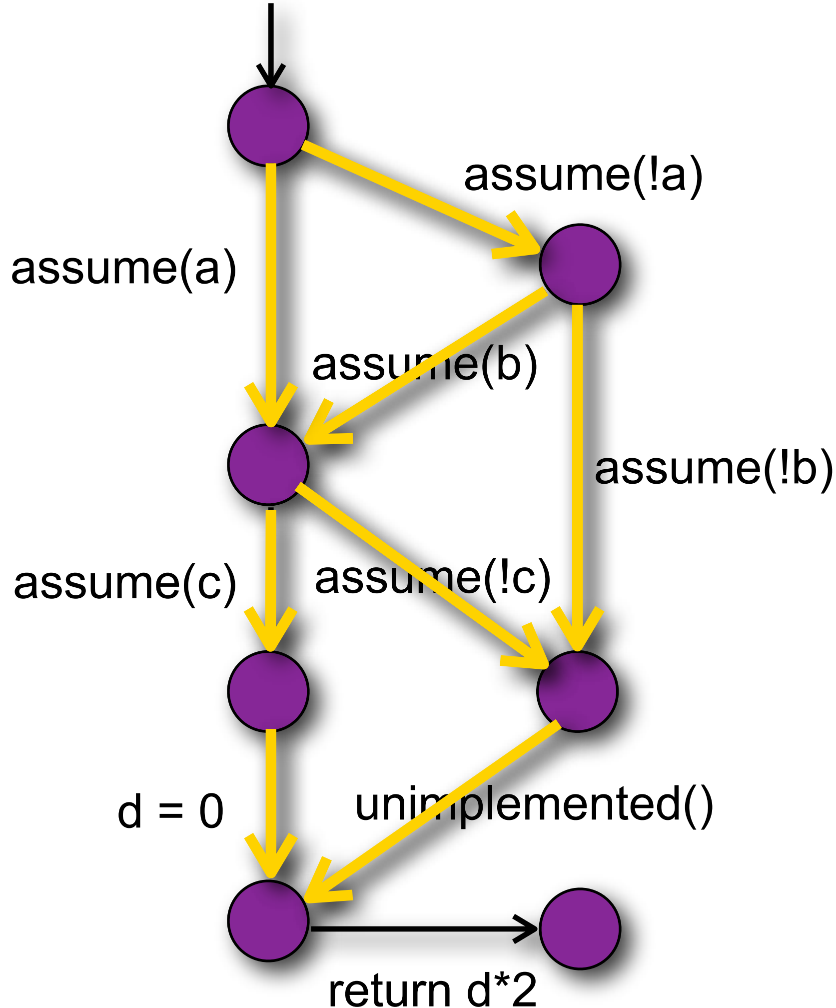
Filter Functions and Target Graphs



- `union(@5,@decisionedge)`

Filter Functions and Target Graphs

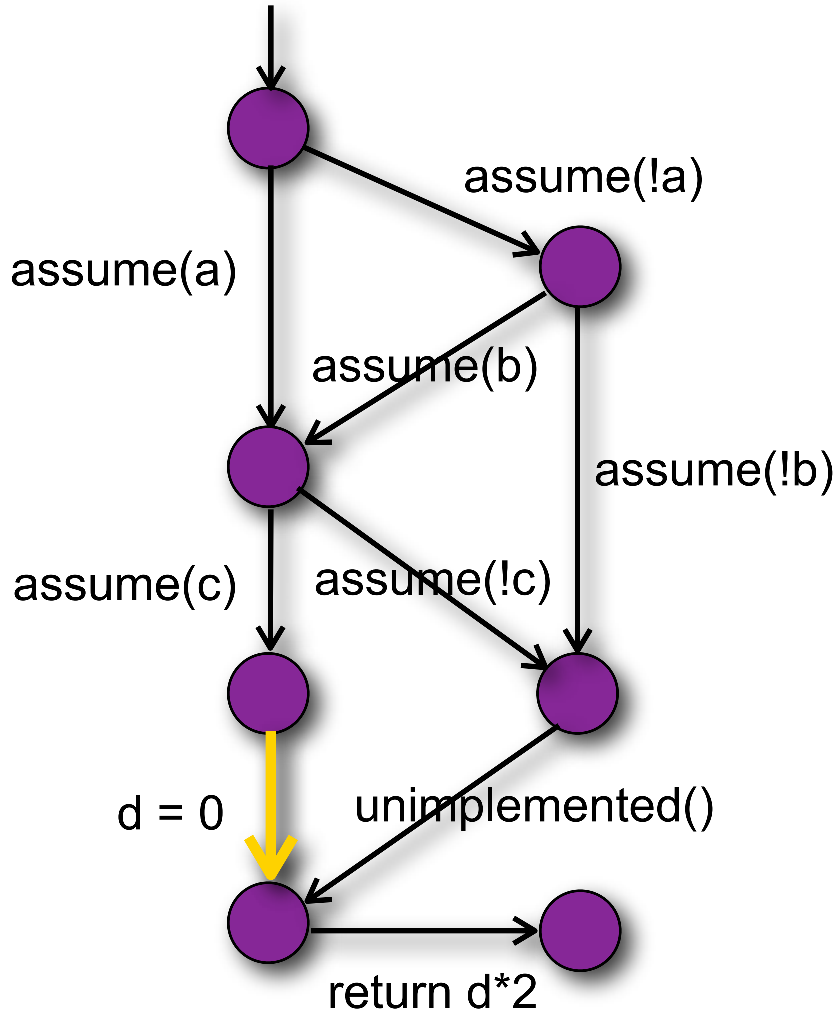
TECHNISCHE UNIVERSITÄT DARMSTADT



- `union(@5,@decisionedge)`
- `complement(@5)`

Filter Functions and Target Graphs

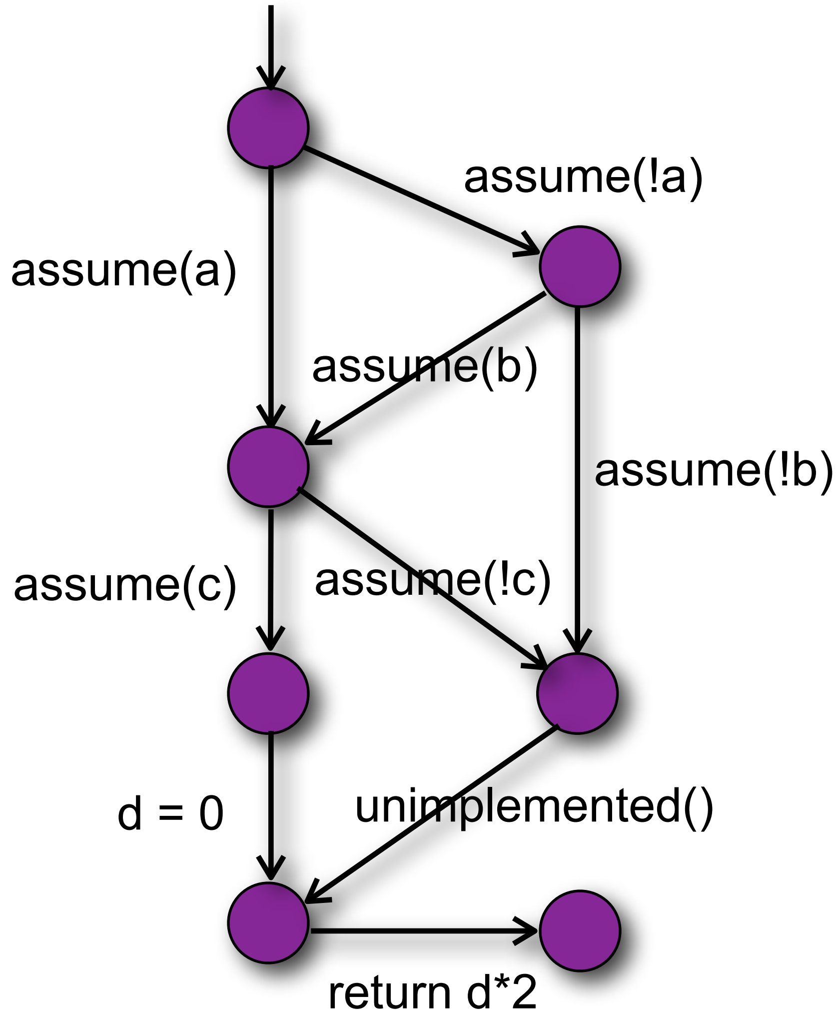
TECHNISCHE UNIVERSITÄT DARMSTADT



- `union(@5,@decisionedge)`
- `complement(@5)`
- `setminus(@decisionedge,@calls)`

