Testing, Abstraction, Theorem Proving: Better Together

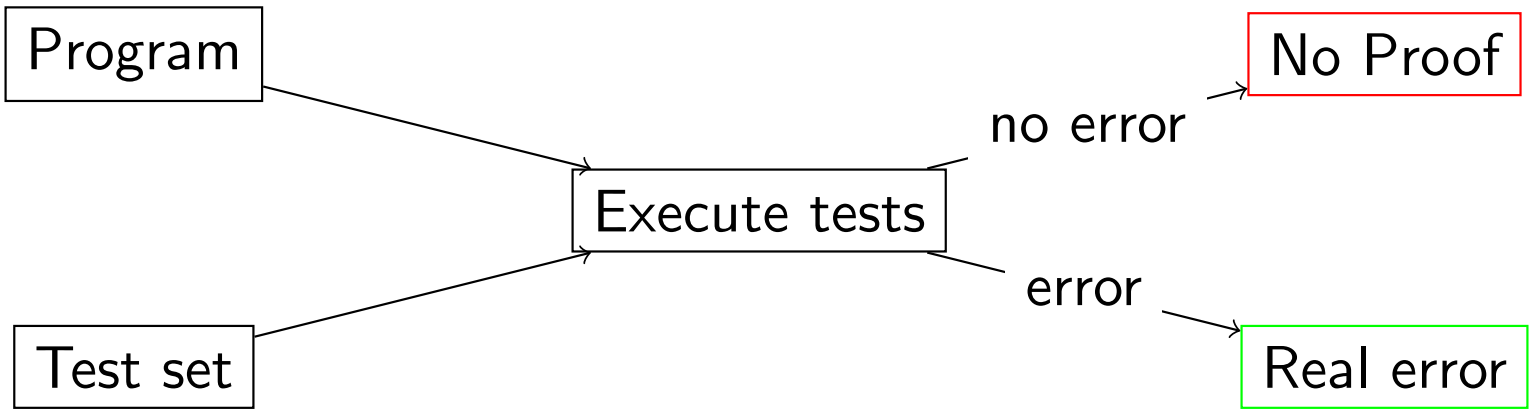
Authors: Greta Yorsh
Thomas Ball
Mooly Sagiv

