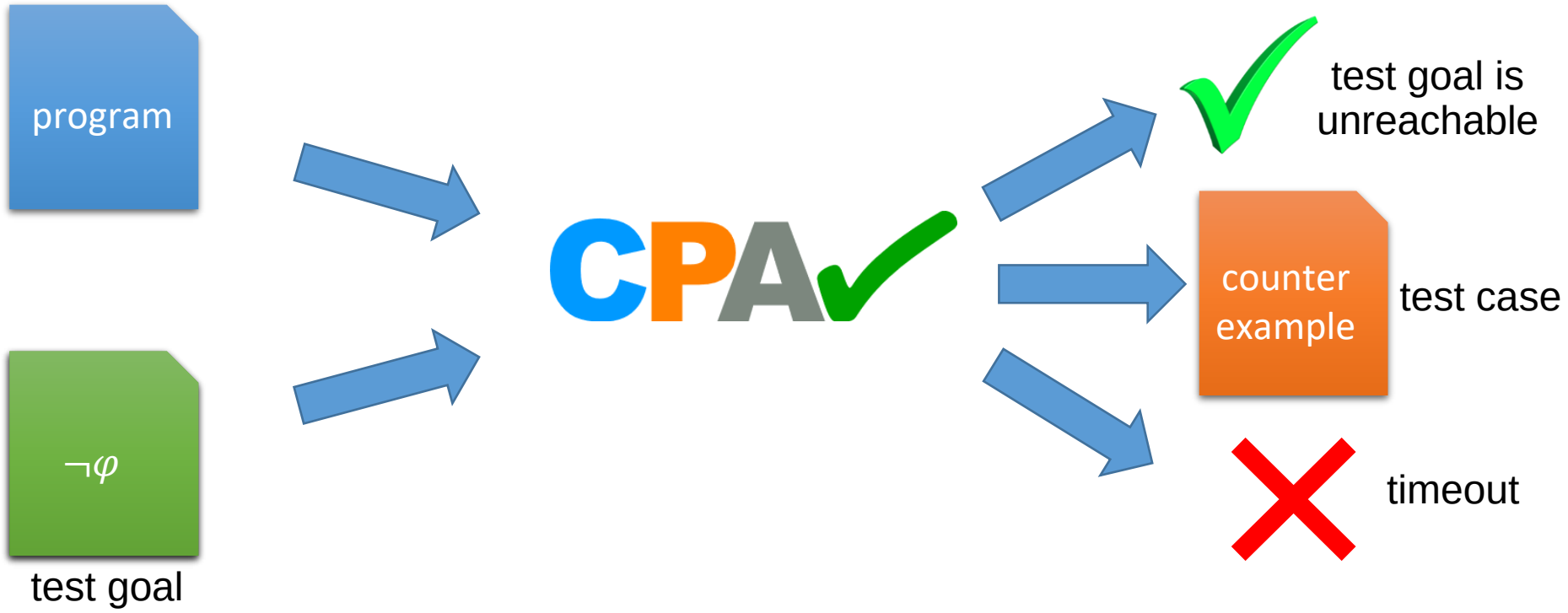
Test Generation with CPAchecker

CPAchecker Workshop Moscow
25.-26.09.2018

Sebastian Ruland
sebastian.ruland@es.tu-darmstadt.de

TEST-CASE GENERATION WITH CPACHECKER

Test-Case Generation with Model Checking



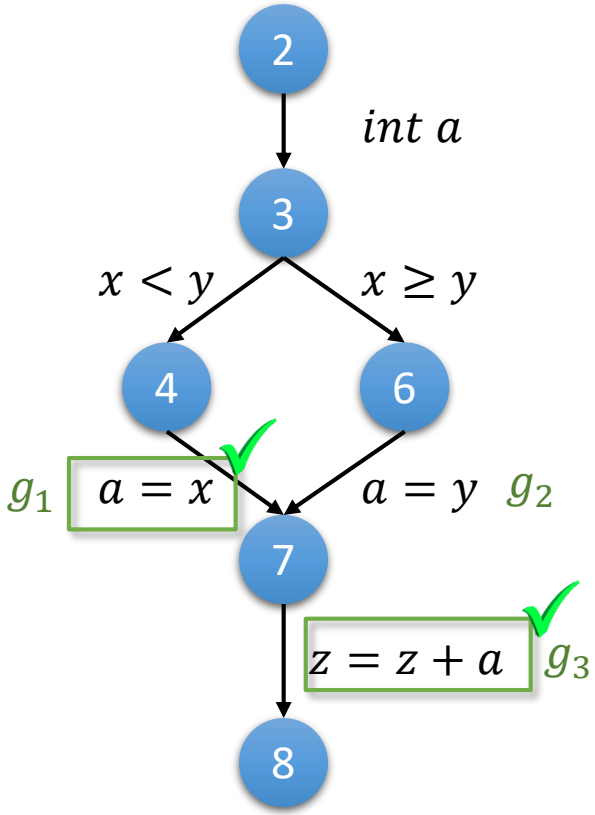
test case for unit testing: $tc = (I, O)$

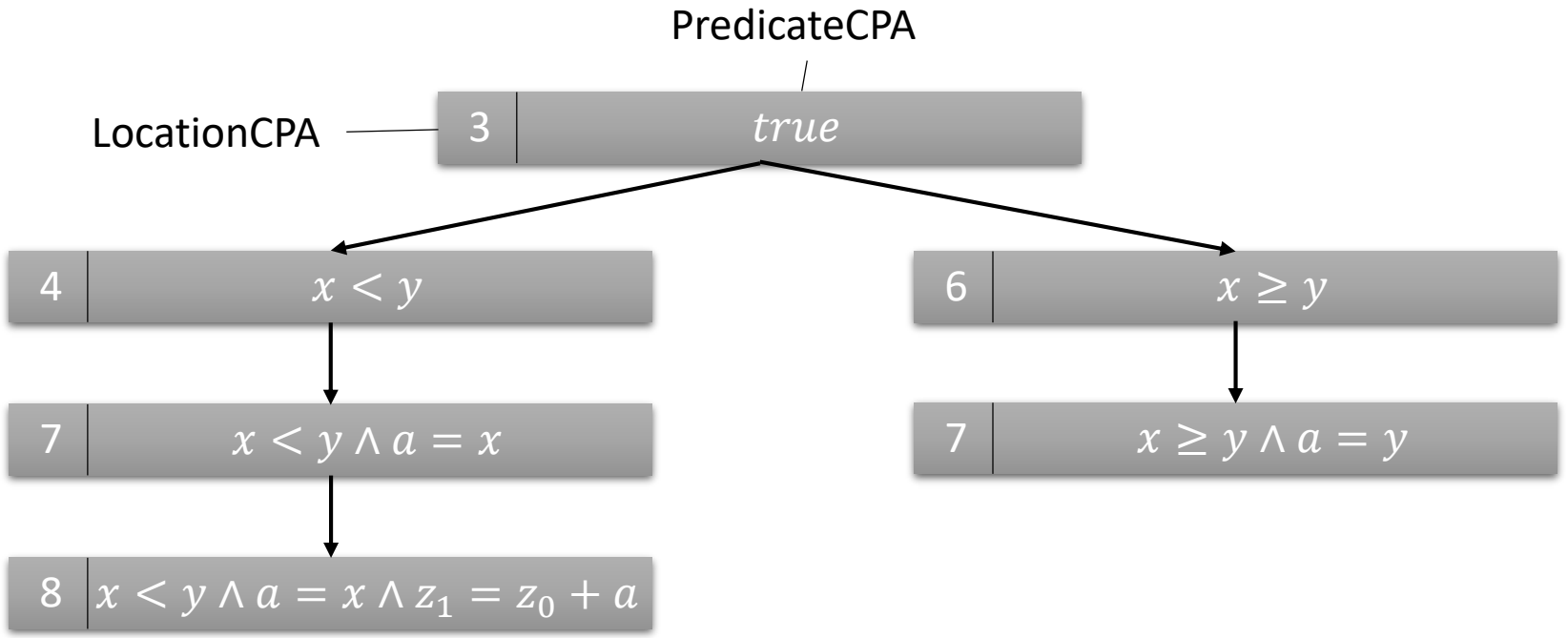
Test Scenarios

- Coverage testing
  - Statement coverage
  - Branch coverage
  - etc.