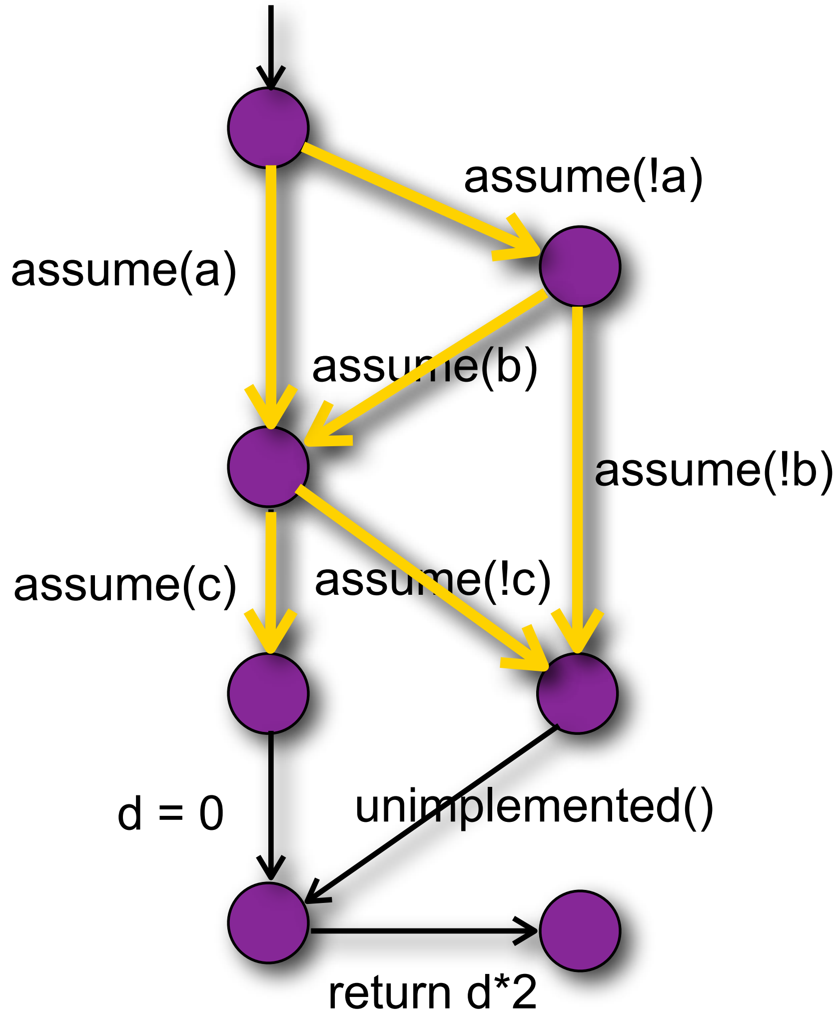
Filter Functions and Target Graphs

TECHNISCHE UNIVERSITÄT DARMSTADT

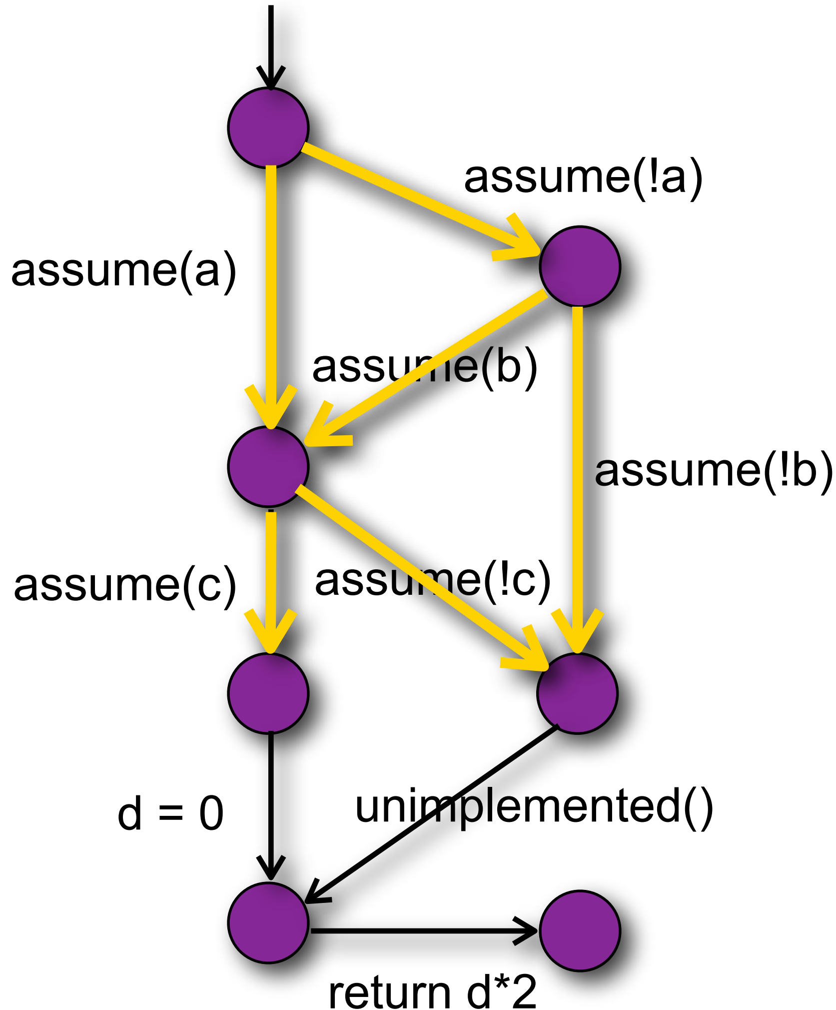


- `union(@5,@decisionedge)`
- `complement(@5)`
- `setminus(@decisionedge,@calls)`
- `intersect(@5,@conditionedge)`