Presenter: Michael Rudolph

Seminar Program Analysis and Software Testing

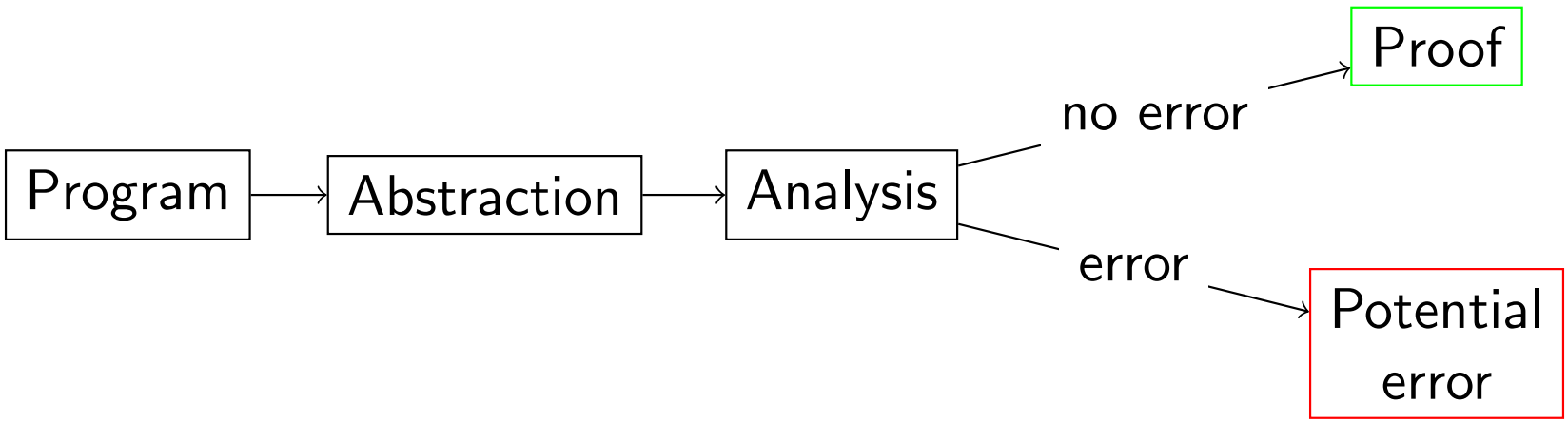
University of Freiburg 2016

Motivation



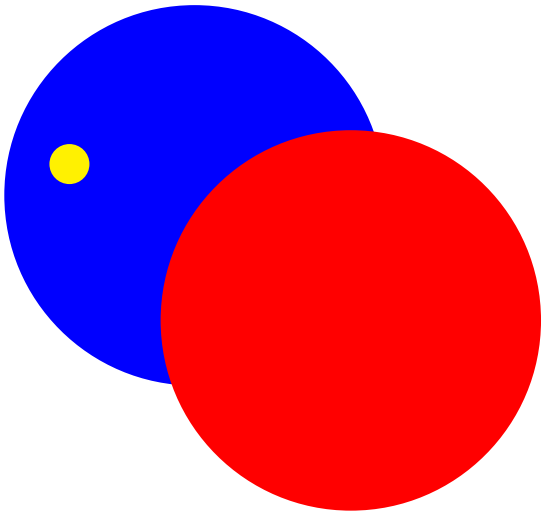
Dynamic Analysis

Motivation



Static Analysis

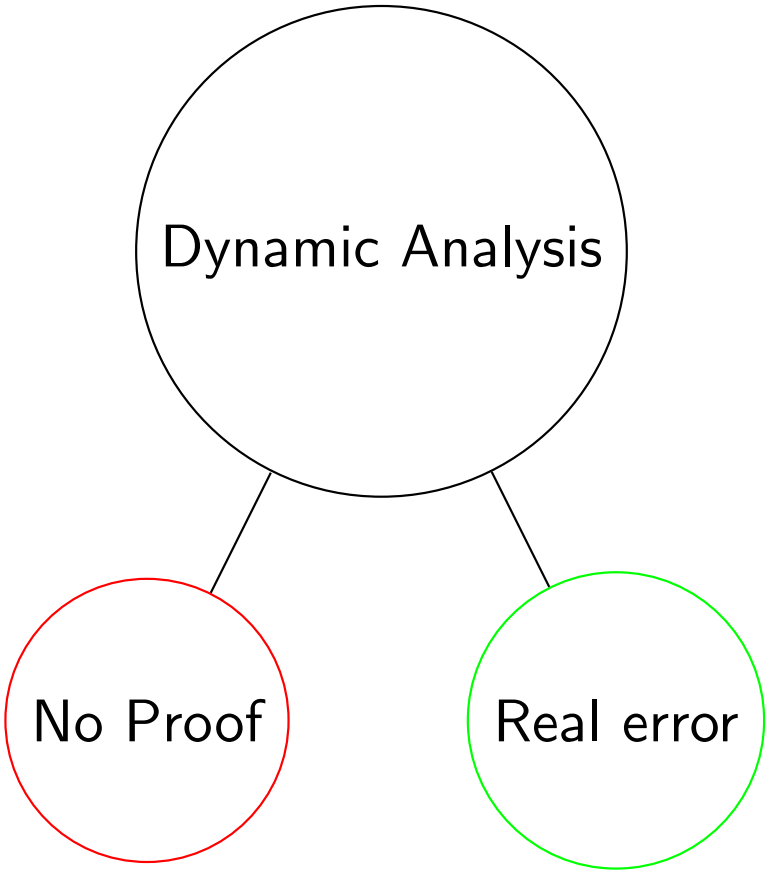
False error

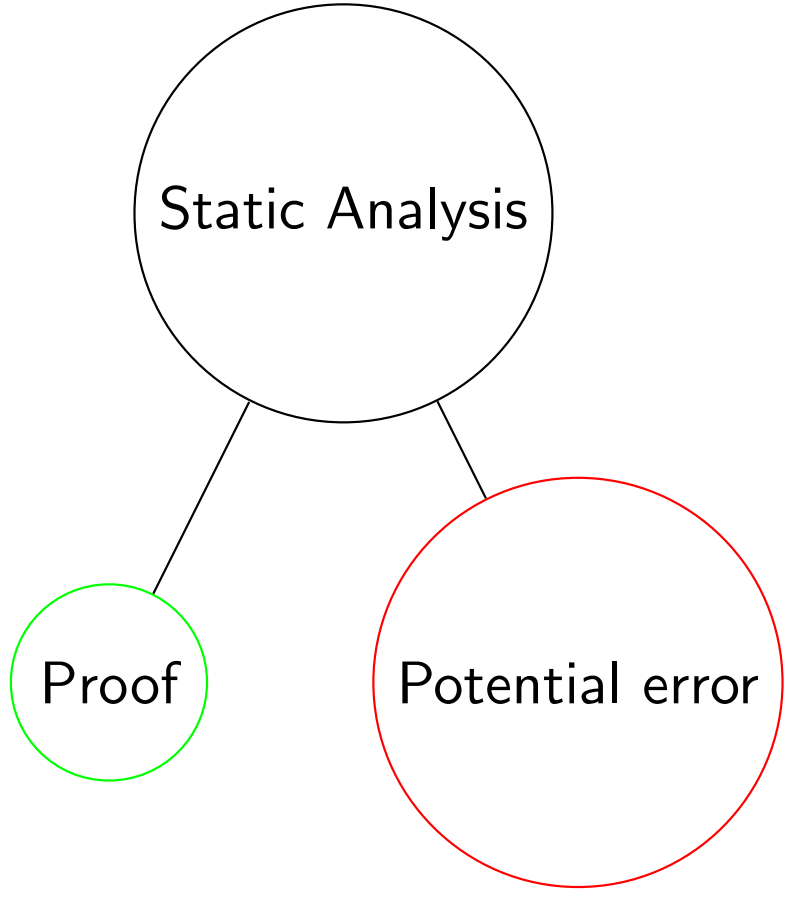


Unreachable concrete states
Reachable concrete states
false error

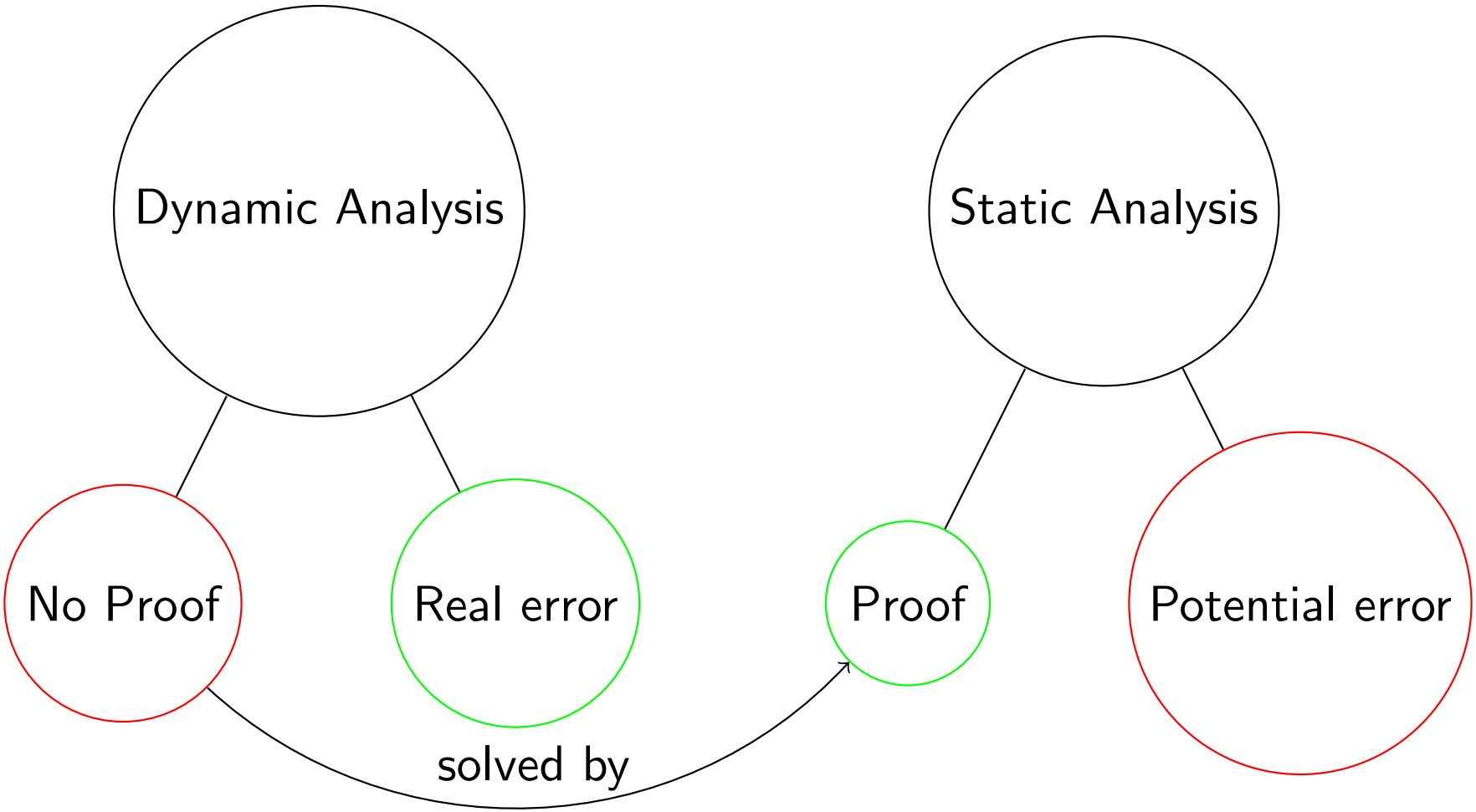
Abstract state = ■ + ■

Motivation

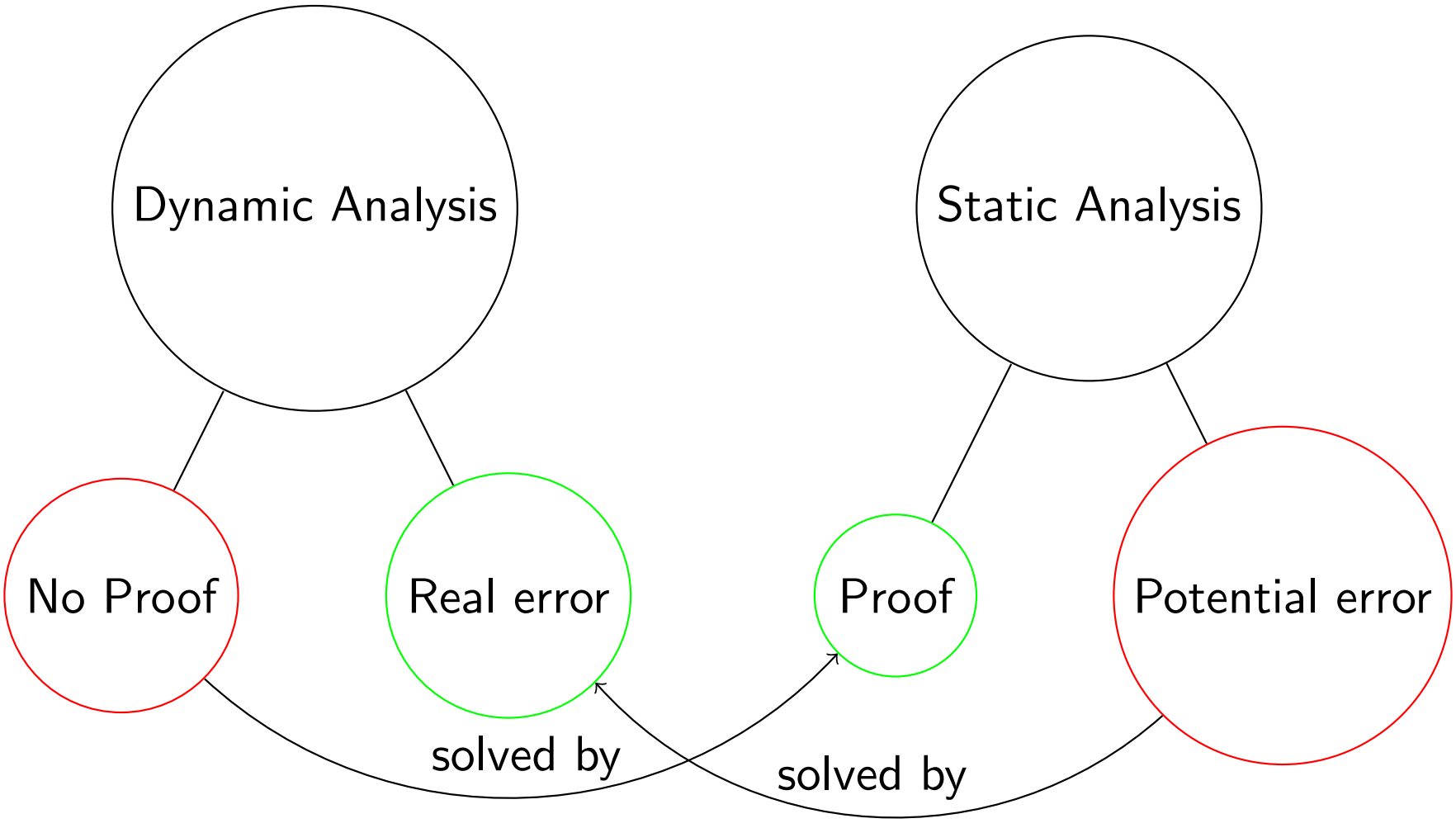

Dynamic Analysis
No Proof
Real error


Static Analysis
Proof
Potential error

Motivation


Dynamic Analysis
Static Analysis
No Proof
Real error
Proof
Potential error
solved by

Motivation


Dynamic Analysis
Static Analysis
No Proof
Real error
Proof
Potential error
solved by
solved by

Motivation

$\Downarrow$

Idea: Combine both approaches.

Table of Contents

Example

Algorithm: foo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```

(foo algorithm, G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Example

Algorithm: foo(3, 0)

1 $int * px = NULL$; `// (pc=I, x=3, y=0, px=NULL)`

Example

Algorithm: foo(3, 0)

I	$int * px = NULL;$	`// (pc=I, x=3, y=0, px=NULL)`
A	$x = x + 1;$	`// (pc=A, x=3, y=0, px=NULL)`

Example

Algorithm: foo(3, 0)

```
I  int * px = NULL;                  // (pc=I, x=3, y=0, px=NULL)
A  x = x + 1;                        // (pc=A, x=3, y=0, px=NULL)
B  if x < 4                          // (pc=B, x=4, y=0, px=NULL)
   then
C  |   px = &x;
```

Example

Algorithm: foo(3, 0)

```
I  int * px = NULL;                    // (pc=I, x=3, y=0, px=NULL)
A  x = x + 1;                          // (pc=A, x=3, y=0, px=NULL)
B  if x < 4                            // (pc=B, x=4, y=0, px=NULL)
   then
C  |   px = &x;
D  if px == &y                         // (pc=D, x=4, y=0, px=NULL)
   then
E  |   x = x + 1;                                  // Dead code
```

Example

Algorithm: foo(3, 0)

```
I  int * px = NULL;                  // (pc=I, x=3, y=0, px=NULL)
A  x = x + 1;                        // (pc=A, x=3, y=0, px=NULL)
B  if x < 4                          // (pc=B, x=4, y=0, px=NULL)
   then
C  |   px = &x;
D  if px == &y                       // (pc=D, x=4, y=0, px=NULL)
   then
E  |   x = x + 1;                                  // Dead code
F  if x < 5                          // (pc=F, x=4, y=0, px=NULL)
   then
G  |   *px = *px + 1;                // (pc=G, x=4, y=0, px=NULL)
```

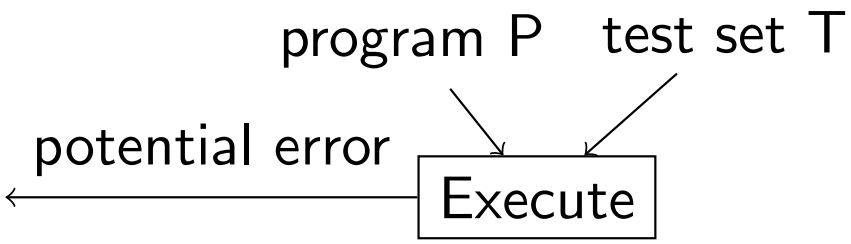
Example

Algorithm: foo(3, 0)

```
I  int * px = NULL;                  // (pc=I, x=3, y=0, px=NULL)
A  x = x + 1;                        // (pc=A, x=3, y=0, px=NULL)
B  if x < 4                          // (pc=B, x=4, y=0, px=NULL)
   then
C  |   px = &x;
D  if px == &y                       // (pc=D, x=4, y=0, px=NULL)
   then
E  |   x = x + 1;                                  // Dead code
F  if x < 5                          // (pc=F, x=4, y=0, px=NULL)
   then
G  |   *px = *px + 1;                // (pc=G, x=4, y=0, px=NULL)
```

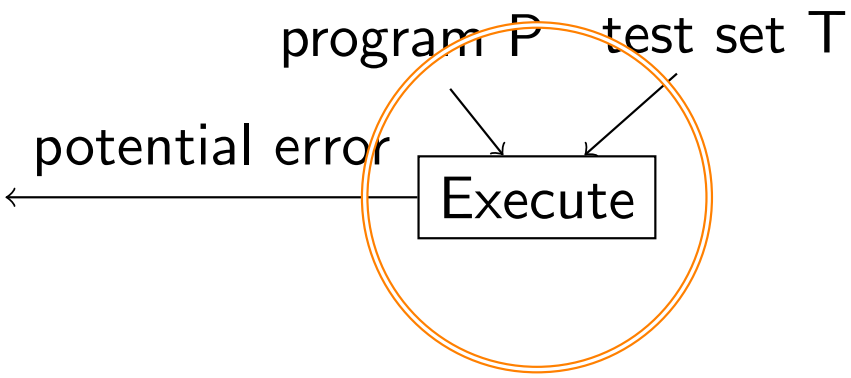
$\Longrightarrow$ Null pointer dereference error

Execute



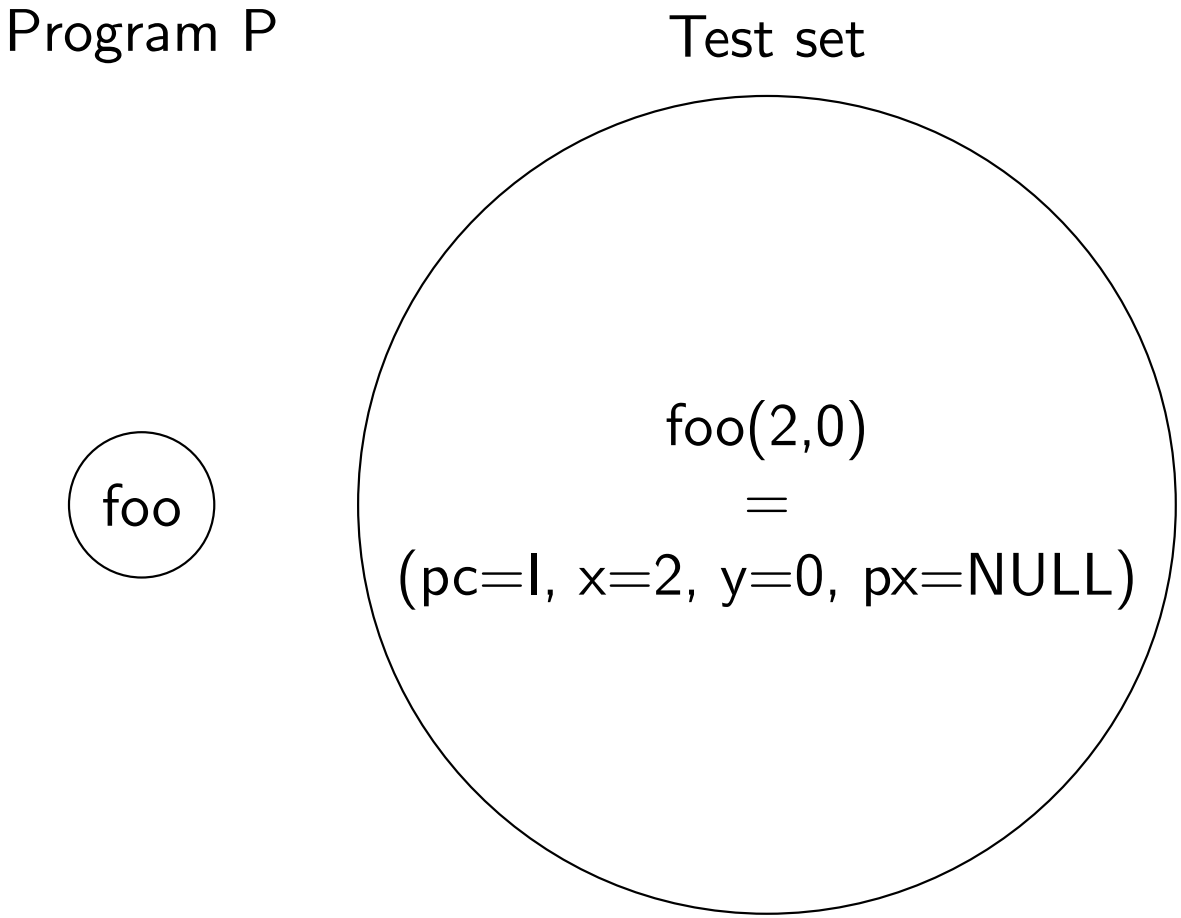
(Figure 1, G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Execute



(Figure 1, G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Execute



An example input.