- Specification testing
  - One program as specification/oracle
  - One program as UUT

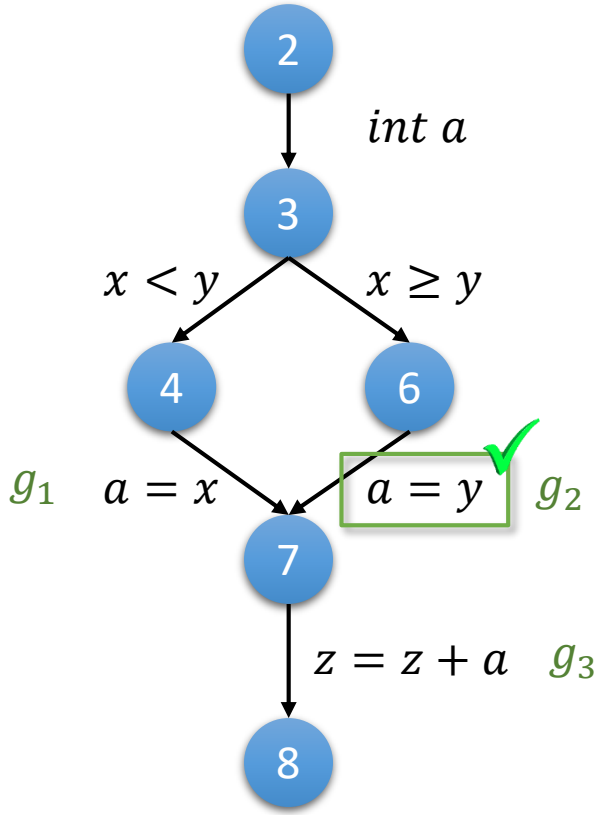
Tiger Algorithm

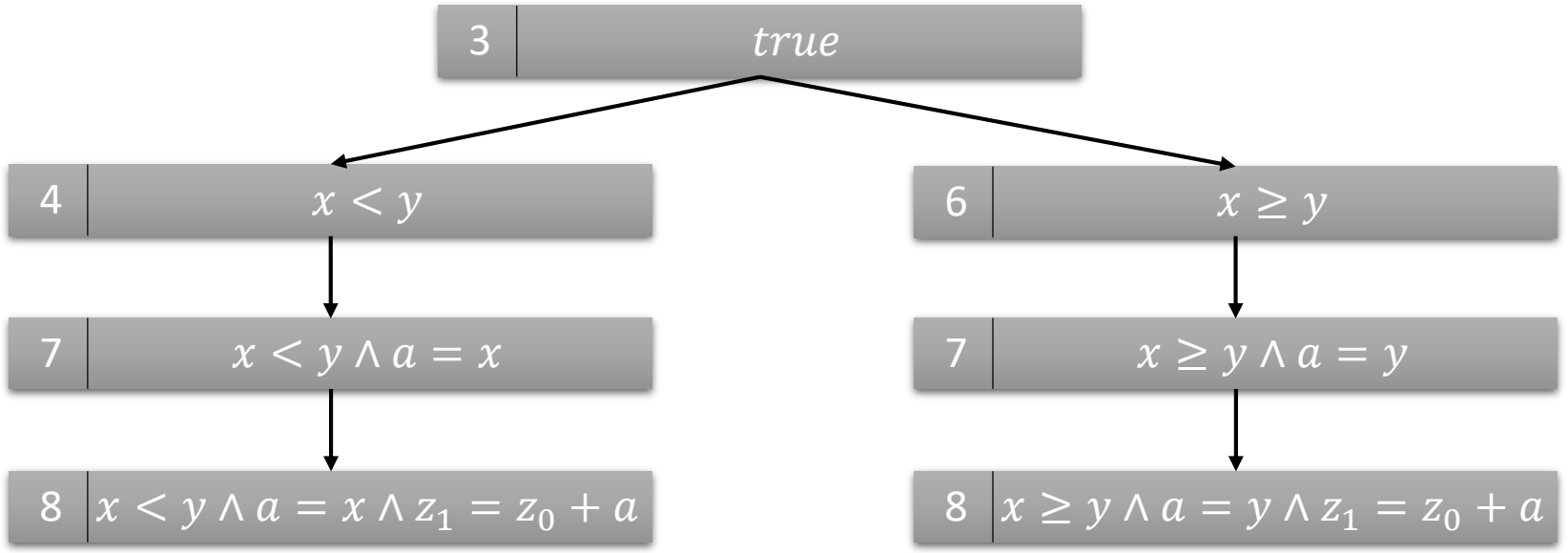




Testcase: $(x < y) \wedge (a = x) \wedge (z_1 = z_0 + a)$

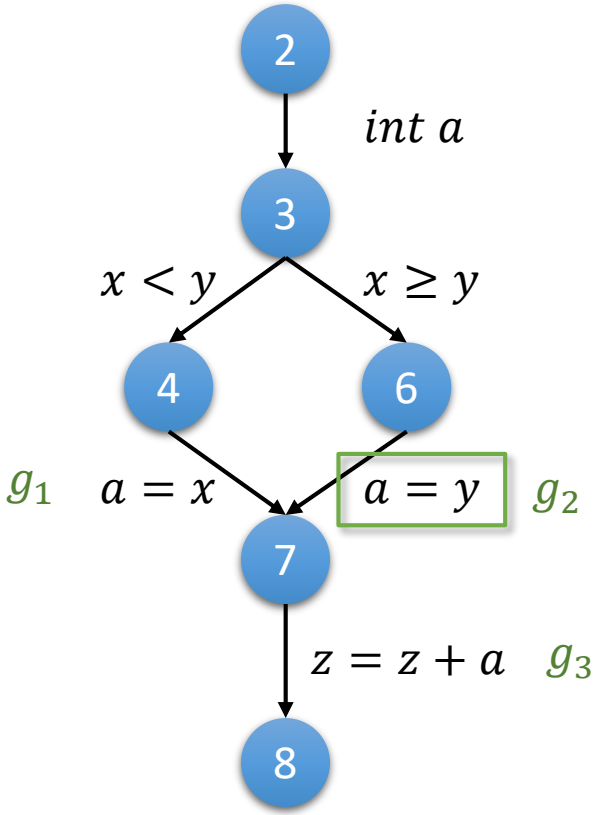
Tiger Algorithm

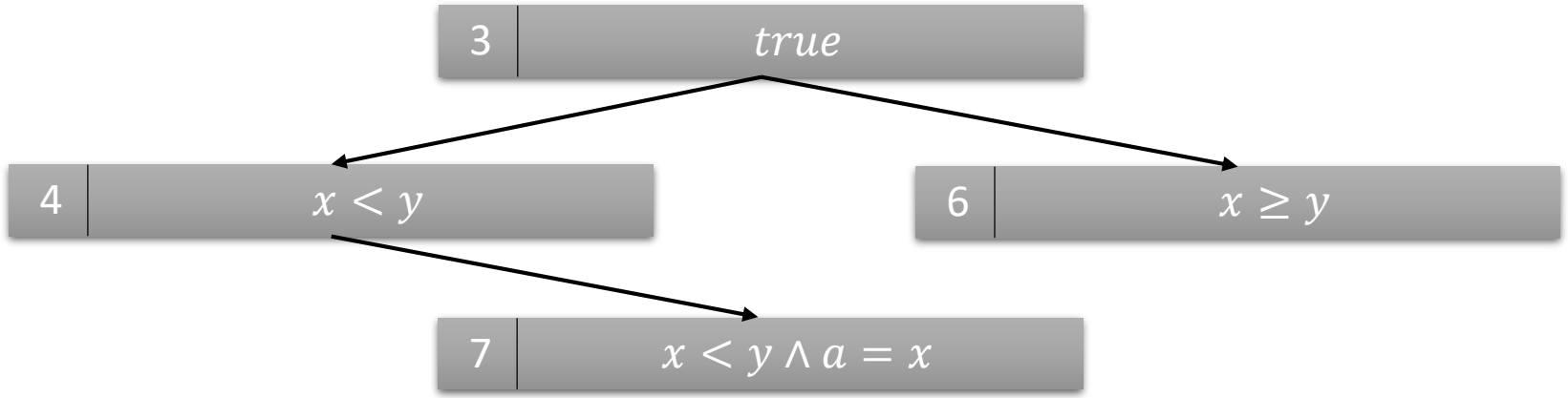




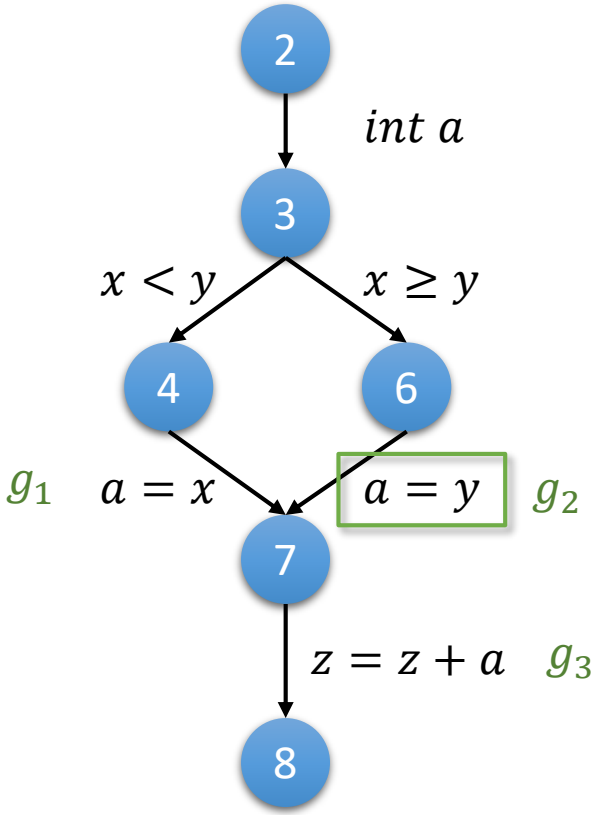
Testcase: $(x \geq y) \wedge (a = y) \wedge (z_1 = z_0 + a)$