Test Goals

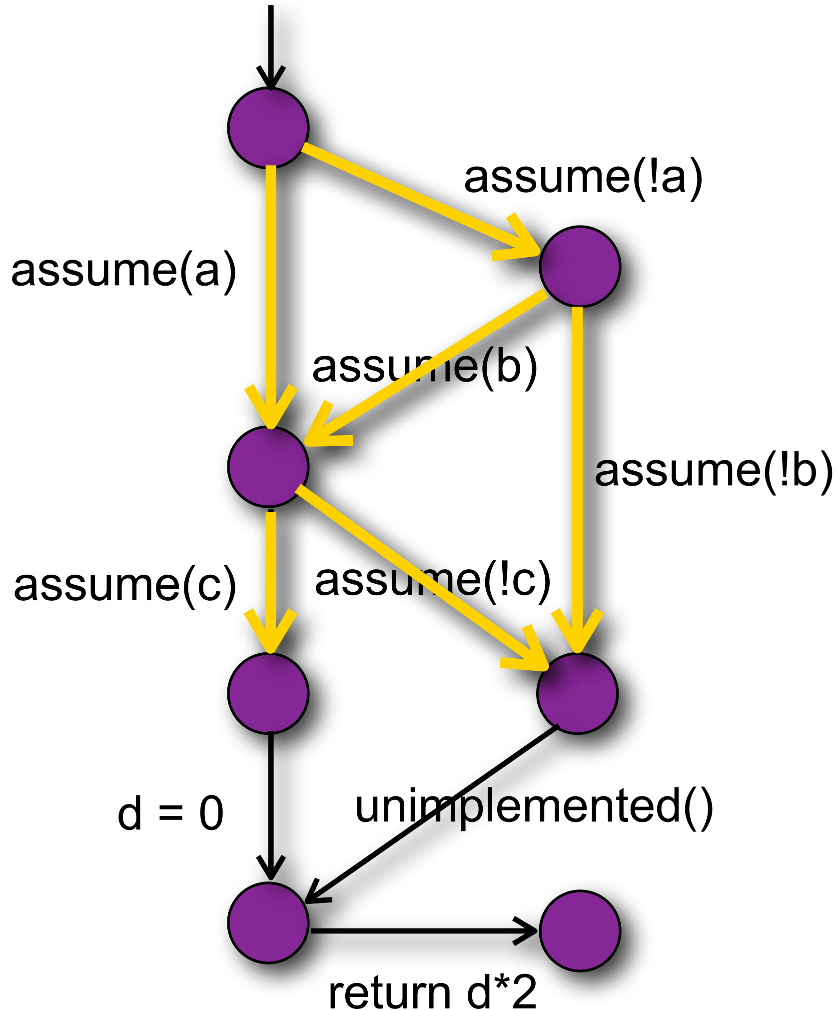


Test Goals



- `cover` **`STATES`**`(@conditiongraph)`
  - a = 1, c = 1
  - a = 0, b = 0

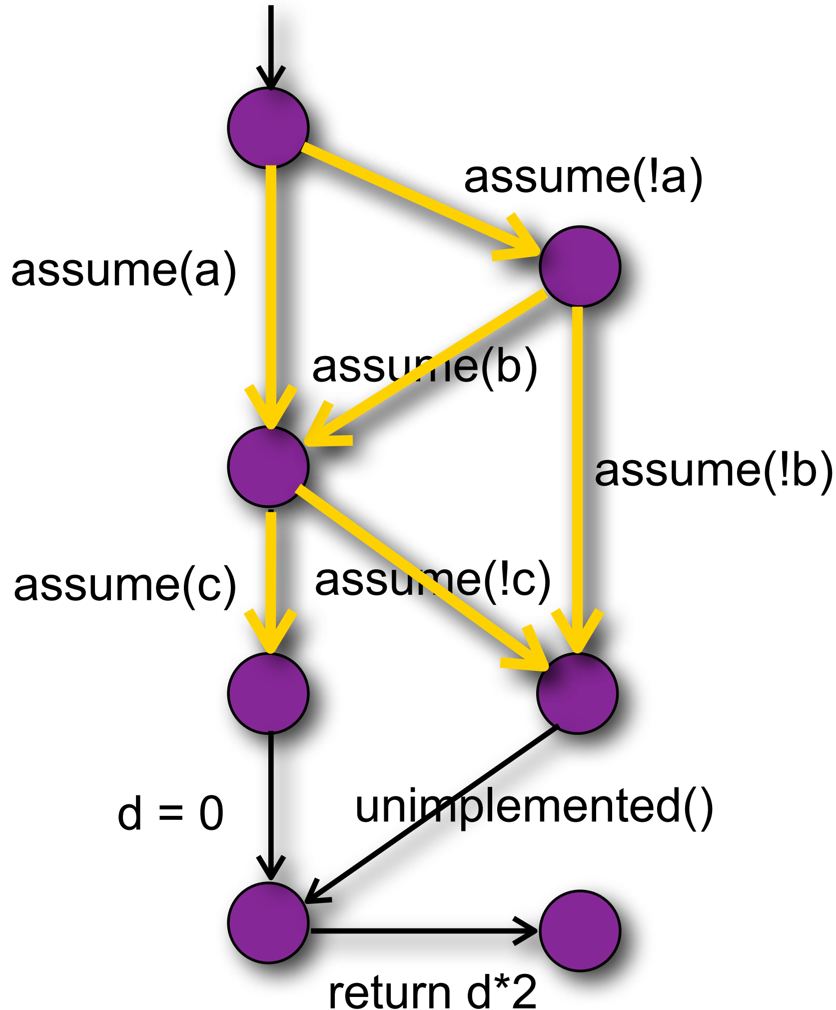
Test Goals



- `cover` **`STATES`**`(@conditiongraph)`
  - a = 1, c = 1
  - a = 0, b = 0
- `cover` **`EDGES`**`(@conditiongraph)`
  - a = 1, c = 1
  - a = 0, b = 0
  - a = 0, b = 1, c = 0