Execute

Algorithm: foo(2, 0)

1 $int * px = NULL$; `// (pc=I, x=2, y=0, px=NULL)`

Execute

Algorithm: foo(2, 0)

I $int * px = NULL;$ `// (pc=I, x=2, y=0, px=NULL)`

A $x = x + 1;$ `// (pc=A, x=2, y=0, px=NULL)`

Execute

Algorithm: foo(2, 0)

```
I   int * px = NULL;          // (pc=I, x=2, y=0, px=NULL)
A   x = x + 1;                // (pc=A, x=2, y=0, px=NULL)
B   if x < 4                  // (pc=B, x=3, y=0, px=NULL)
    then
C   |   px = &x;              // (pc=C, x=3, y=0, px=NULL)
```

Execute

Algorithm: foo(2, 0)

I	$int * px = NULL;$	// (pc=I, x=2, y=0, px=NULL)
A	$x = x + 1;$	// (pc=A, x=2, y=0, px=NULL)
B	**if** $x < 4$	// (pc=B, x=3, y=0, px=NULL)
	then	
C	$\quad px = \&x;$	// (pc=C, x=3, y=0, px=NULL)
D	**if** $px == \&y$	// (pc=D, x=3, y=0, px=¬NULL)
	then	
E	$\quad x = x + 1;$	

Execute

Algorithm: foo(2, 0)

```
I  int * px = NULL;                    // (pc=I, x=2, y=0, px=NULL)
A  x = x + 1;                          // (pc=A, x=2, y=0, px=NULL)
B  if x < 4                            // (pc=B, x=3, y=0, px=NULL)
   then
C  |   px = &x;                        // (pc=C, x=3, y=0, px=NULL)
D  if px == &y                         // (pc=D, x=3, y=0, px=¬NULL)
   then
E  |   x = x + 1;
F  if x < 5                            // (pc=F, x=3, y=0, px=¬NULL)
   then
G  |   *px = *px + 1;                  // (pc=G, x=3, y=0, px=¬NULL)
```

Execute

Execute test set $\Rightarrow$ No Error.

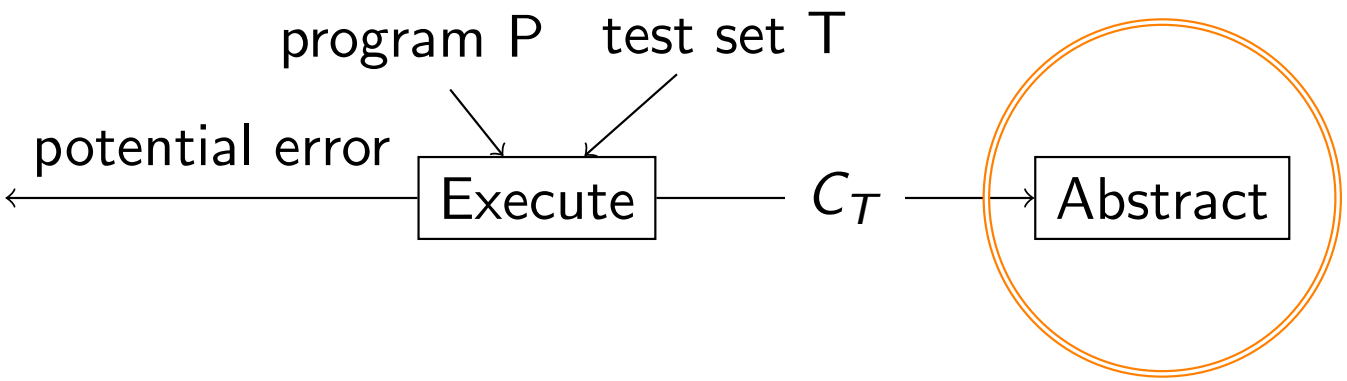
but

Execute foo(3,0) $\Rightarrow$ Error

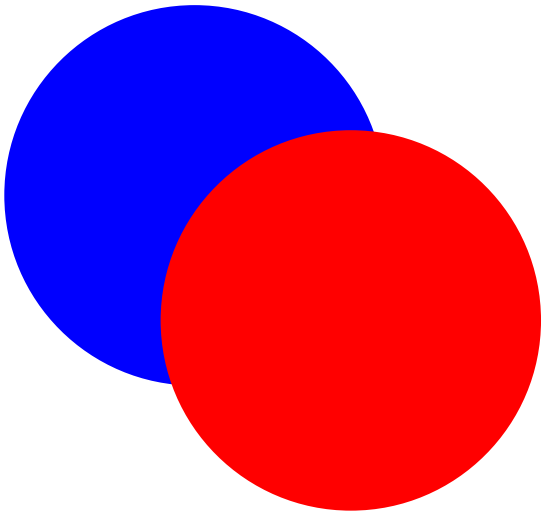
$\Downarrow$

Execute phase gives no proof.

Abstract



Abstract



Unreachable concrete states

Reachable concrete states

Abstract state =  +

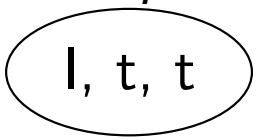
Abstract

$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$

C_T

pc=l, x=2, y=0, px=NULL

A_T



Algorithm: foo(2,0)

1 $int * px = NULL;$

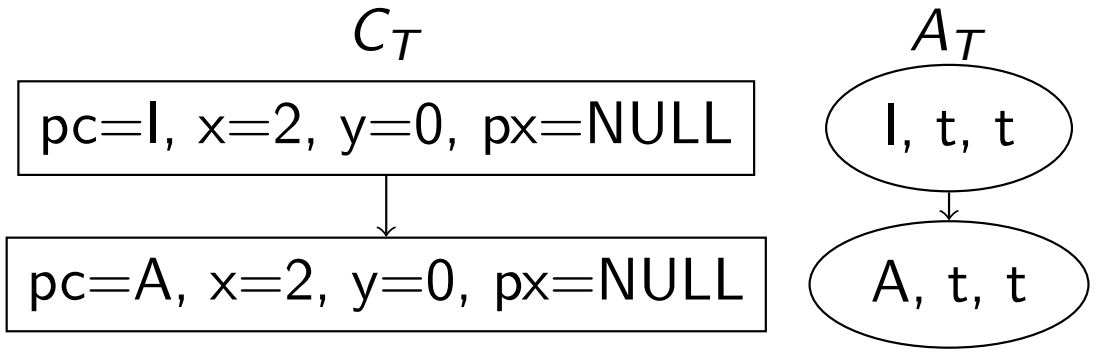
Abstract

$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$

Algorithm: foo(2,0)

I $int * px = NULL;$

A $x = x + 1;$



Abstract

$$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$$

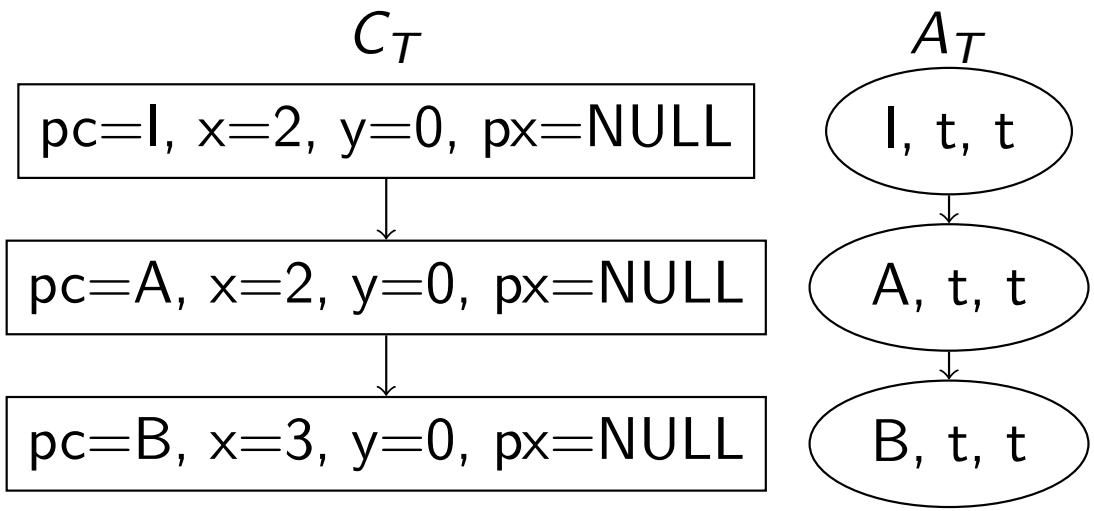
Algorithm: foo(2,0)

I $int * px = NULL;$

A $x = x + 1;$

B **if** $x < 4$ **then**

C $\quad px = \&x;$



Abstract

$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$

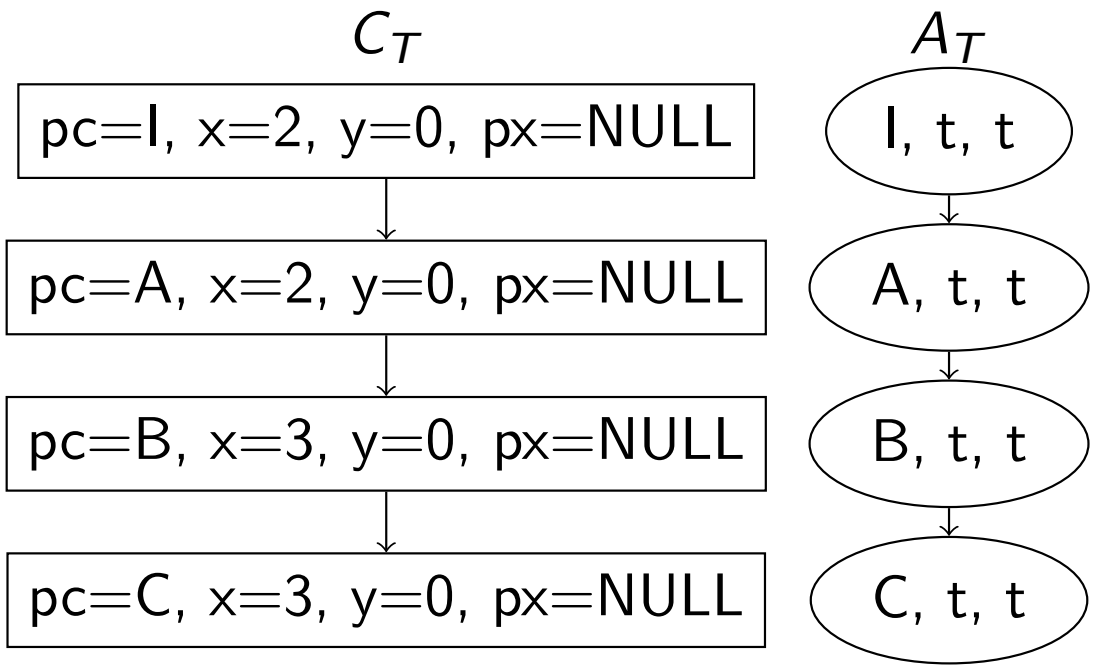
Algorithm: foo(2,0)