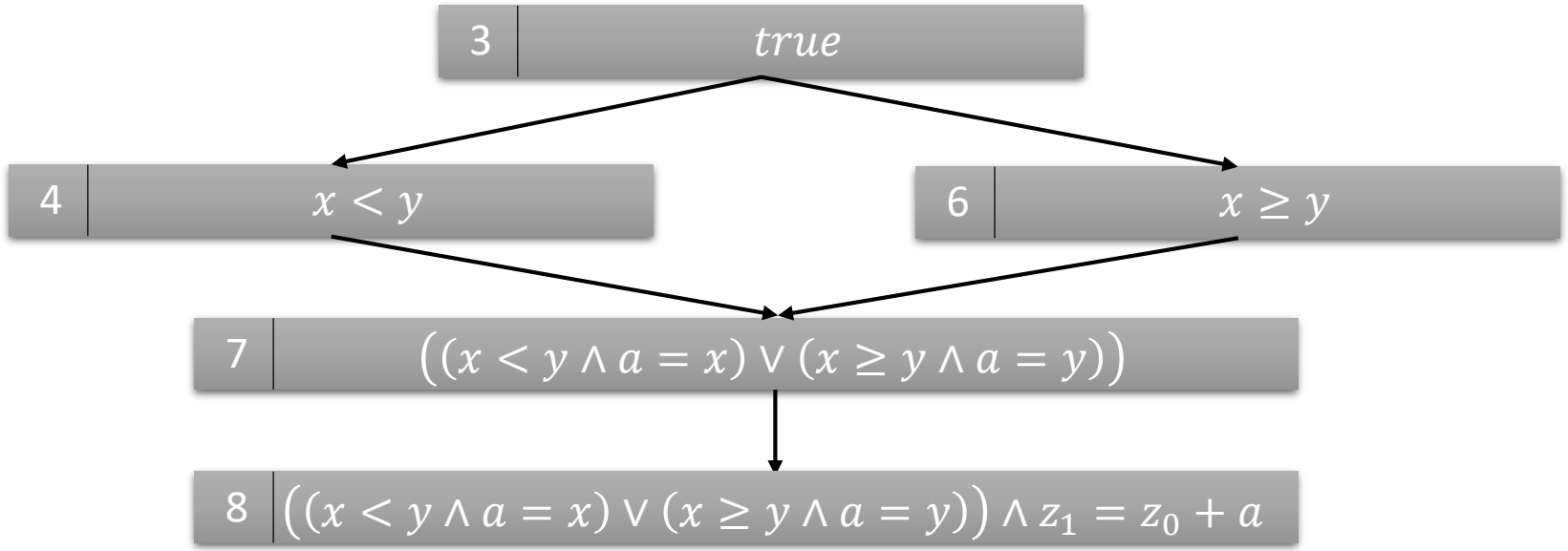
Merging





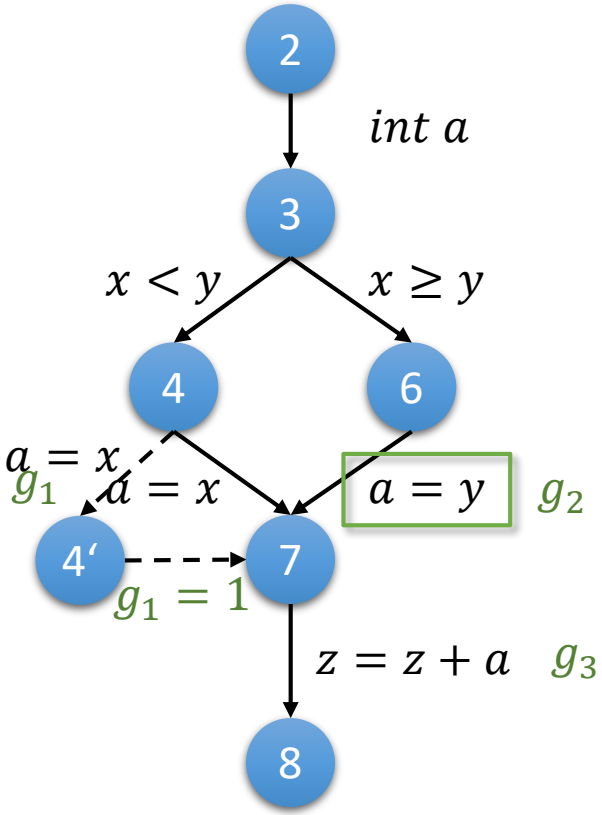
Merging

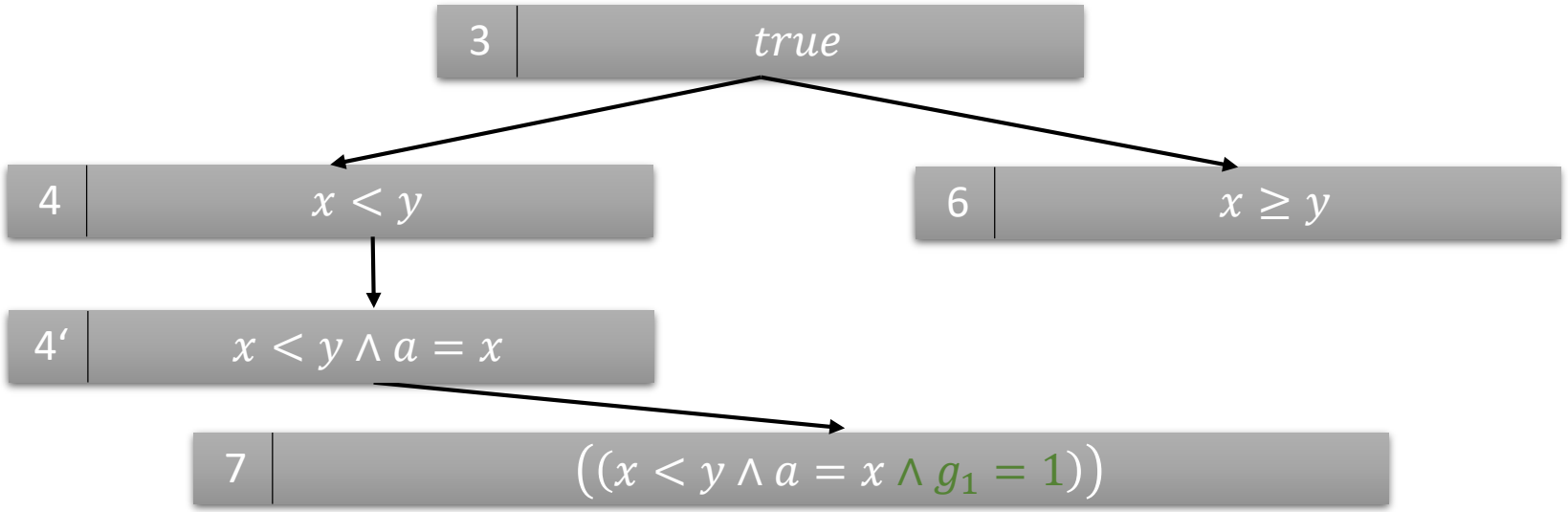




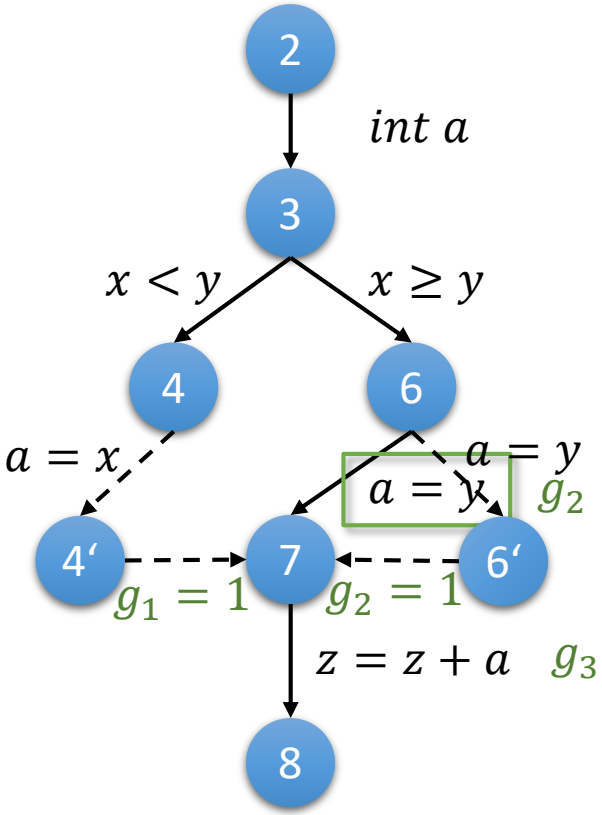
Which goal corresponds to which path?
Needed for output and input values