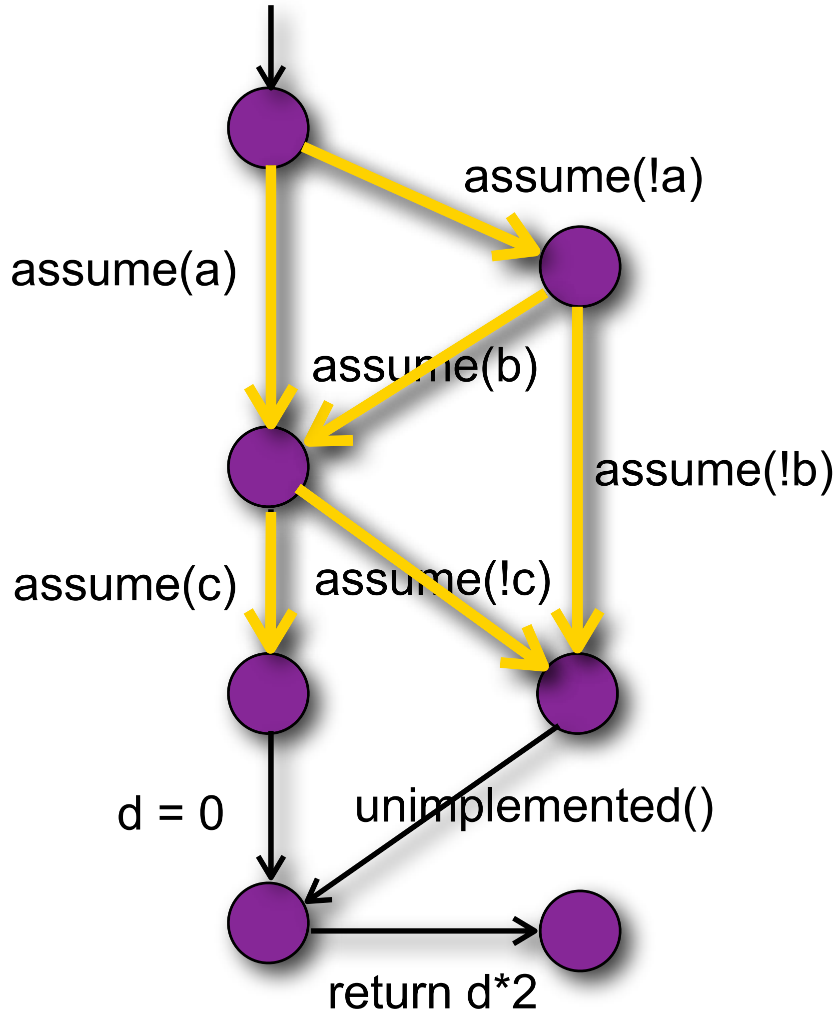
Test Goals

TECHNISCHE UNIVERSITÄT DARMSTADT



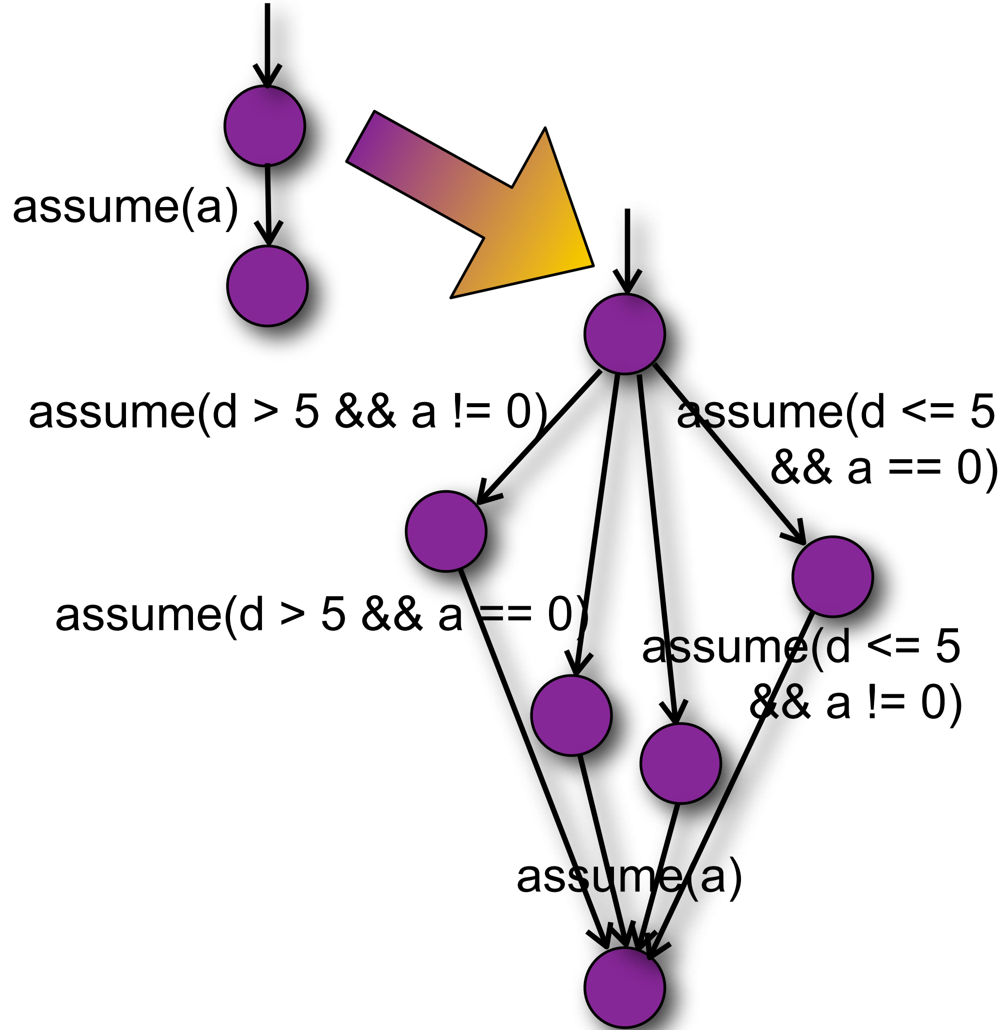
- `cover STATES(@conditiongraph)`
  - a = 1, c = 1
  - a = 0, b = 0
- `cover EDGES(@conditiongraph)`
  - a = 1, c = 1
  - a = 0, b = 0
  - a = 0, b = 1, c = 0
- `cover PATHS(@conditiongraph,1)`
  - a = 1, c = 1
  - a = 0, b = 0