I $int * px = NULL;$

A $x = x + 1;$

B **if** $x < 4$ **then**

C $\quad px = \&x;$

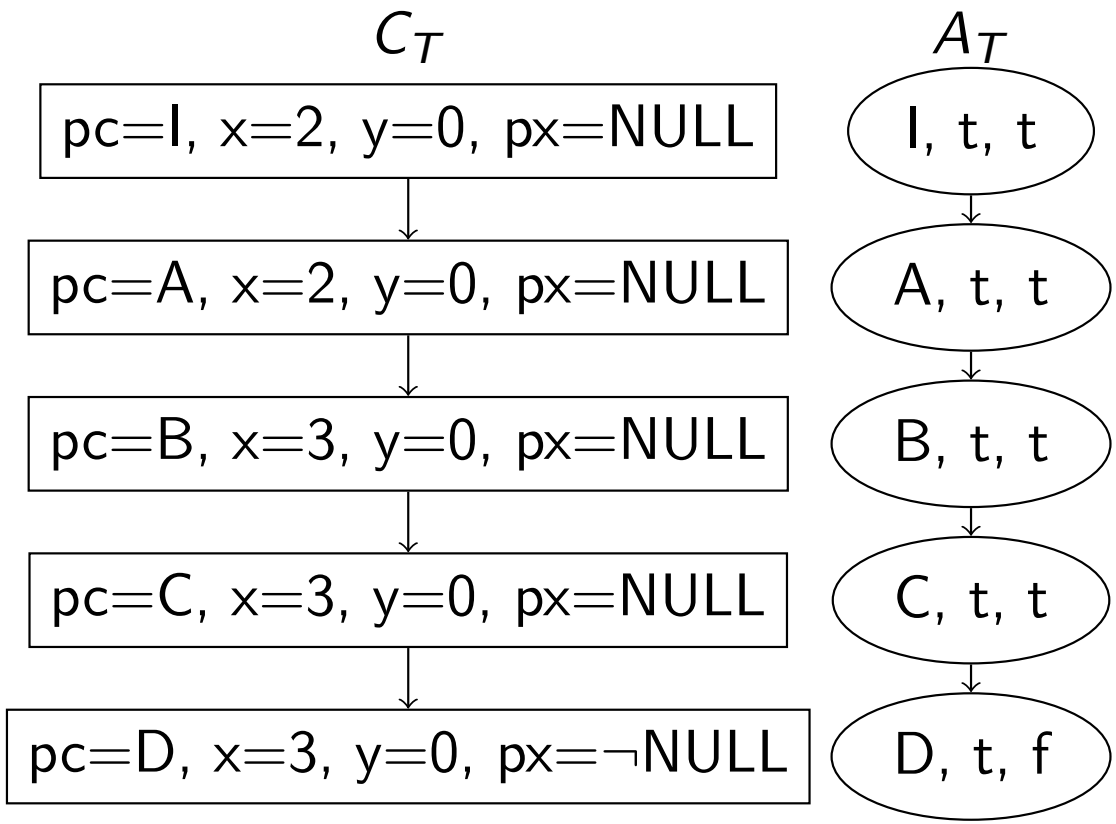


Abstract

$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$

Algorithm: foo(2,0)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
```

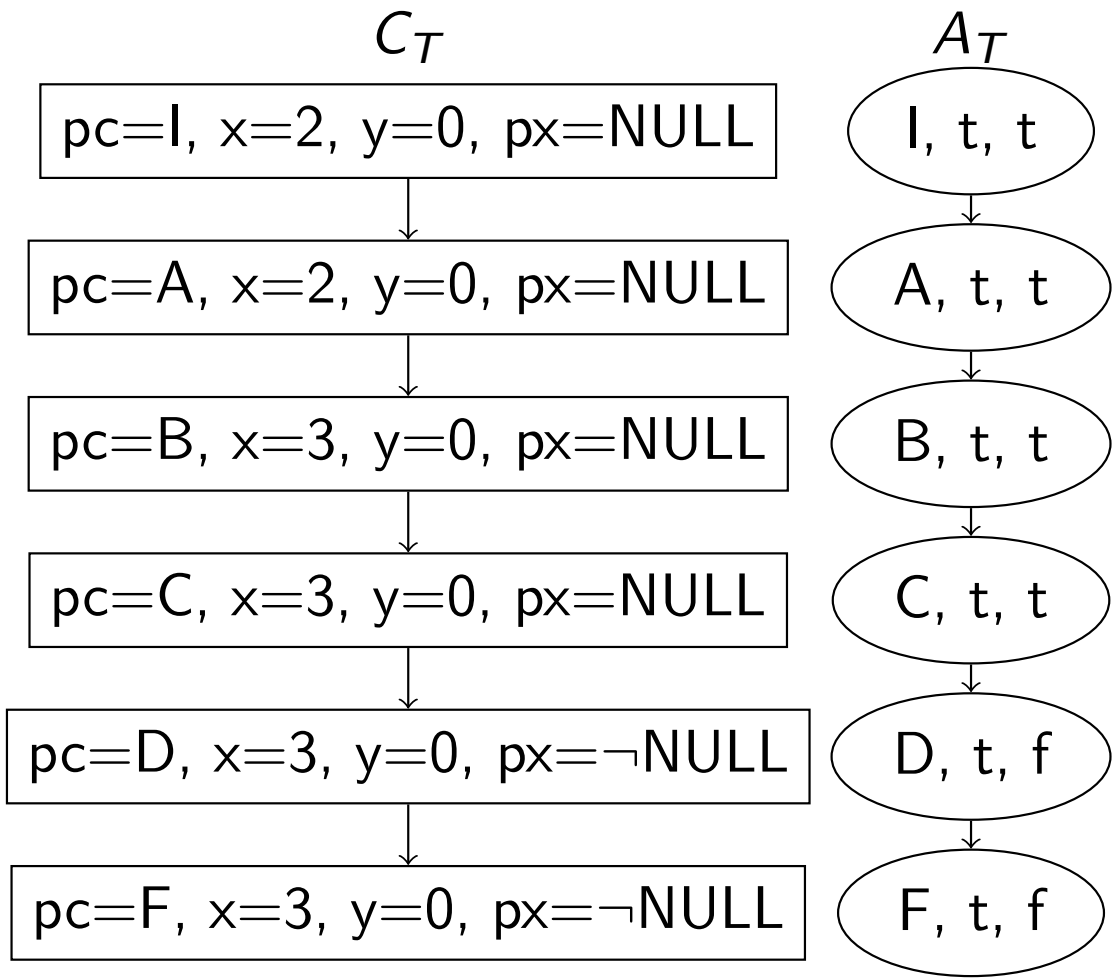


Abstract

$$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$$

Algorithm: foo(2,0)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px+1;
```