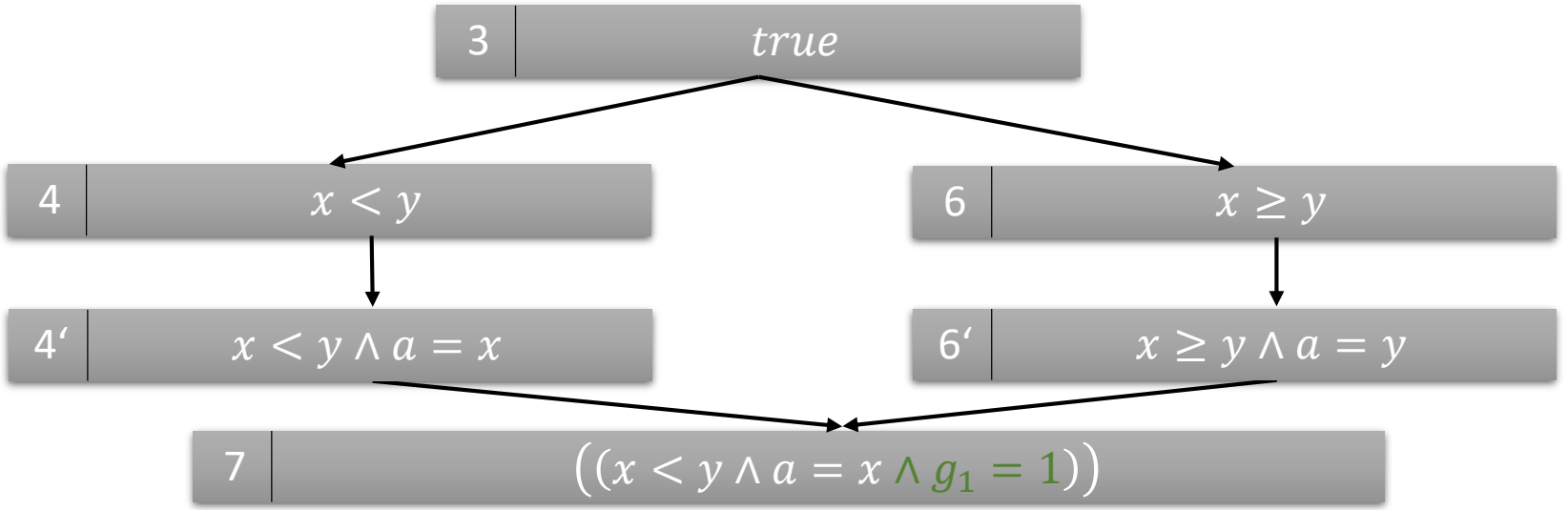
Merging with (Test-Goal) Weaving



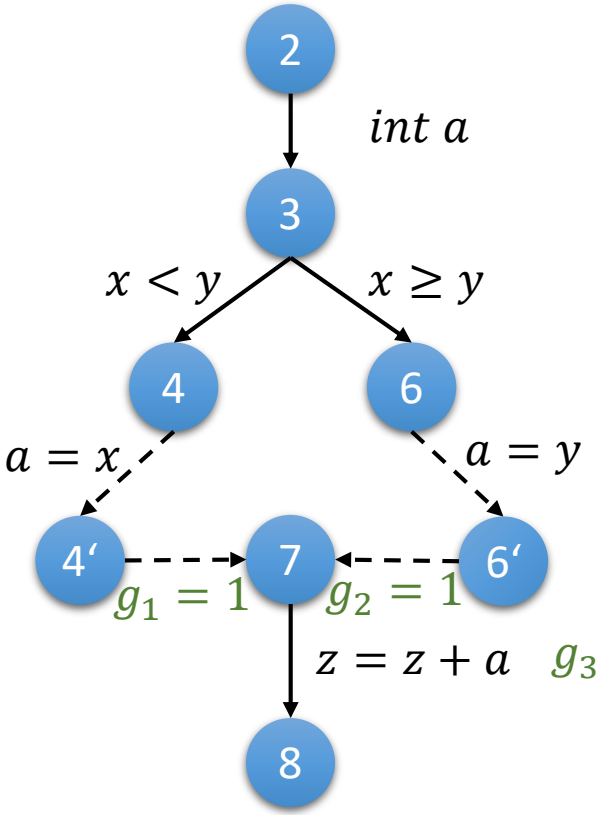


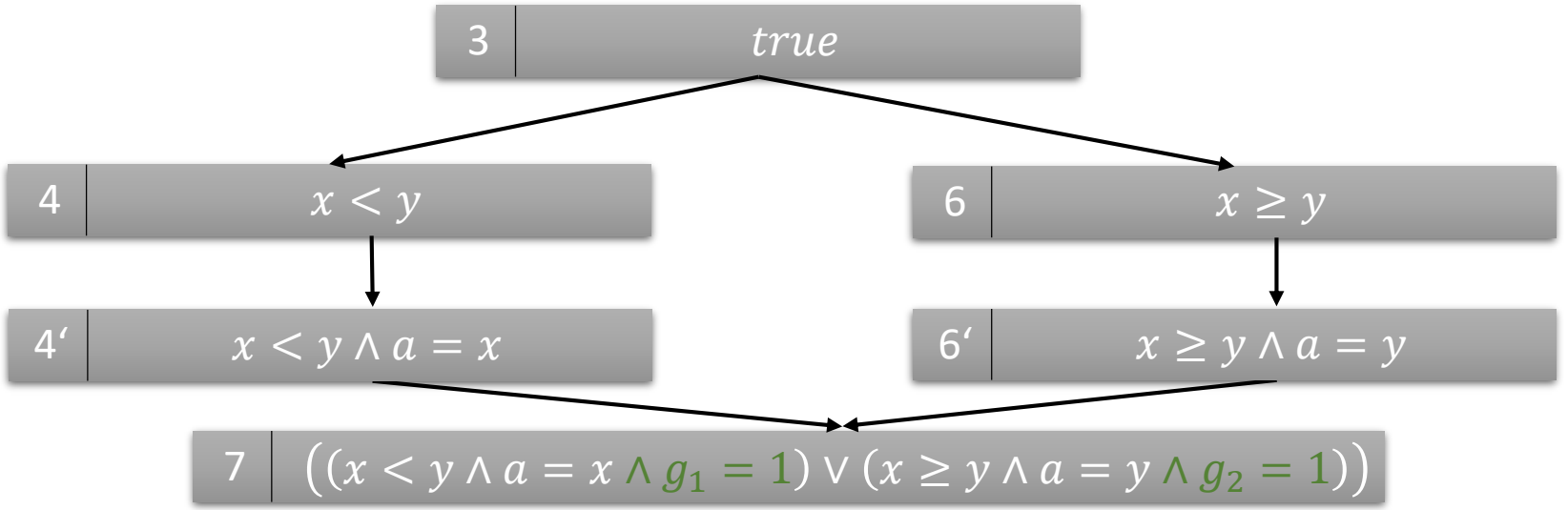
Merging with (Test-Goal) Weaving



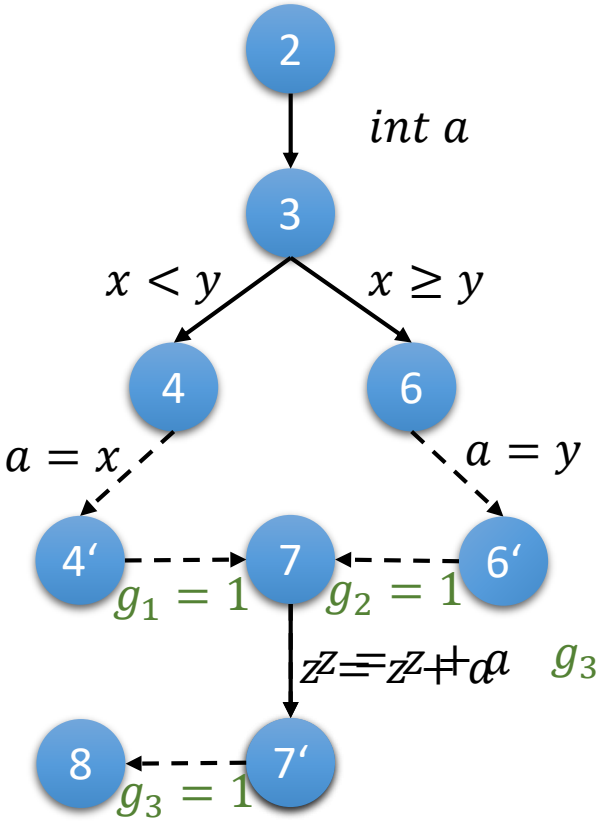


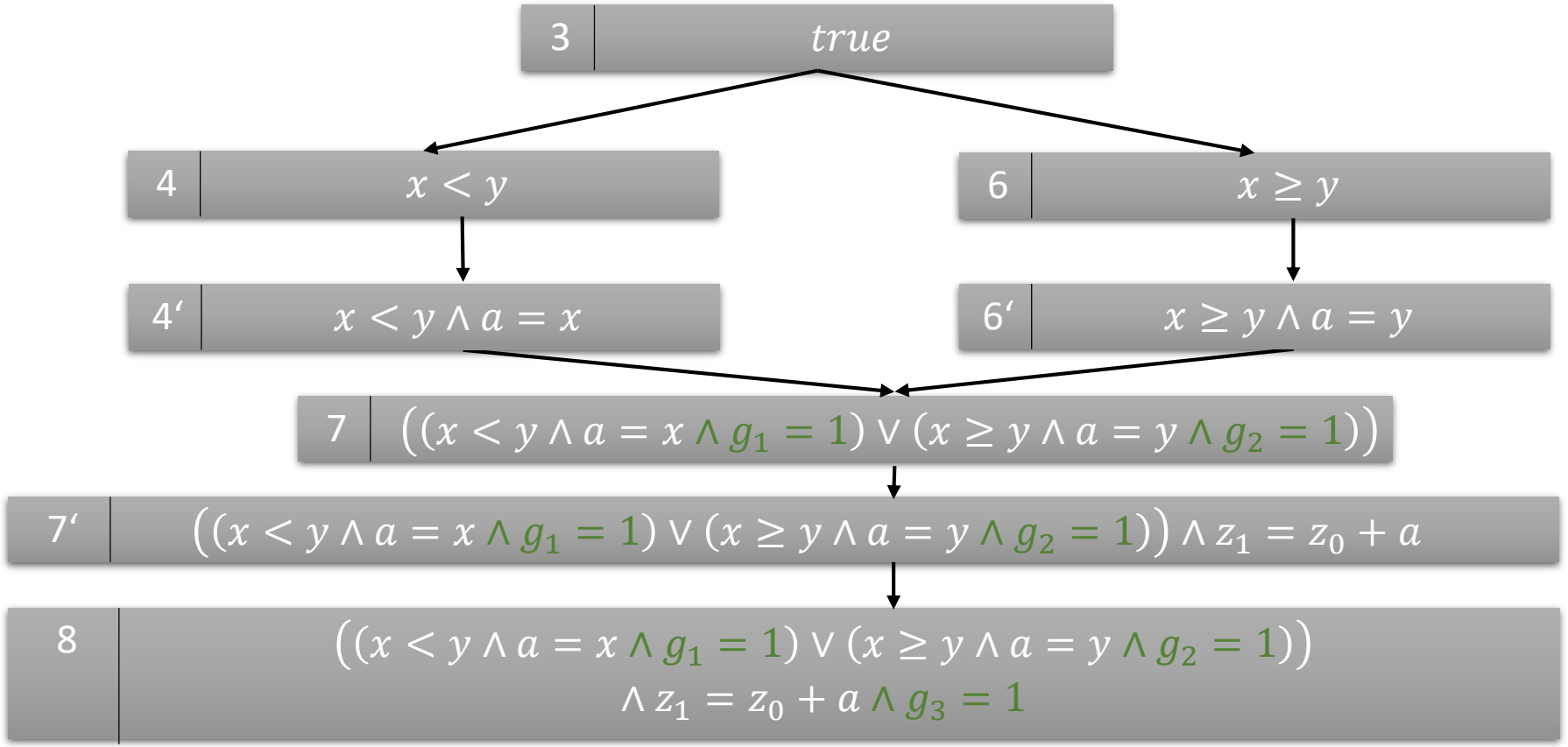
Merging with (Test-Goal) Weaving





Merging with (Test-Goal) Weaving





Test-Case Generation Conclusion

- Test-Case consists of input and output values
  - => merging needs test-goal weaving

- Test-Suite contains test-cases for optimal coverage

MUTATION TESTING

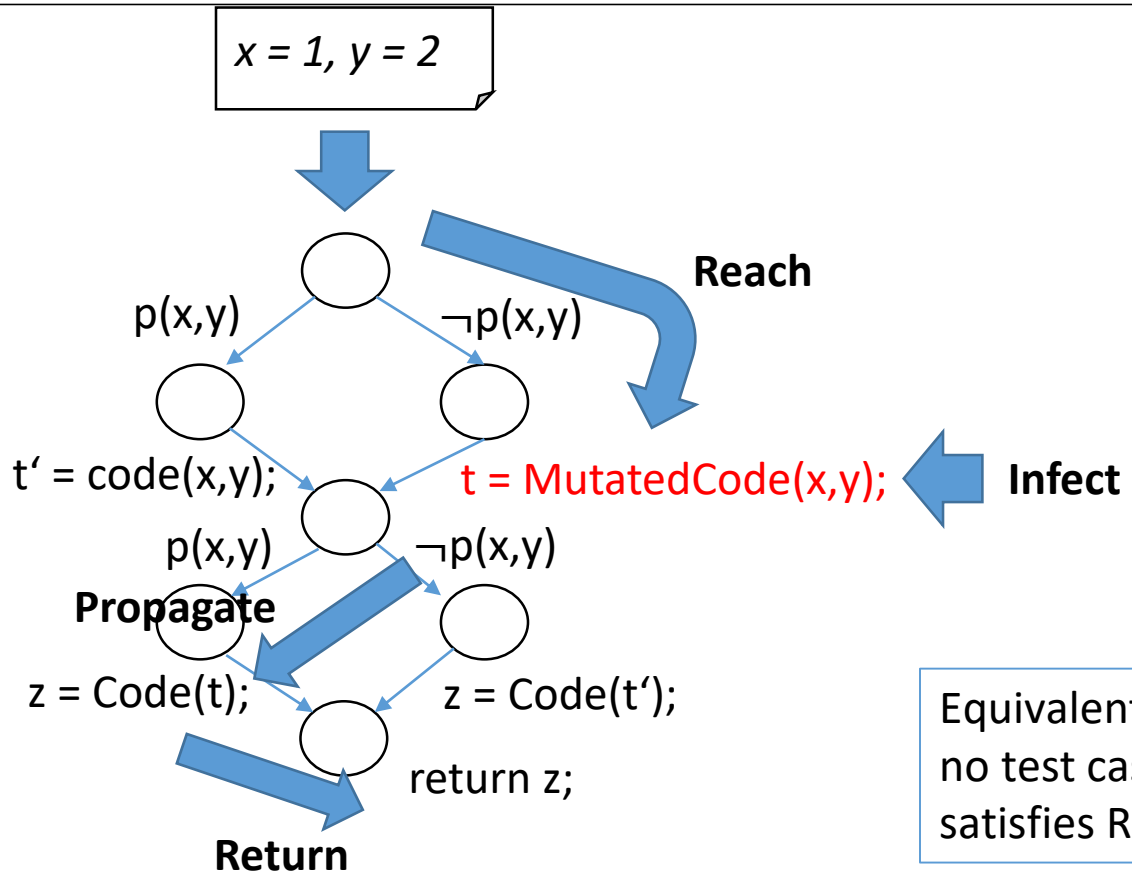
Mutation Testing

- Originally used to measure test-suite effectiveness
- Mimic frequent faults
- based on coupling hypothesis

RIPR

```
int foo(int x; int y) {
…

t = MutatedCode;
…
return z;
}
```



Equivalent mutant iff
no test case exists that
satisfies RIPR

Mutation Testing (Testsuite Effectiveness Measurement)

```
int func(int x, int y) {
  int r;
  if (x < y) {
    r = x+1;
  } else {
    r = y-x;
  }
  return r;
}
```

```
int mutant(int x, int y) {
  int r;
  if (x < y) {
    r = x+1;
  } else {
    r = y+x;
  }
  return r;
}
```

```
int main(){

  int x = 1;
  int y = 2;

  int result = func(x,y);
  int resultm = mutant(x,y);

  if(result!=resultm)
    mutation: printf("mutation");

}
```

extracted from an existing test case

goal

Mutation Testing (Mutant-Detecting Test-Case Generation)

```
int func(int x, int y) {
  int r;
  if (x < y) {
    r = x+1;
  } else {
    r = y-x;
  }
  return r;
}
```

```
int mutant(int x, int y) {
  int r;
  if (x < y) {
    r = x+1;
  } else {
    r = y+x;
  }
  return r;
}
```

```
int main(){

  int x = _nondet_int();
  int y = _nondet_int();

  int result = func(x,y);
  int resultm = mutant(x,y);

  if(result!=resultm)
    mutation: printf("mutation");

}
```