  - a = 0, b = 1, c = 0
  - a = 0, b = 1, c = 1
  - a = 1, c = 0

Test Goals (continued)



- Test goals for MC/DC-style dependency coverage: `DEPS`

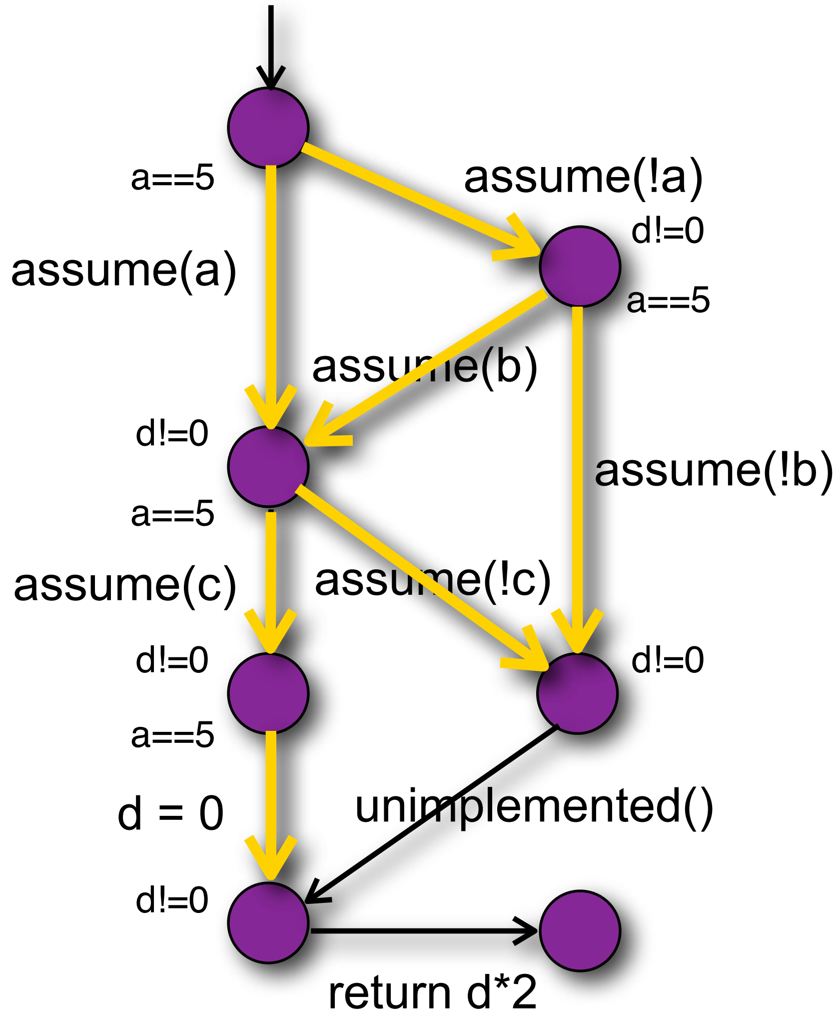
Test Goals (continued)



- Test goals for MC/DC-style dependency coverage: `DEPS`
- Predicated CFAs: `cover STATES(@conditiongraph, {d > 5}, {a != 0})`

Test Goals (continued)