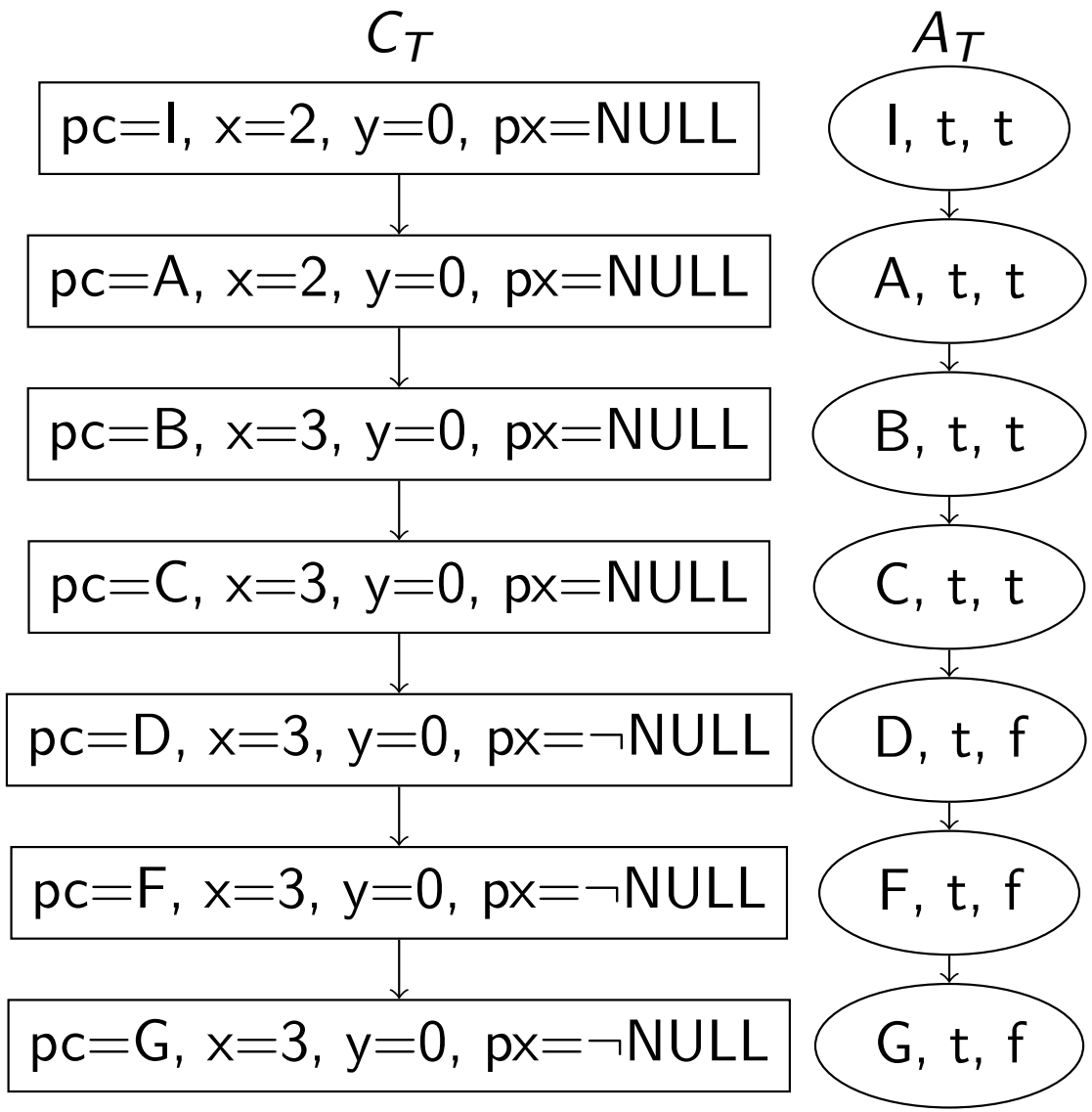


Abstract

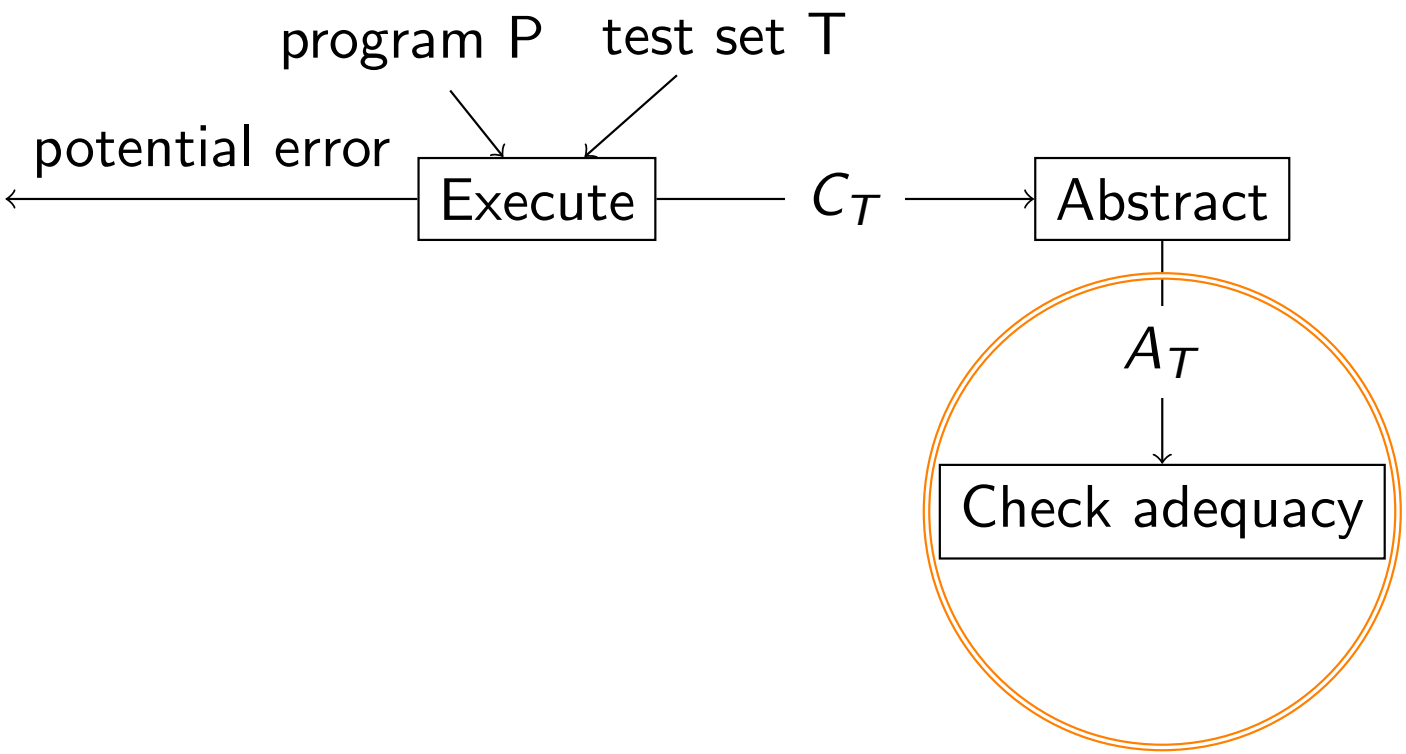
$$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$$

Algorithm: foo(2,0)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px+1;
```



Check adequacy



Check adequacy

What are **successor states** ? Let's look at our example.

Algorithm: part of foo(int x, int y)

```
A  x = x + 1;
B  if x < 4 then
C  |   px = &x;                          // If B is true.
D  if px == &y                           // If B is false.
   then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```

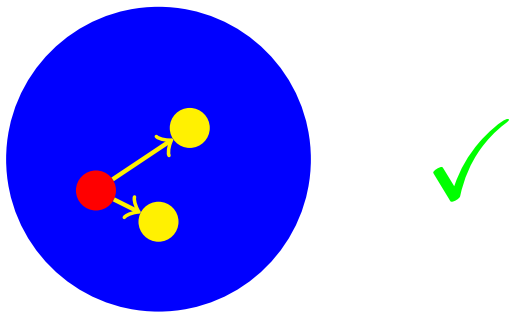
Successor states of B

Check adequacy

A set of tests T is adequate under a given abstraction

$\Leftrightarrow$

for all concrete states which are represented by A_T it holds that their successor states are covered by A_T, too.



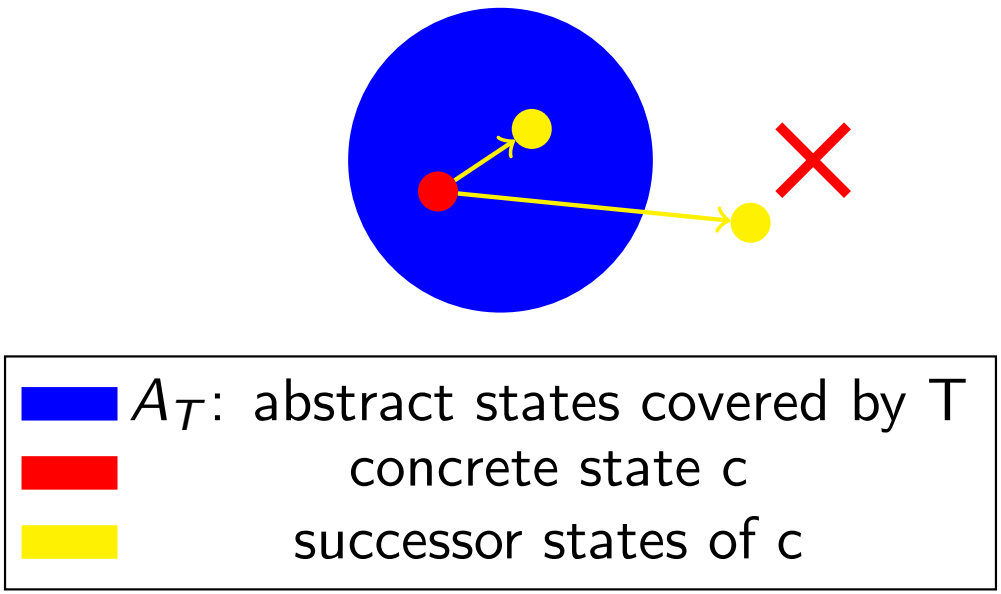
	A_T: abstract states covered by T
	concrete state c
	successor states of c

Check adequacy

A set of tests T is adequate under a given abstraction

$\Leftrightarrow$

for all concrete states which are represented by A_T it holds that their successor states are covered by A_T, too.