Mutation Testing (Mutant-Detecting Test-Case Generation)

```
int func(int x, int y) {
  int r;
  if (x < y) {
    r = x+1;
  } else {
    r = y-x;
  }
  return r;
}
```

```
int mutant0(int x, int y) {
  …
}

int mutant1(int x, int y) {
  …
}

int mutant2(int x, int y) {
  …
}
…
```

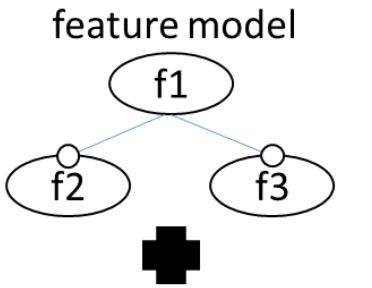
```
int main(){

  int x = _nondet_int();
  int y = _nondet_int();

  int result = func(x,y);

  if(result!=mutant0(x,y))
    mut0:printf("mutation0");
  if(result!=mutant1(x,y))
    mut1:printf("mutation1");
  if(result!=mutant2(x,y))
    mut2:printf("mutation2");
  …
  …
}
```


GPCE




```
int foo(int x; int y) {
...
#ifdef f2
 t = MutatedCode;
#endif
...
return z;
}
```

SPL code

```
int foo(int x; int y) {
...
#ifdef f2
 t = MutatedCode;
#endif
...
return z;
}
```

program variant of configuration $\{ f1 \}$ does not contain mutation

```
int foo(int x; int y) {
...
#ifdef f2
 t = MutatedCode;
#endif
...
return z;
}
```

program variant of configuration $\{ f1, f2 \}$ contains mutation

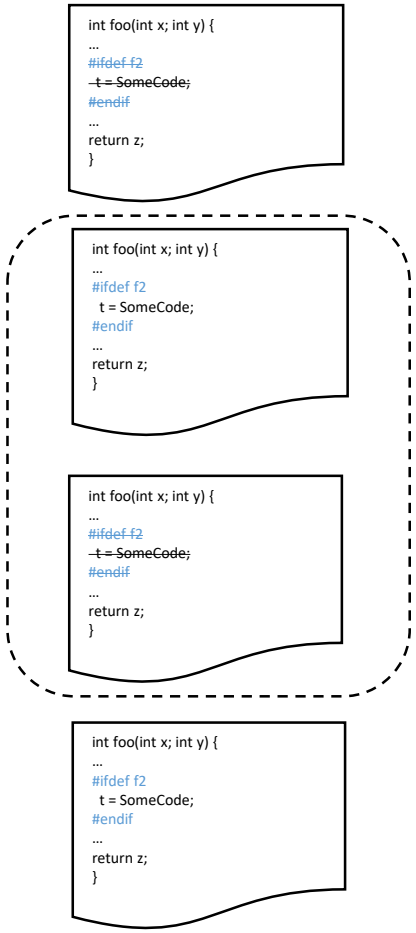
```
int foo(int x; int y) {
...
#ifdef f2
 t = MutatedCode;
#endif
...
return z;
}
```

program variant of configuration $\{ f1, f3 \}$ does not contain mutation

```
int foo(int x; int y) {
...
#ifdef f2
 t = MutatedCode;
#endif
...
return z;
}
```

program variant of configuration $\{ f1, f2, f3 \}$ contains mutation

GPCE



Sample

Mutant is killed by sample iff the sample contains a program variant for which a RIPR test case exists

x = 1, y = 2

x = 2, y = 1

x = 0, y = 0

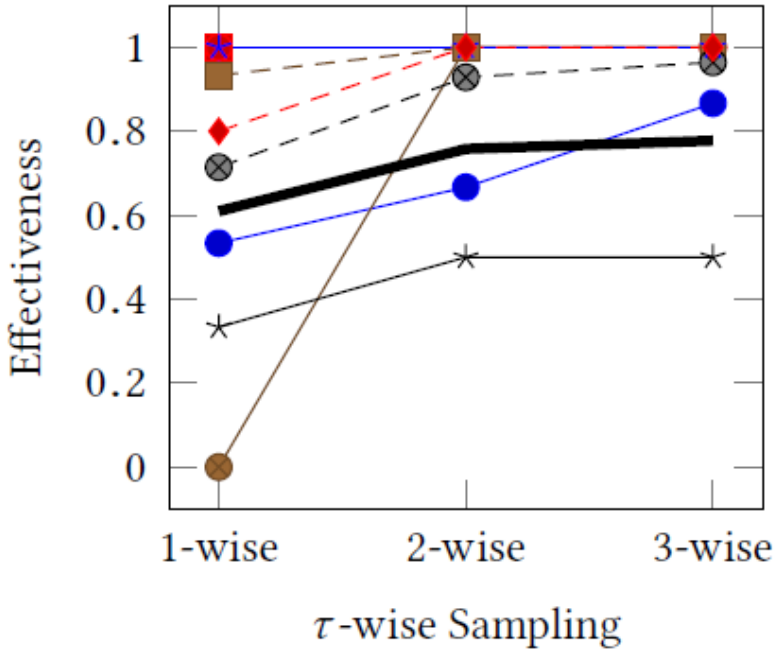
x = 1, y = 1

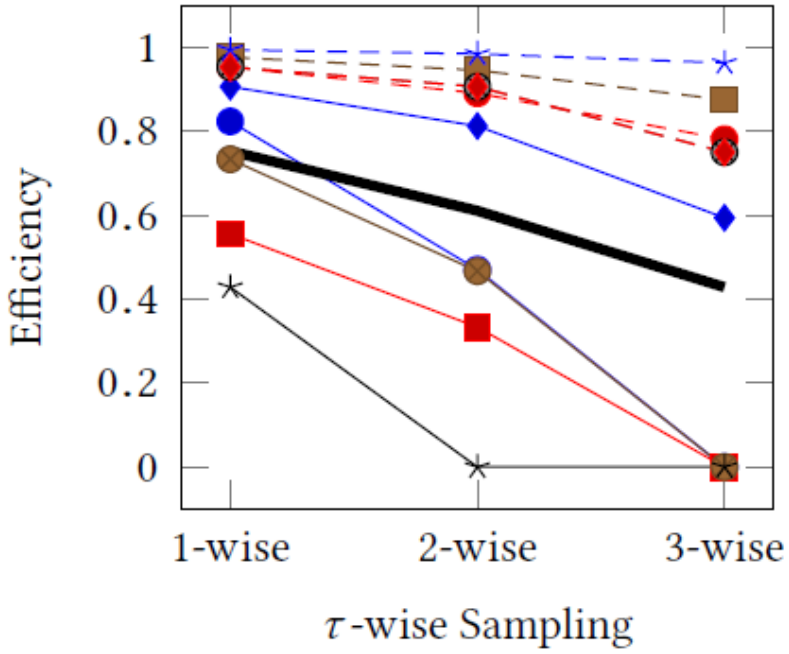
…

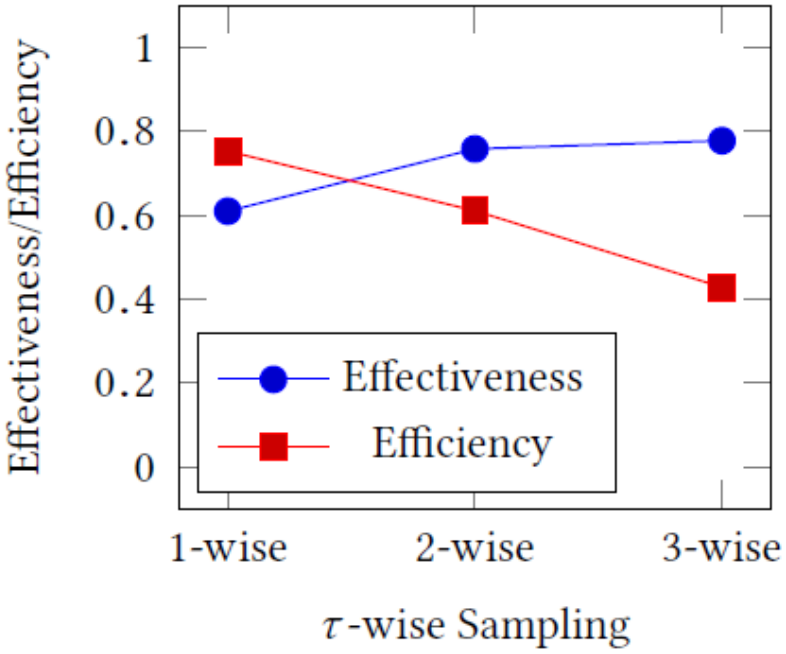
„perfect“ test suite TS

(generated by our new mutation-driven TS-Gen Approach)