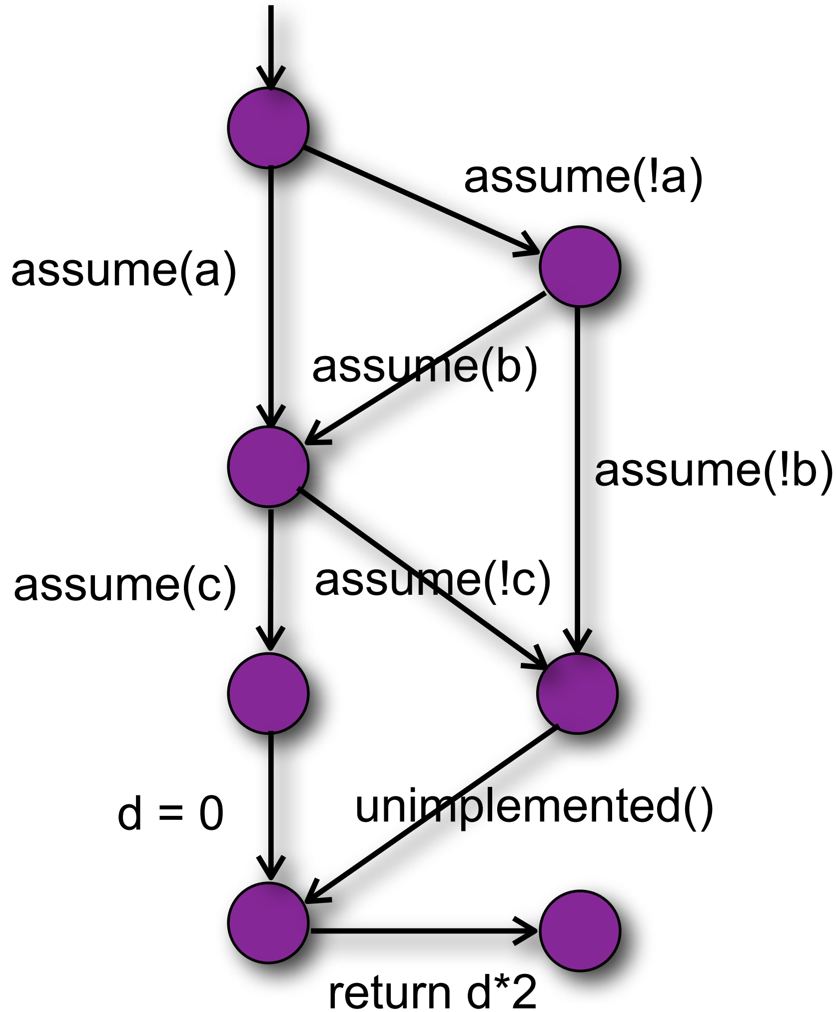
TECHNISCHE UNIVERSITÄT DARMSTADT



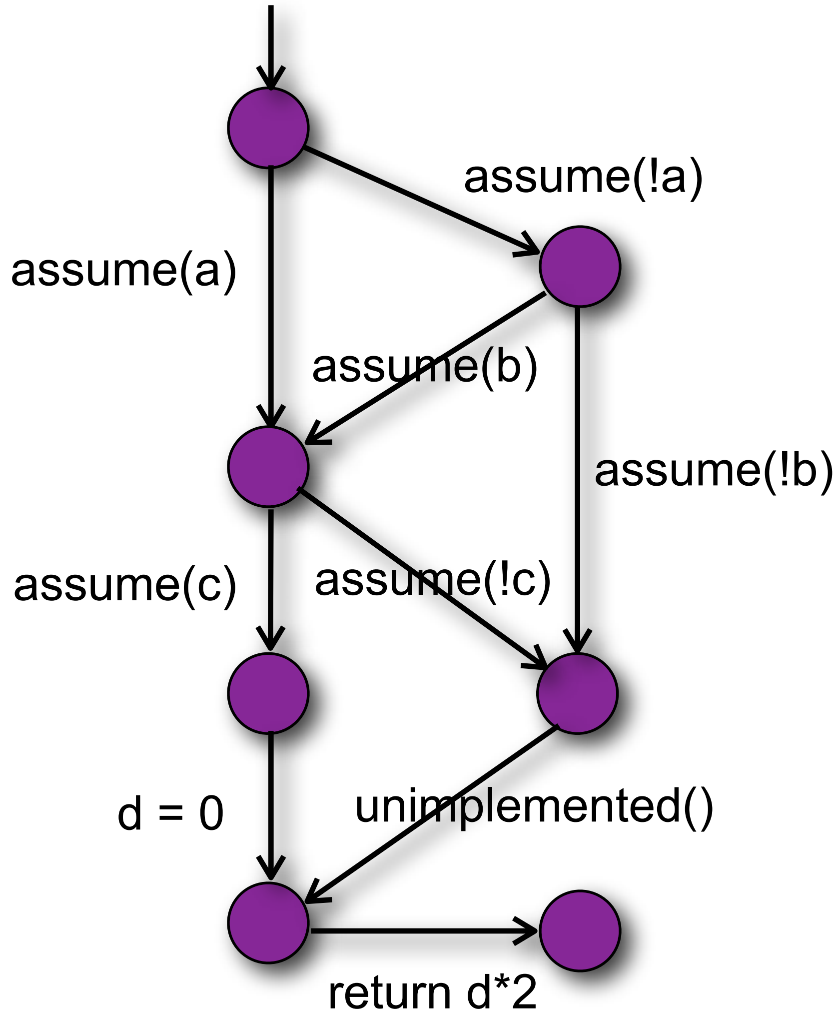
- Test goals for MC/DC-style dependency coverage: `DEPS`
- Predicated CFAs: `cover STATES(@conditiongraph, {d > 5}, {a != 0})`
- Pre- and post-conditions: `cover {a == 5}EDGES(@3) {d != 0}`