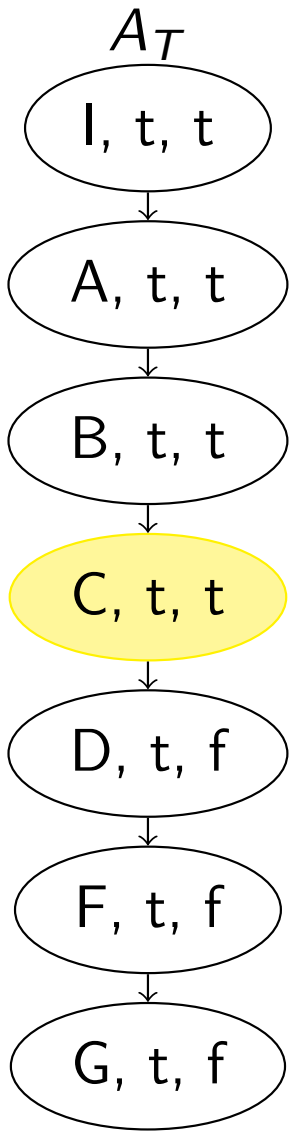


Check adequacy

$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$

Algorithm: foo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4 then
C  |    px = &x;          // (pc=C, x<4, px=Null)
D  if px == &y
   then
E  |    x = x + 1;
F  if x < 5 then
G  |    *px = *px + 1;
```

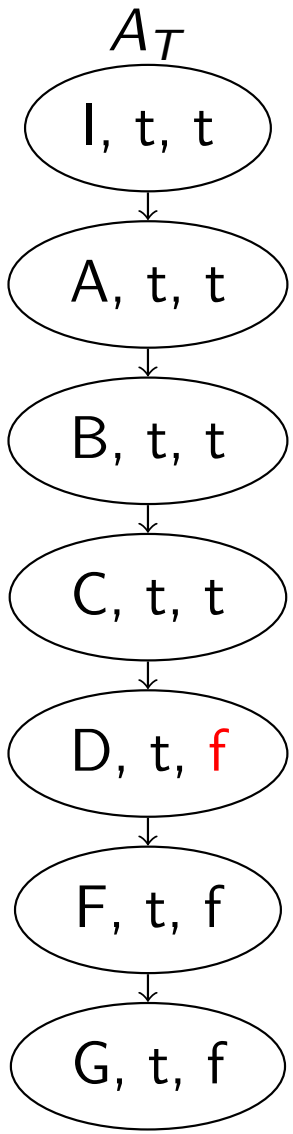


Check adequacy

$\alpha(C) = \{(pc, x < 5, px = NULL) | (pc, x, y, px) \in C\}$

Algorithm: foo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4 then
C  |   px = &x;
D  if px == &y              // (pc=D, x>=4, px=NULL)
   then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```



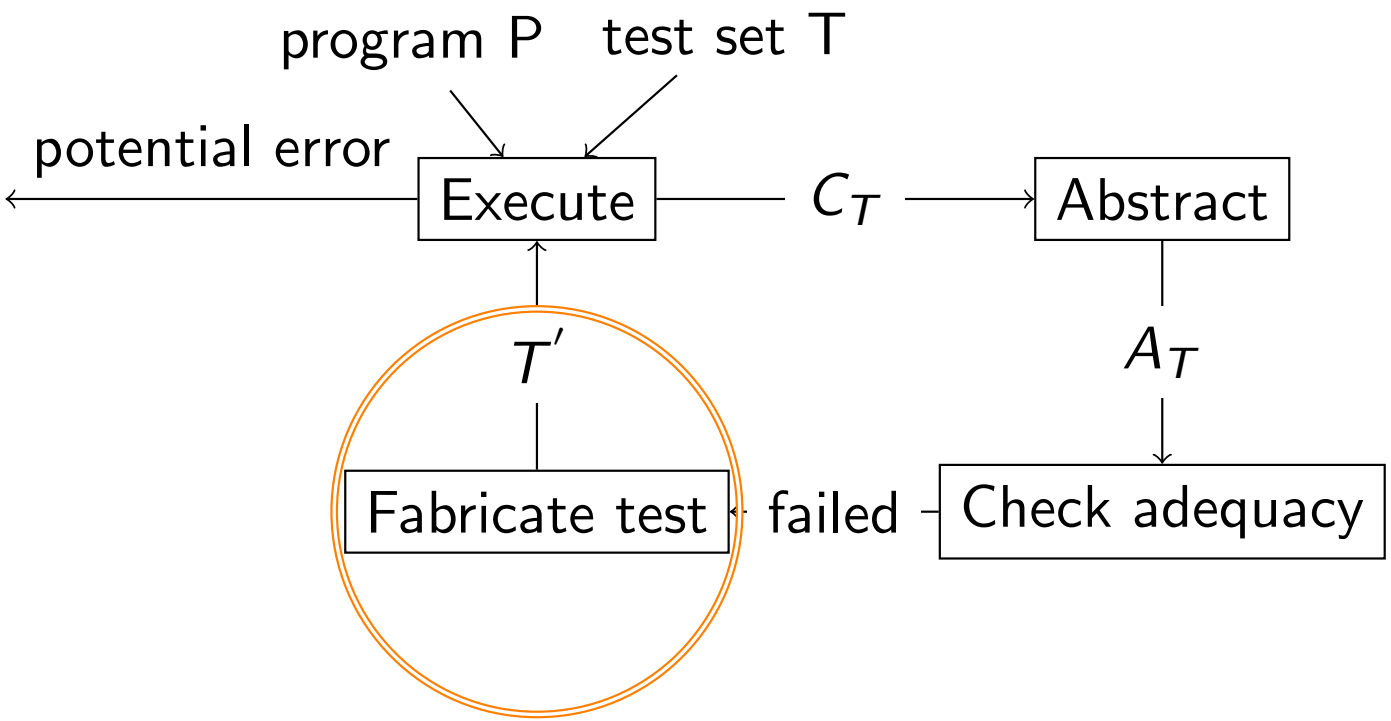
Check adequacy

The successor state (D,4,0,NULL) is not covered by A_T

$\Downarrow$

adequacy check fails.

Fabricate test



Fabricate test

The successor state (D,4,0,NULL) is not covered by A_T

$\Downarrow$

adequacy check fails.

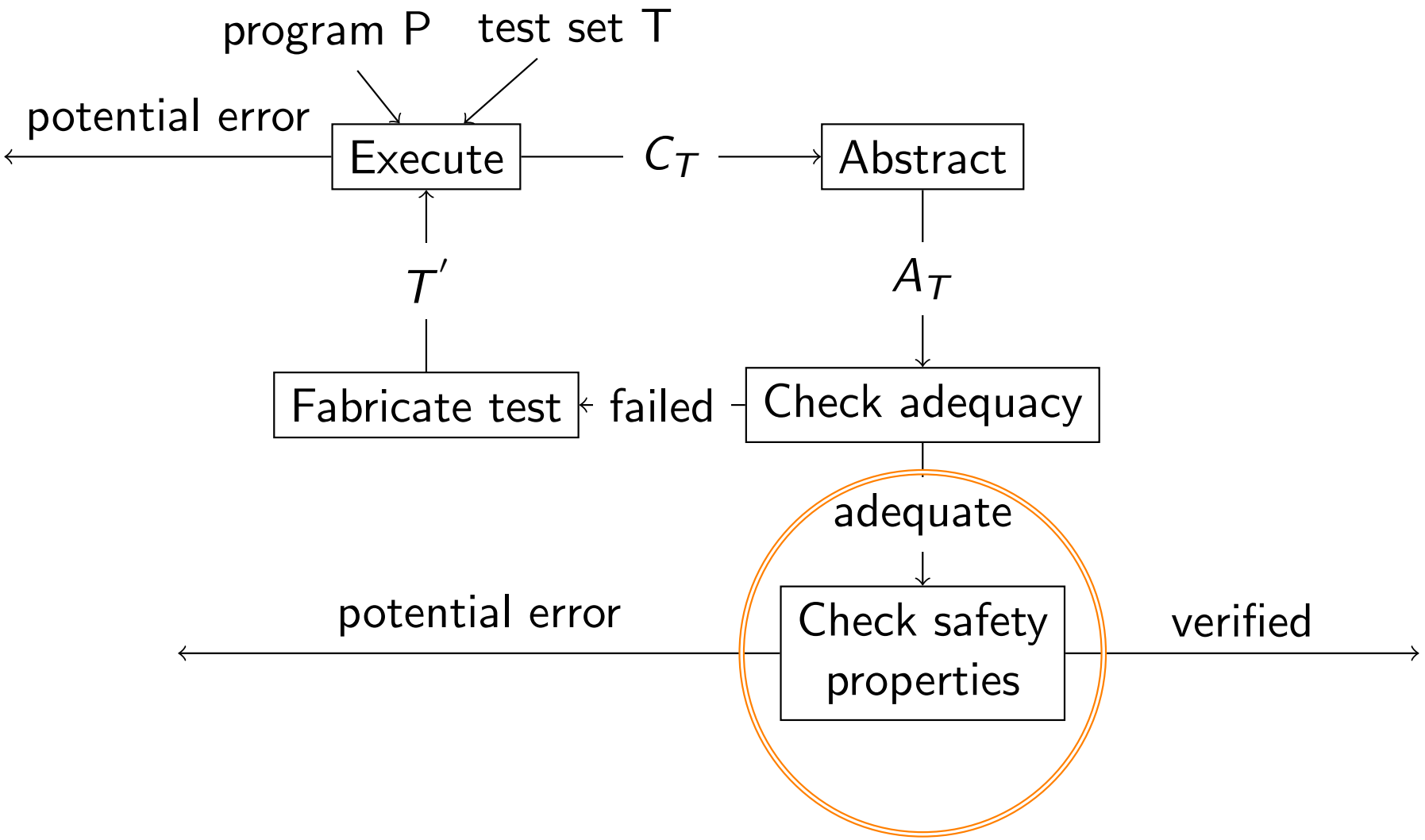
$\Downarrow$

model generator fabricates a pair of concrete states, e.g.
(B,4,0,NULL), (D,4,0,NULL)
concrete states need not to be reachable.

$\Downarrow$

extend test set T by the generated concrete states.

Check safety property



Check safety property

What is a **safety property** ? Let's look at our example.

Algorithm: part of foo(int x, int y)

A $x = x + 1;$
B **if** $x < 4$ **then**
C $\quad px = \&x;$
D **if** $px == \&y$ **then**
E $\quad x = x + 1;$
F **if** $x < 5$ **then**
G $\quad *px = *px + 1;$ `// error if px = NULL`

$\Rightarrow$ safety property: abstract error state (G,t,t) $\notin A_T$