Results


Effectiveness
1
0.8
0.6
0.4
0.2
0
1-wise
2-wise
3-wise
τ-wise Sampling


Efficiency
1
0.8
0.6
0.4
0.2
0
1-wise
2-wise
3-wise
τ-wise Sampling


Effectiveness/Efficiency
1
0.8
0.6
0.4
0.2
0
Effectiveness
Efficiency
1-wise
2-wise
3-wise
τ-wise Sampling

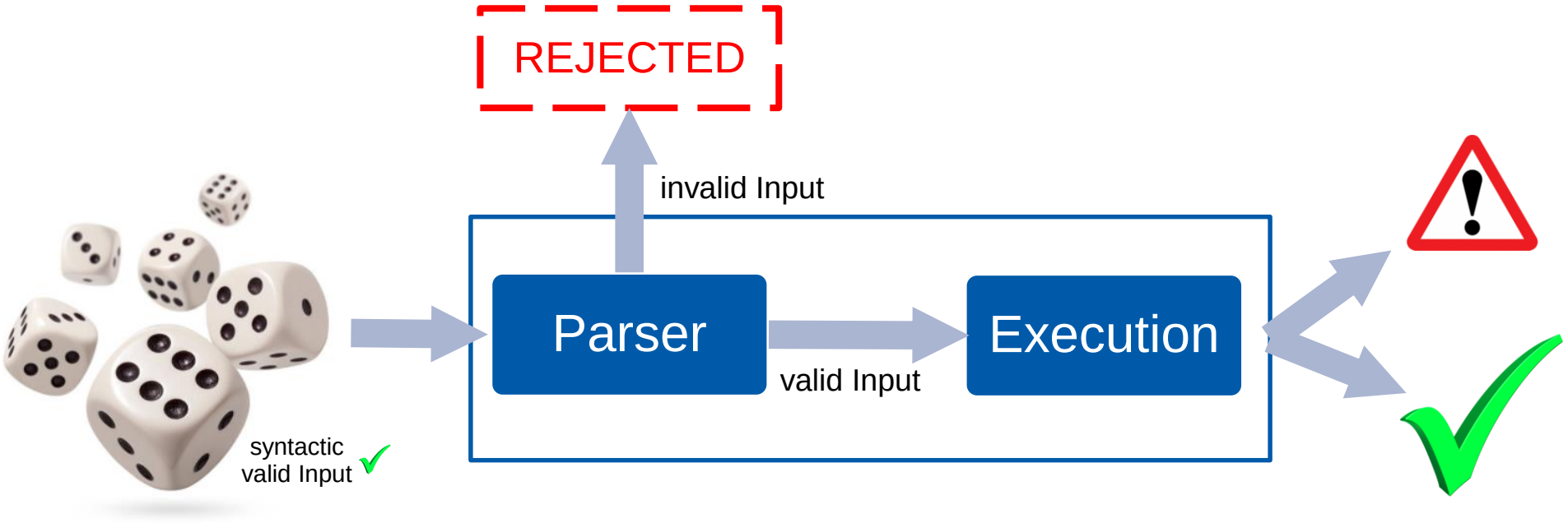
Mutation Testing Conclusion

- Testsuite Effectiveness Measurement
- Mutant-Detecting Test-Case Generation provides means to compare testing strategies to a „perfect test“ in terms of mutation detection
- Already used to evaluate effectiveness of samples in SPL-testing