Path Monitors

TECHNISCHE UNIVERSITÄT DARMSTADT

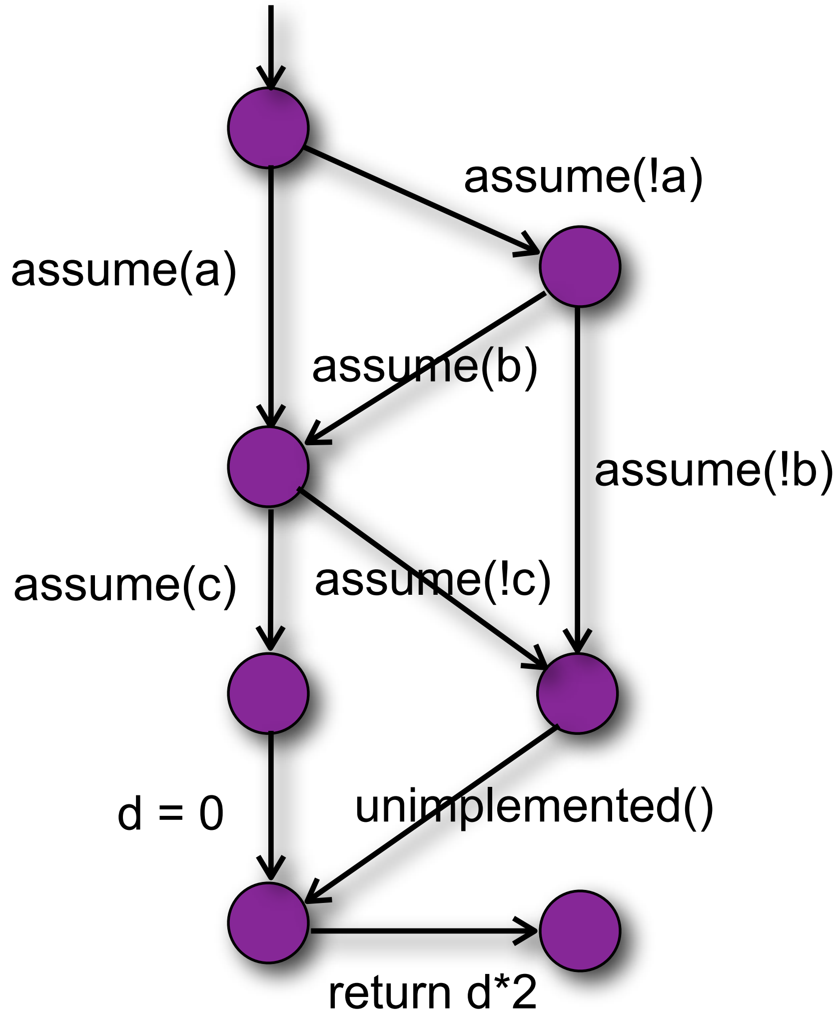


Path Monitors



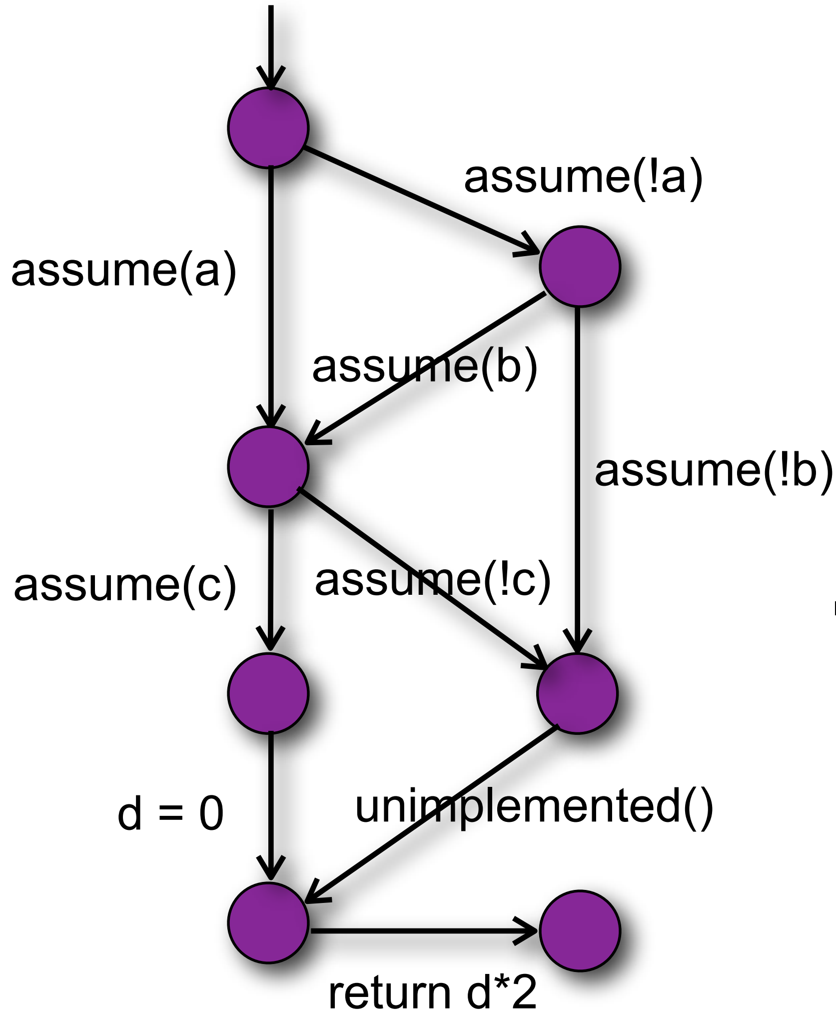
- How to exclude undesired paths? Describe specific executions?

Path Monitors



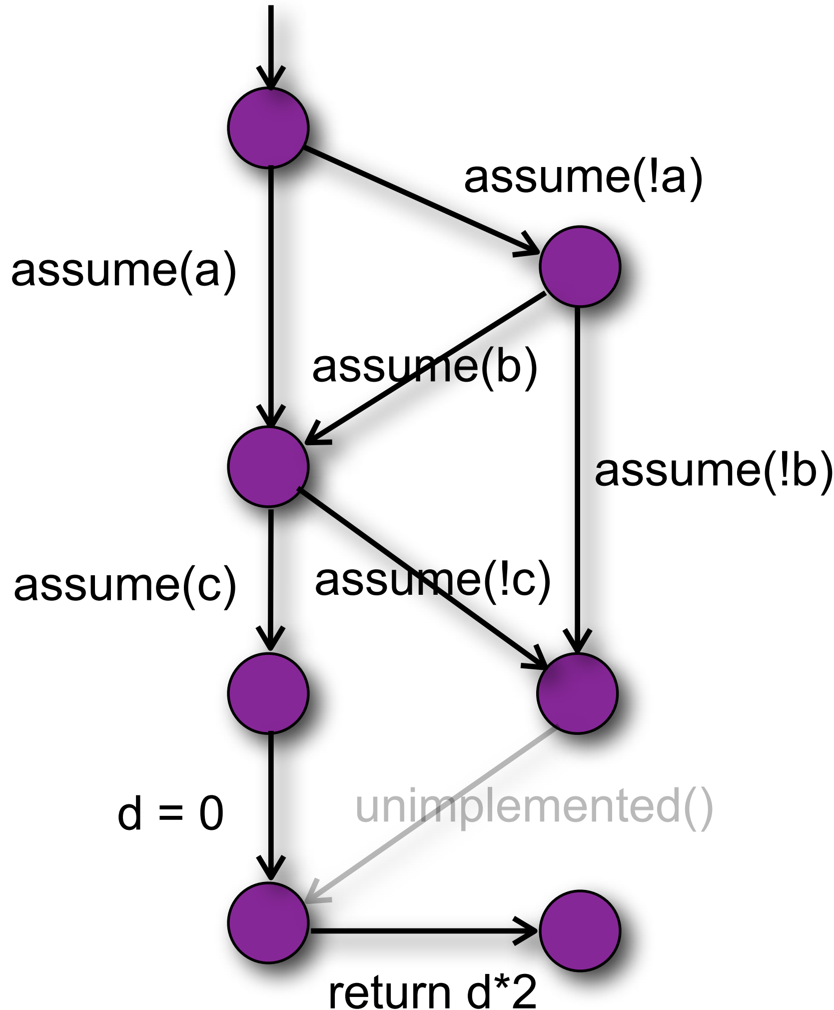
- How to exclude undesired paths? Describe specific executions?
- Regular expressions over CFA edges, use filter functions!