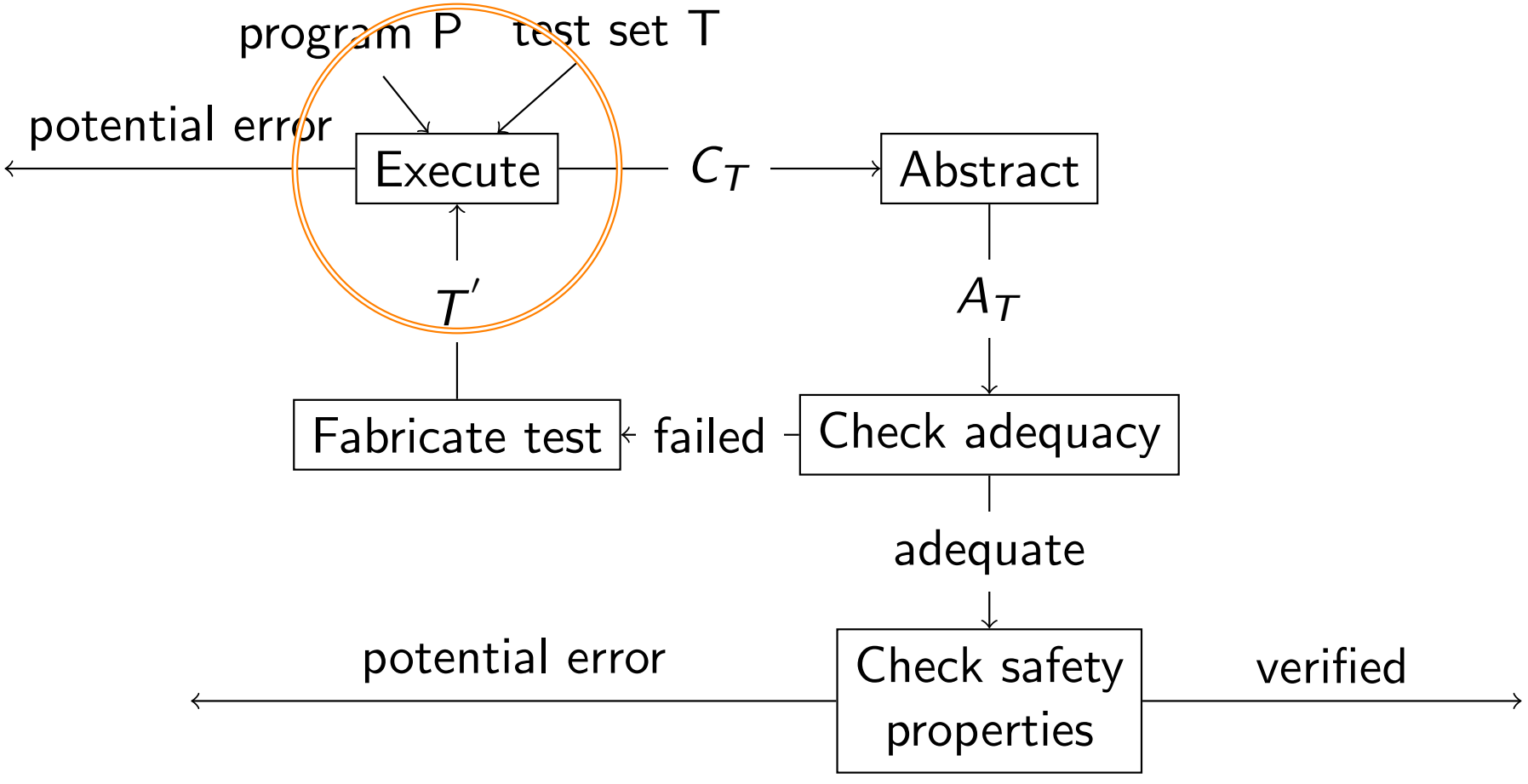
Table of Contents

Input

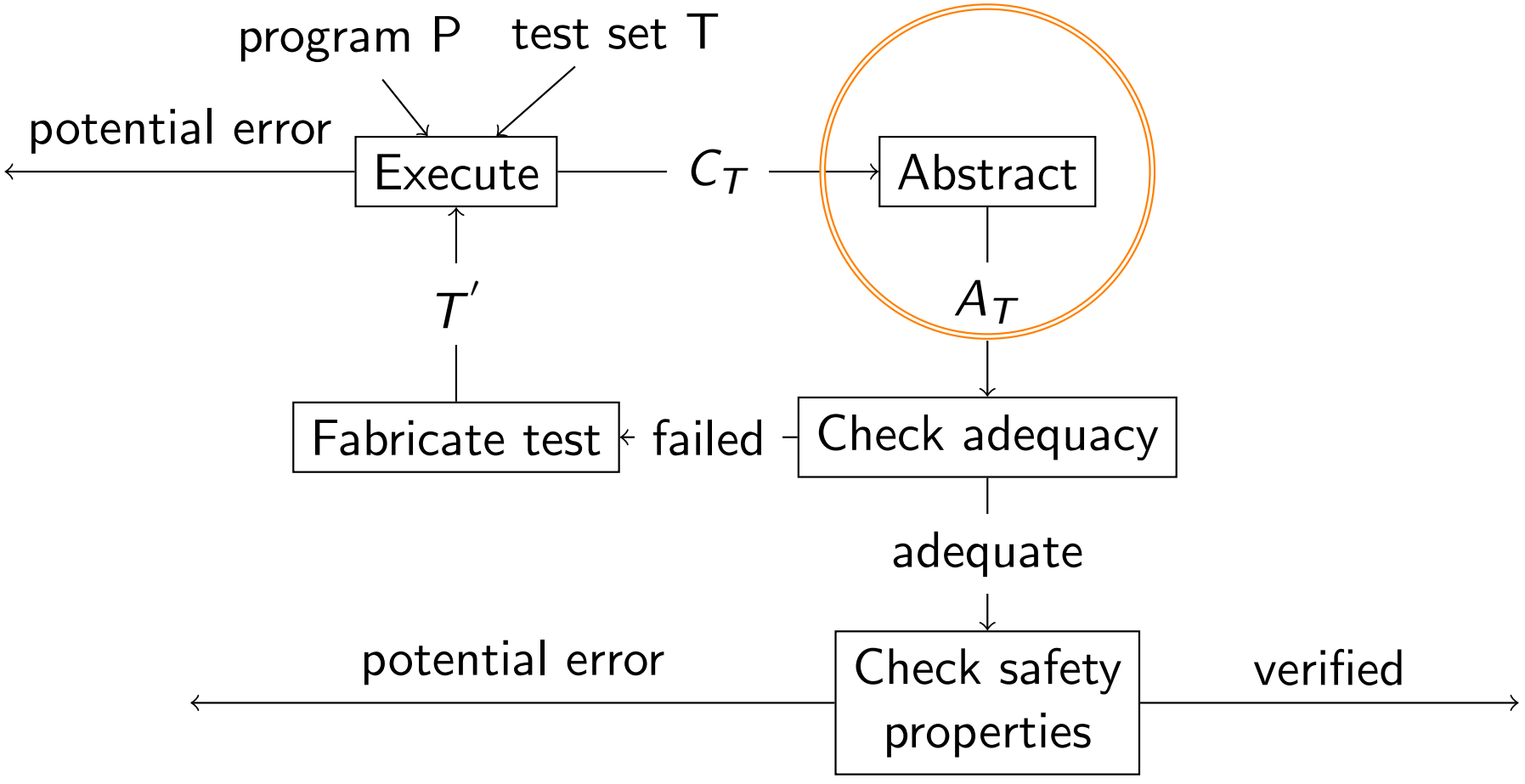
- Program P
  foo
- Test set T

$$\{\underbrace{(pc = l, x = 2, y = 0, px = NULL)}_{fixedFoo(2,0)},$$
$$\underbrace{(pc = l, x = 6, y = 0, px = NULL)}_{fixedFoo(6,0)},$$
$$\underbrace{(pc = l, x = 11, y = 0, px = NULL)}_{fixedFoo(11,0)}\}$$

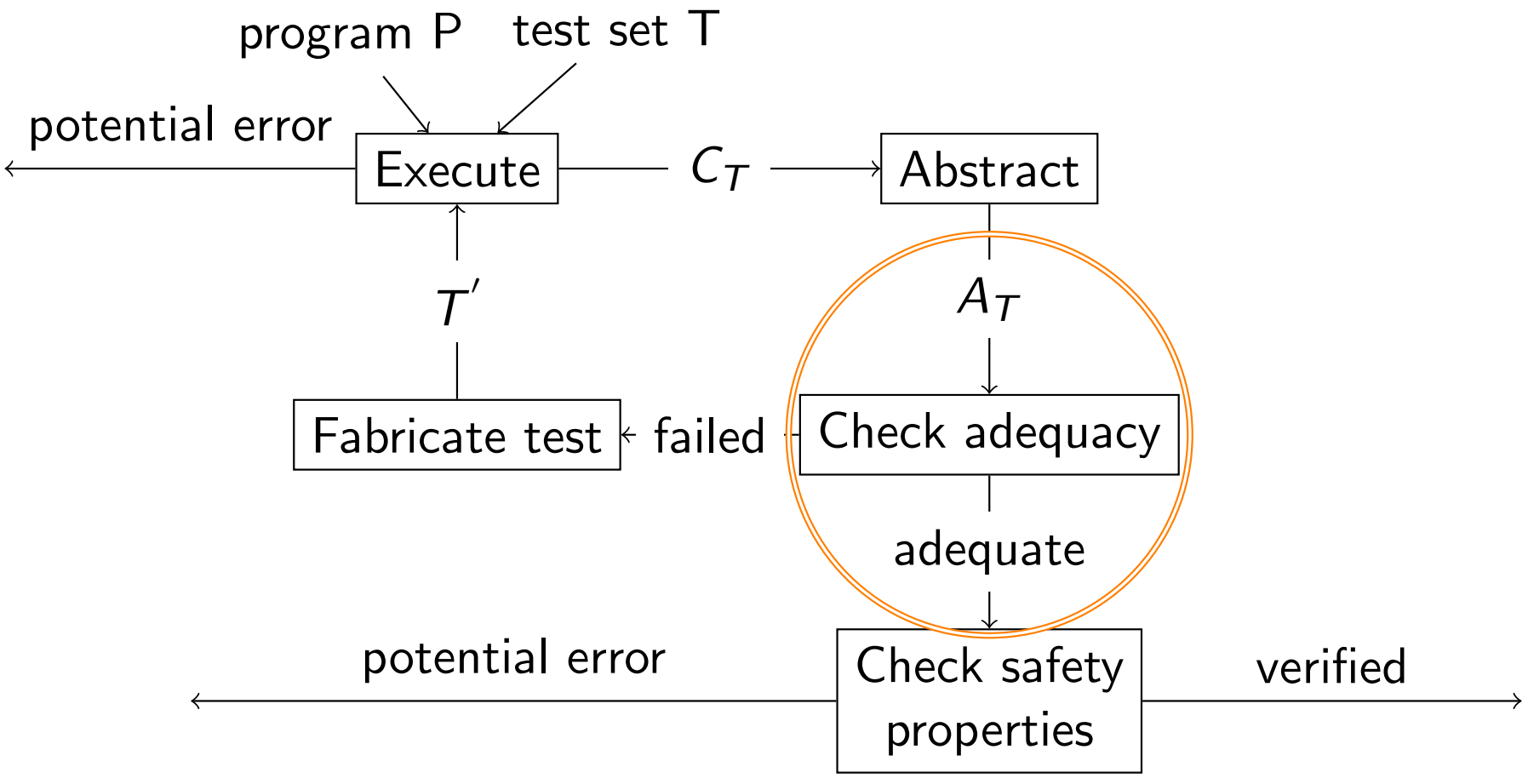
Run until check adequacy



Run until check adequacy

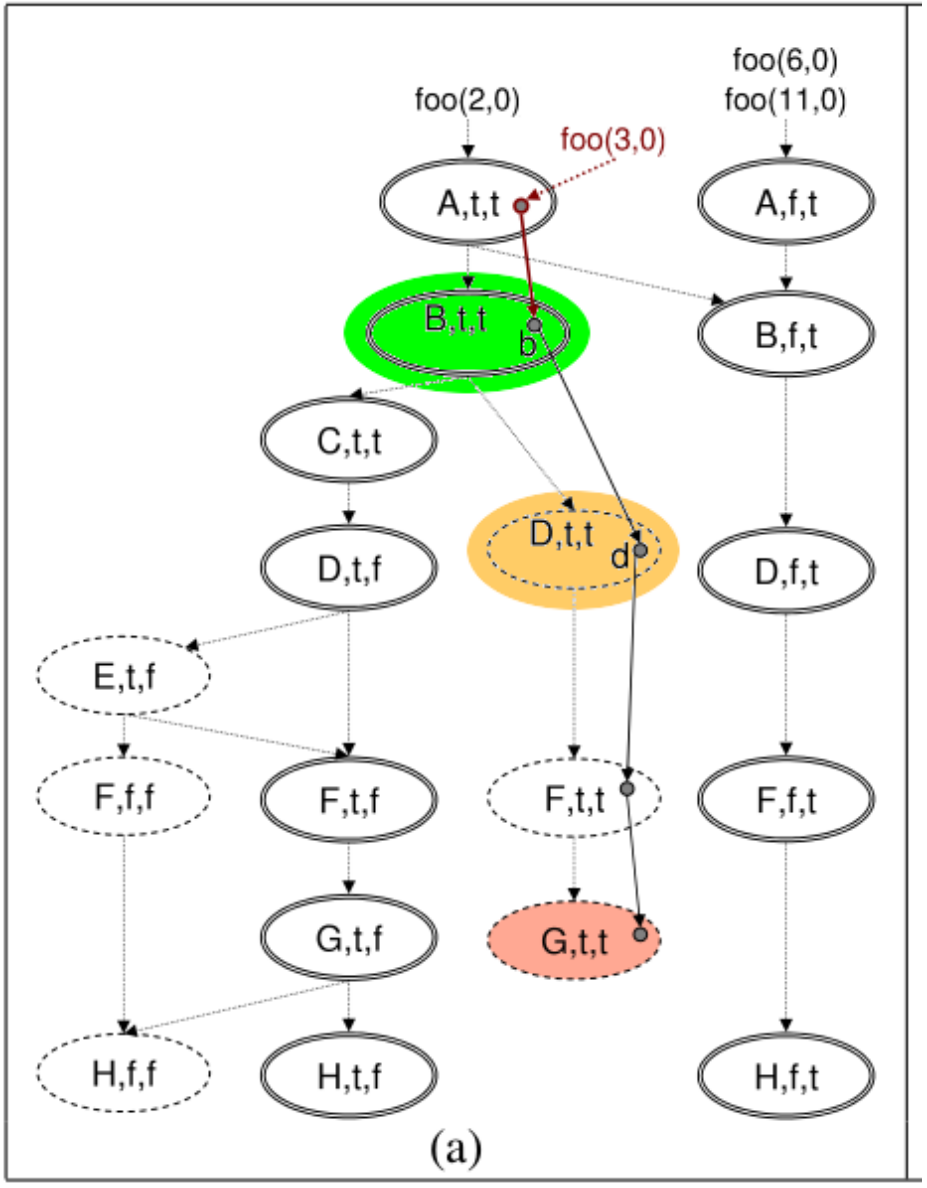


Run until check adequacy



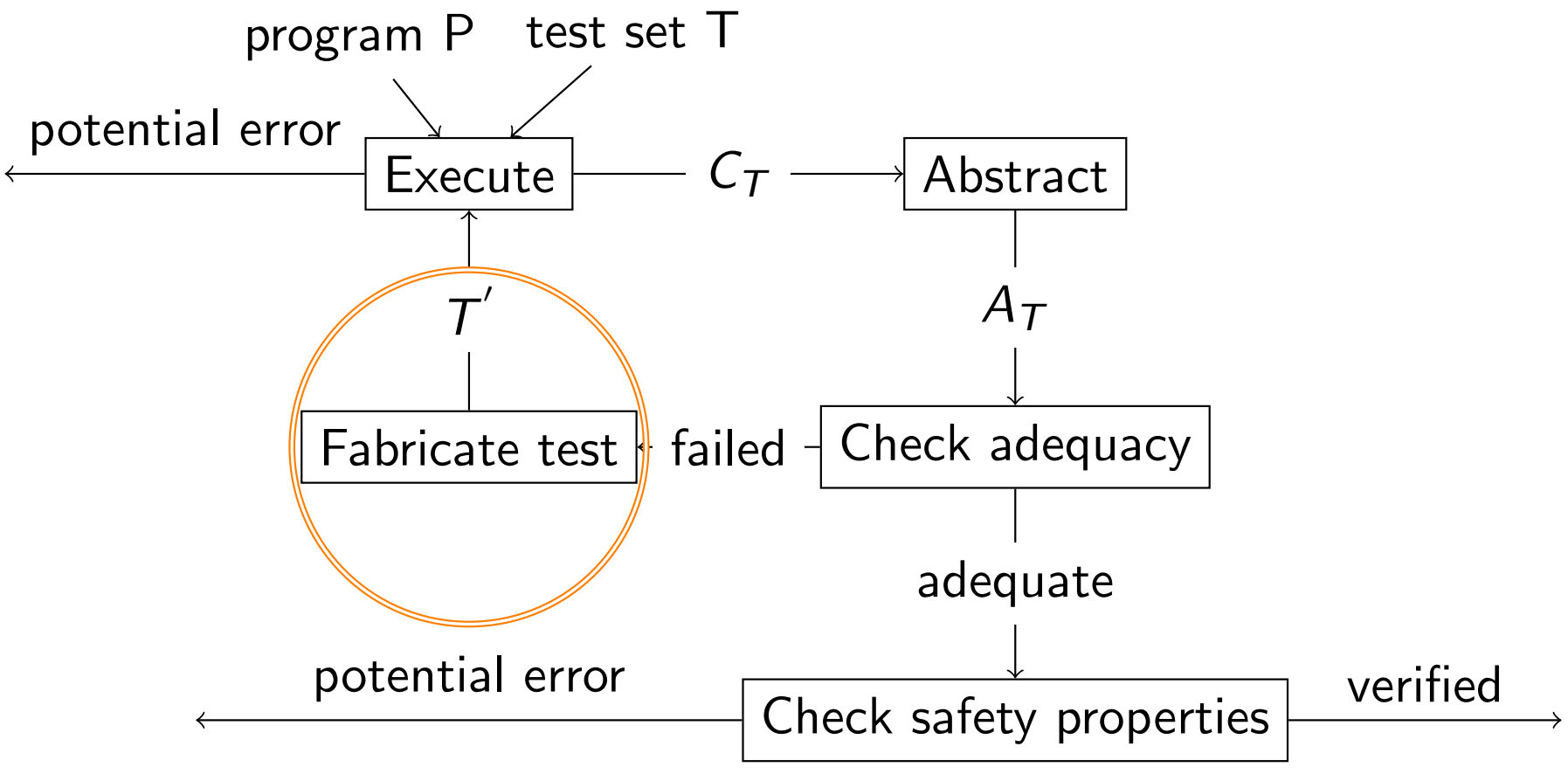
Algorithm: foo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```



(Figure 2(a), G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Fabricate test



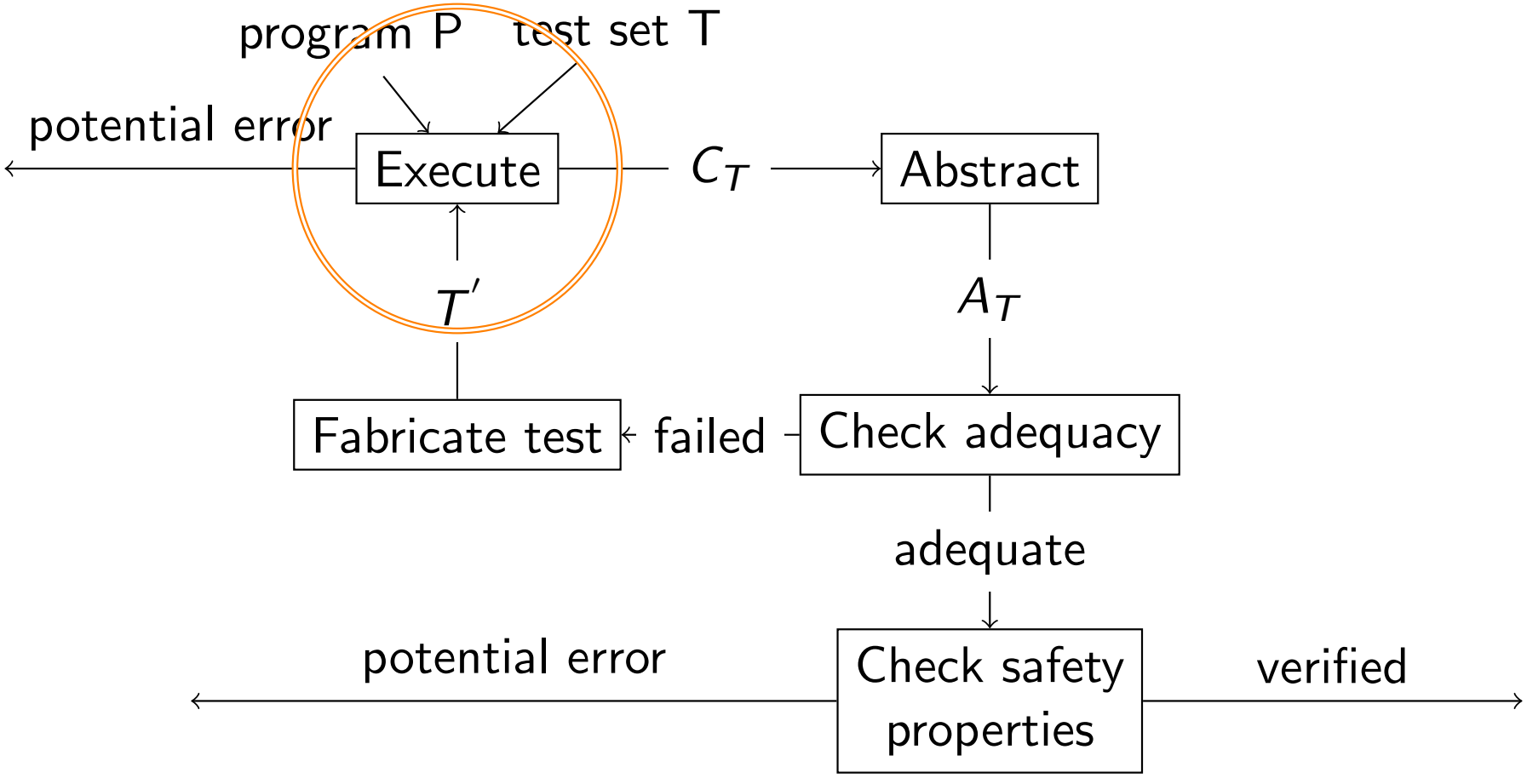
Fabricate test

Algorithm: fixedFoo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 5 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```

Fabricate tests
(pc=B, x=4, y=0, px=NULL)
(pc=D, x=4, y=0, px=NULL)
Remark: Both tests are reachable.

Execute


program P
test set T
potential error
Execute
C_T
Abstract
T'
A_T
Fabricate test
failed
Check adequacy
adequate
potential error
Check safety properties
verified

Execute (pc=B, x=4, y=0, px=NULL)

Algorithm: foo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4                          // (pc=B, x=4, y=0, px=NULL)
   then
C  |   px = &x;
D  if px == &y                       // (pc=D, x=4, y=0, px=NULL)
   then
E  |   x = x + 1;
F  if x < 5                          // (pc=F, x=4, y=0, px=NULL)
   then
G  |   *px = *px + 1;                // (pc=G, x=4, y=0, px=NULL)
```