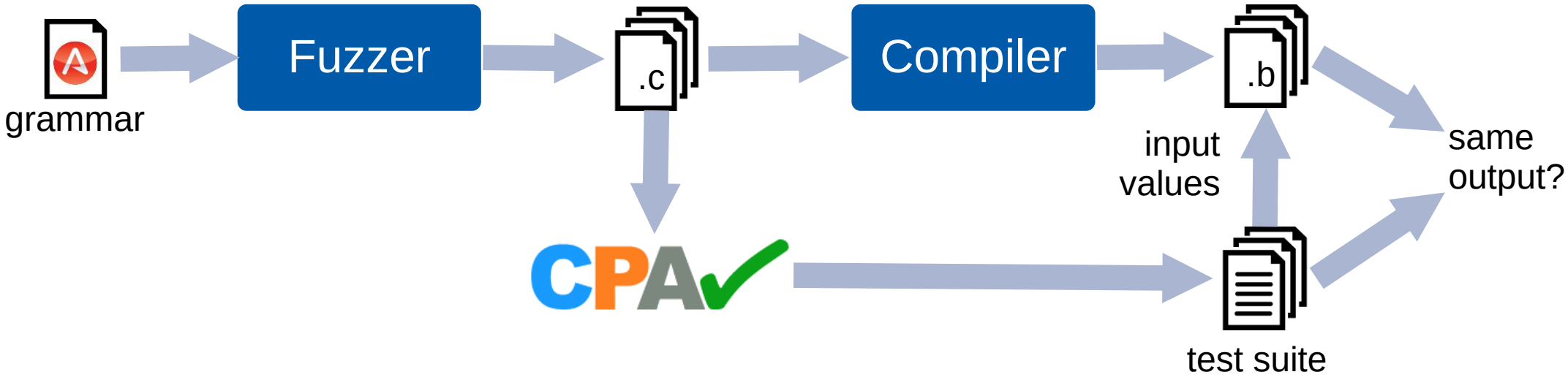
BLACKBOX FUZZING

Manual Blackbox Fuzzing

Blackbox Fuzzing for Complex Inputs

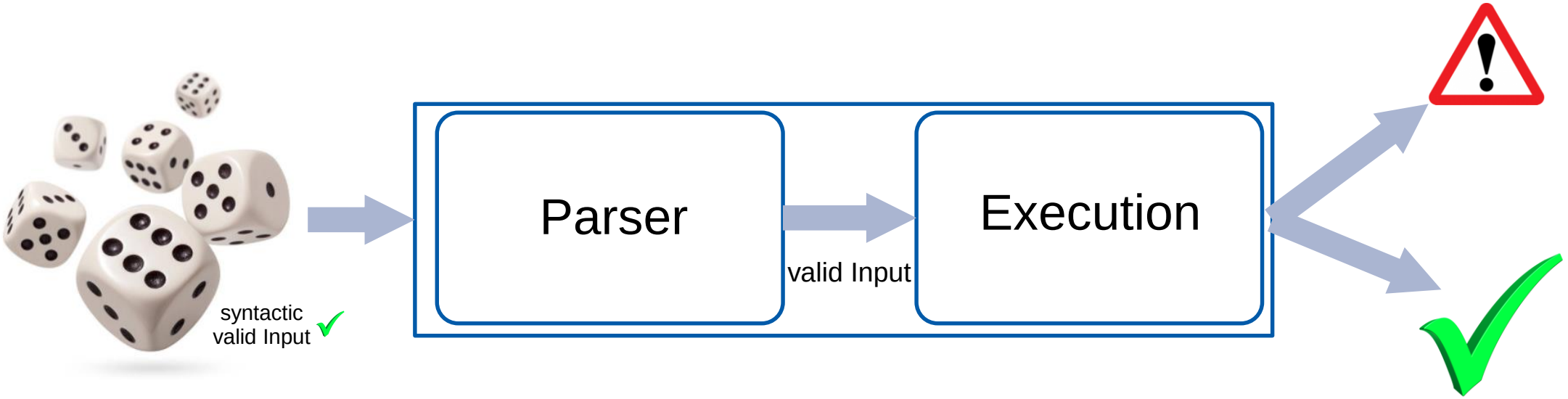


Grammar-based Blackbox Fuzzing



WHITEBOX FUZZING

Whitebox Fuzzing

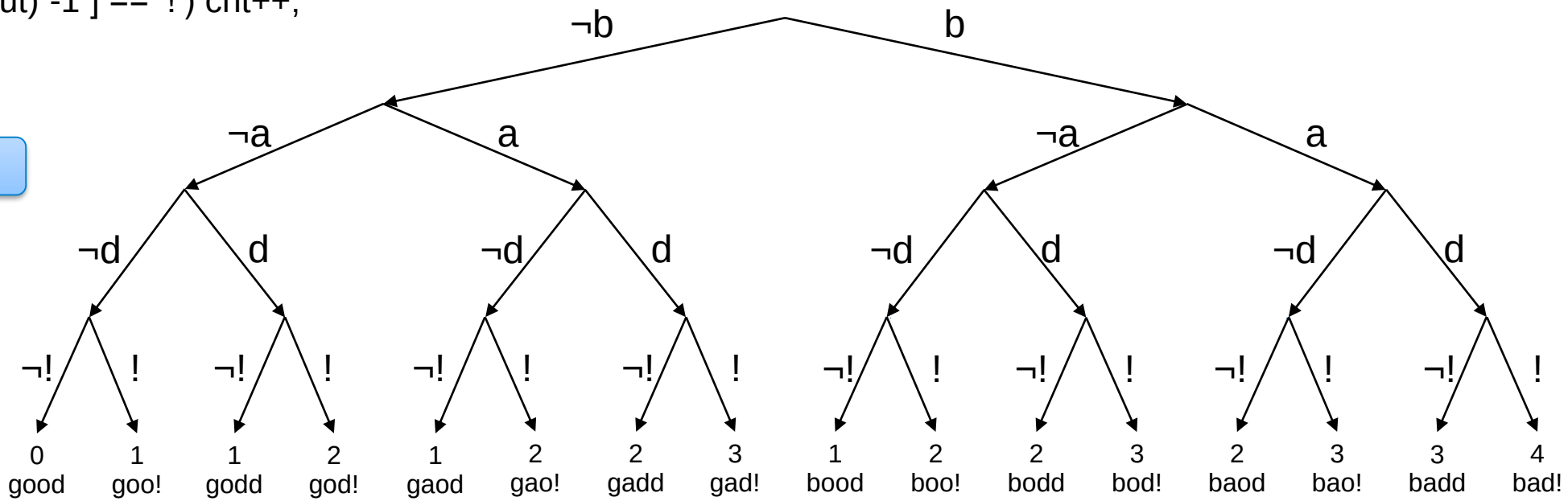


Example

```
void top(char input[]) {
    int cnt = 0;
    if (input[arraySize(input) -4 ] == ‘b’) cnt++;
    if (input[arraySize(input) -3 ] == ‘a’) cnt++;
    if (input[arraySize(input) -2 ] == ‘d’) cnt++;
    if (input[arraySize(input) -1 ] == ‘!’) cnt++;
    if (cnt >= 4) crash();
}
```

goal

Input: „good“



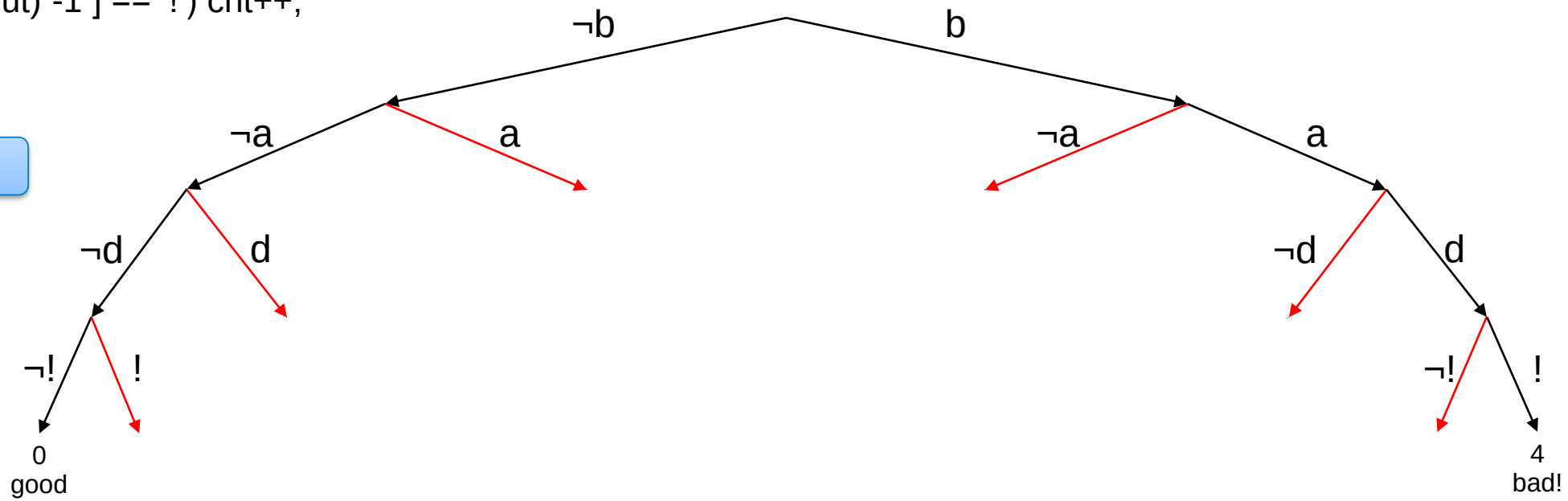
Example

```
void top(char input[]) {
    int cnt = 0;
    if (input[arraySize(input) -4 ] == ‘b’) cnt++;
    if (input[arraySize(input) -3 ] == ‘a’) cnt++;
    if (input[arraySize(input) -2 ] == ‘d’) cnt++;
    if (input[arraySize(input) -1 ] == ‘!’) cnt++;
    if (cnt >= 4) crash();
}
```

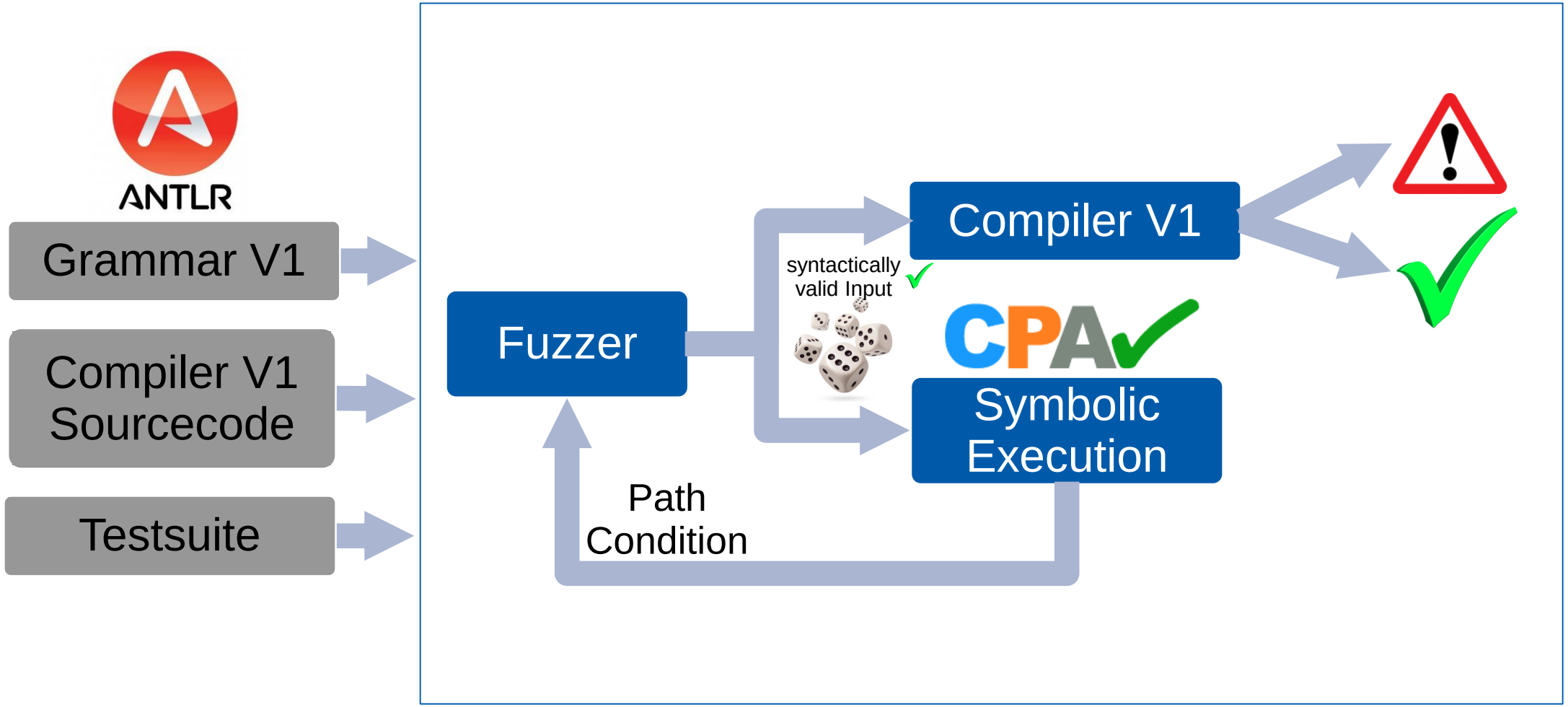
goal

Input: „good“

Grammar:
input: input? (‚good‘ | ‚bad!‘ | ‚poor‘)



Grammar-based Whitebox Fuzzing



Conclusion

- CPAchecker used to create test-cases consisting of input and output values
  - mainly used to create all test-cases needed for different coverage criteria

- Different partitioning strategies for multi-goal test-generation currently evaluated

- Future plans to use test-case generation with CPAchecker in combination with Blackbox Fuzzing
- Future plans to use symbolic execution with CPAchecker for Whitebox Fuzzing