Path Monitors



- How to exclude undesired paths? Describe specific executions?
- Regular expressions over CFA edges, use filter functions!
  - `COMPLEMENT(@call(unimplemented))*`

Path Monitors

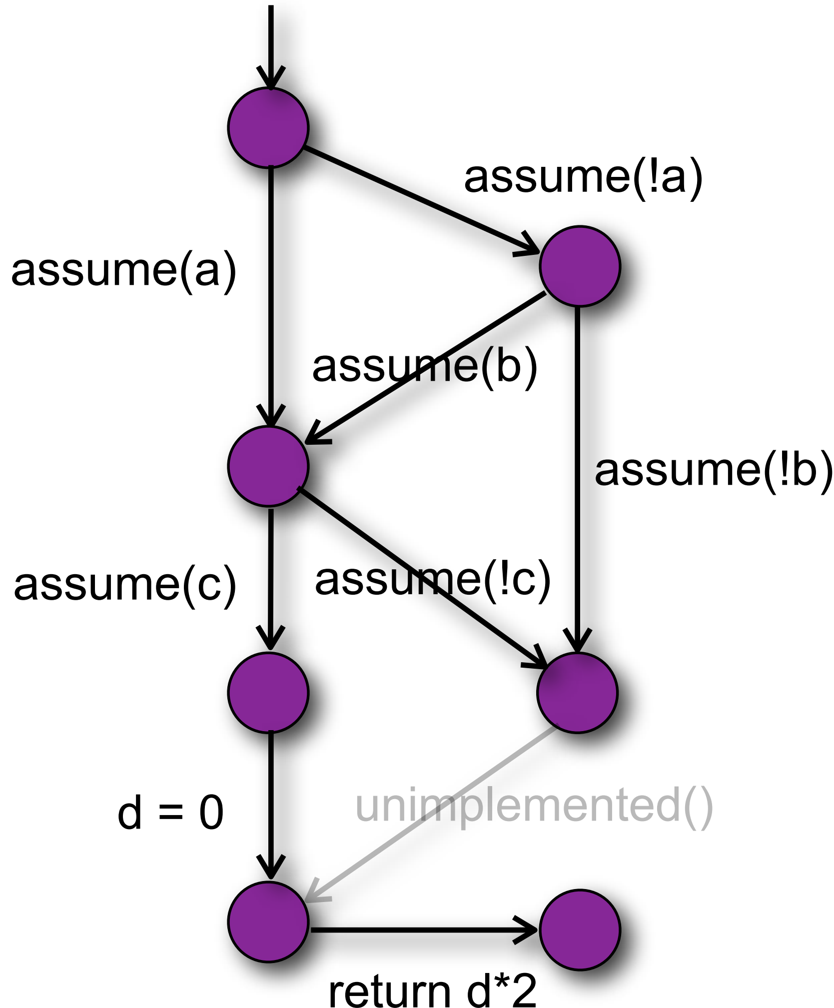


- How to exclude undesired paths? Describe specific executions?
- Regular expressions over CFA edges, use filter functions!

- `COMPLEMENT(@call(unimplemented))*`

Path Monitors

TECHNISCHE UNIVERSITÄT DARMSTADT



- How to exclude undesired paths? Describe specific executions?
- Regular expressions over CFA edges, use filter functions!

  - `COMPLEMENT(@call(unimplemented))*`
  - `(id*.@call(insert).id*) >= 10 . @call(sort) . id*`

Coverage Sequences

TECHNISCHE UNIVERSITÄT DARMSTADT

- Cover all pairs of conditions in insert and sort:
  - ```
    cover EDGES(intersect(@basicblockentry, @func(insert))) ->
          EDGES(intersect(@basicblockentry, @func(sort)))
    ```
- Cartesian product of test goals
- Intermediate paths restricted using path monitors
  - ```
    cover EDGES(@call(partition))
    -[COMPLEMENT(@exit(partition))*]>
    EDGES(INTERSECT(@func(partition), @conditionedge))
    ```

FQL Queries

TECHNISCHE UNIVERSITÄT DARMSTADT

```
in prefix cover goals passing scope
```

- Condition coverage in function "compare" with test cases which call "compare" from inside function "sort" only
  - ```
    cover EDGES(INTERSECT(@CONDITIONEDGE,@FUNC(compare)))
    passing (COMPLEMENT(@CALL(compare)) +
             INTERSECT(@CALL(compare),@FUNC(sort)))*
    ```
- Cover all basic blocks in function "mainloop" while never calling "init"
  - ```
    cover EDGES(INTERSECT(@BASICBLOCKENTRY,@FUNC(mainloop)))
    passing COMPLEMENT(@CALL(init))*
    ```
- Basic block coverage in function "sort" and require each test case to use a list of at least five but at most 15 elements
  - ```
    in @FUNC(sort) cover EDGES(@ENTRY(sort)){"len>=5 && len<=15"}
    -[COMPLEMENT(@EXIT(sort))]> EDGES(@BASICBLOCKENTRY)
    ```
    ```

Current Work

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Finalizing FQL (dependency coverage)
- Completing FShell
- Backend using Configurable Program Analysis
- Eclipse plug-in
- DO-178B compatible test case generation: Slope Testing