Execute (pc=B, x=4, y=0, px=NULL)

Algorithm: foo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 4                          // (pc=B, x=4, y=0, px=NULL)
   then
C  |   px = &x;
D  if px == &y                       // (pc=D, x=4, y=0, px=NULL)
   then
E  |   x = x + 1;
F  if x < 5                          // (pc=F, x=4, y=0, px=NULL)
   then
G  |   *px = *px + 1;                // (pc=G, x=4, y=0, px=NULL)
```

$\Longrightarrow$ Null pointer dereference error

Report potential error

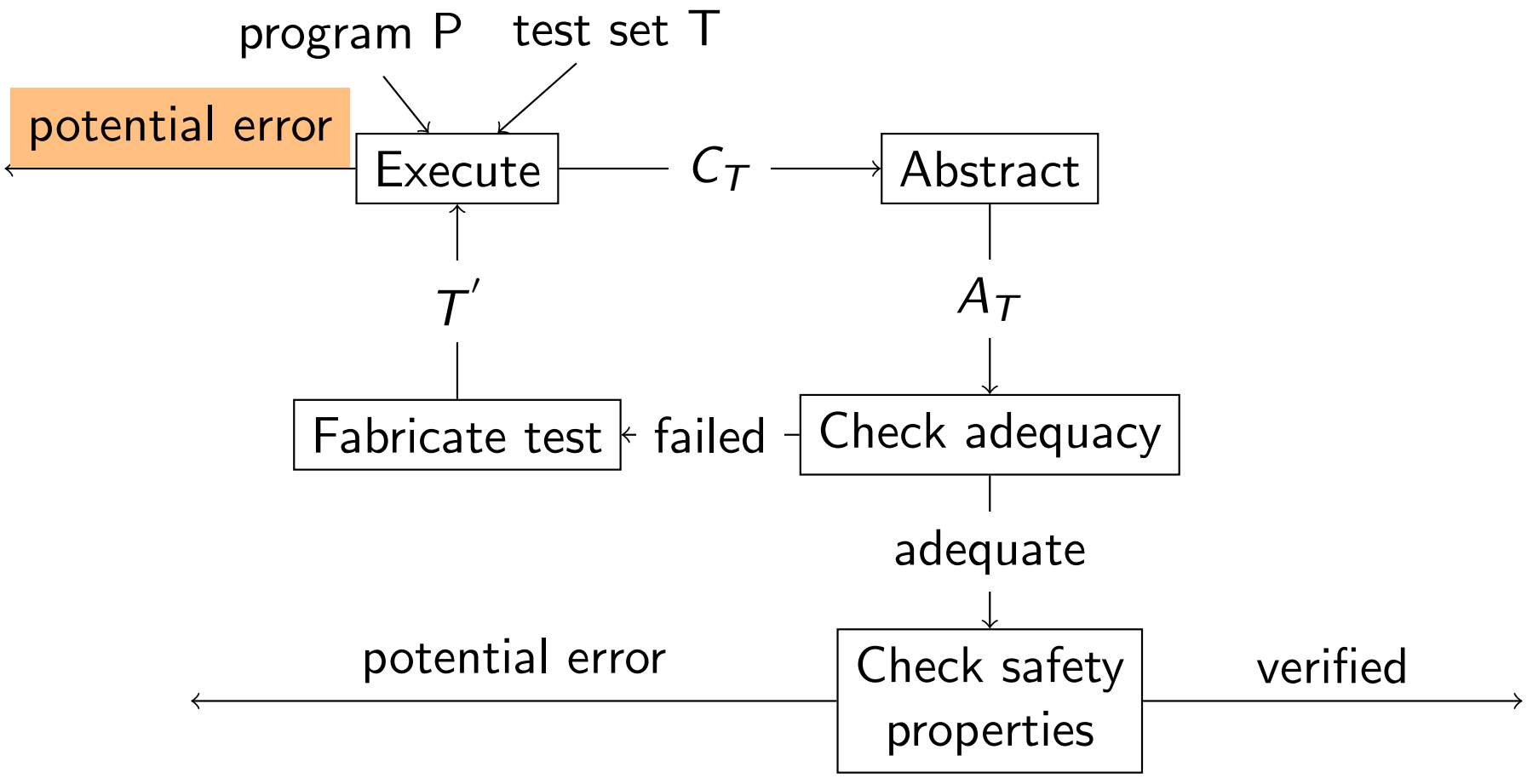


Table of Contents

Modify example foo

Algorithm: fixedFoo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 5                                   // instead of x < 4
   then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```

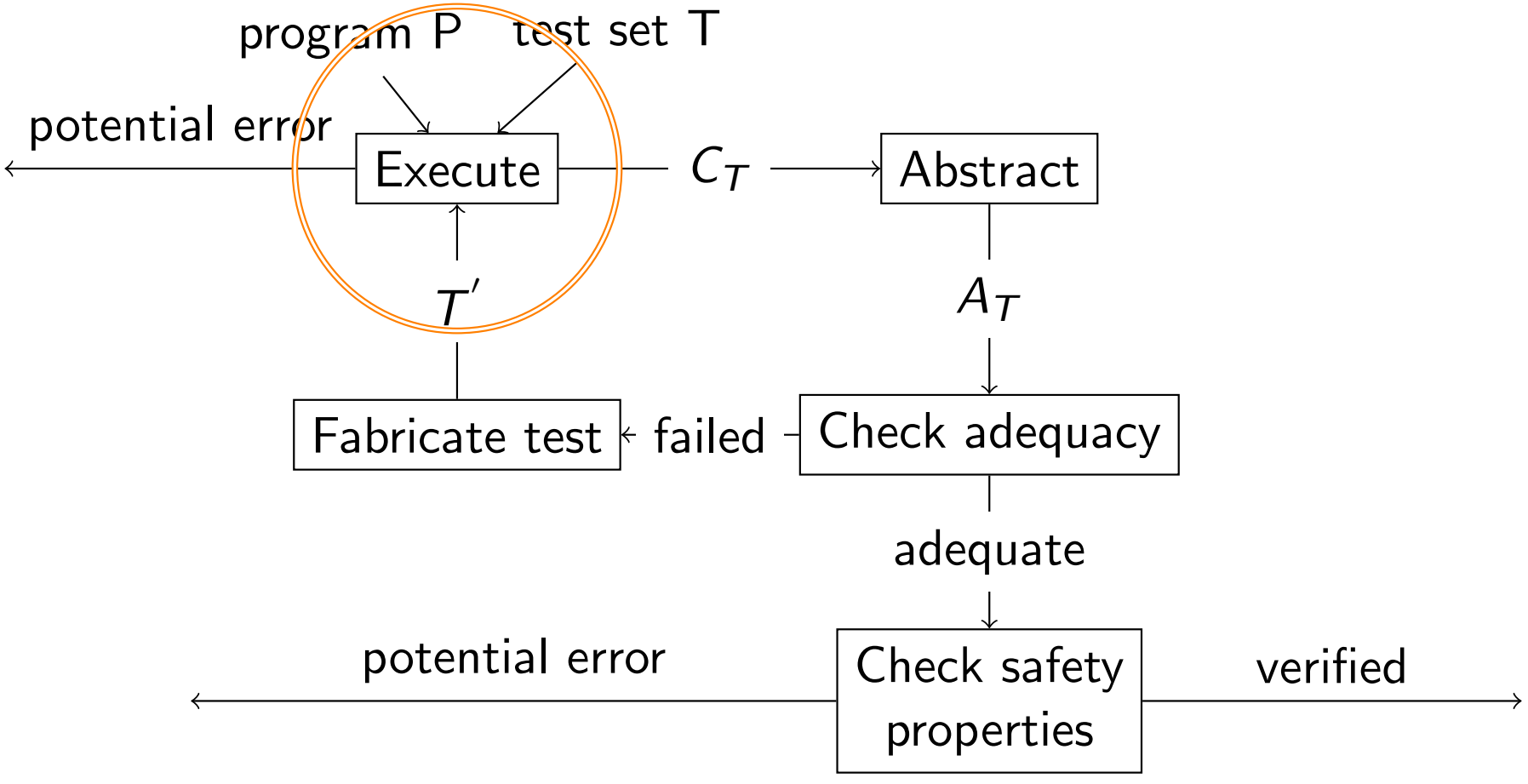
(fixed_foo algorithm, G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Input

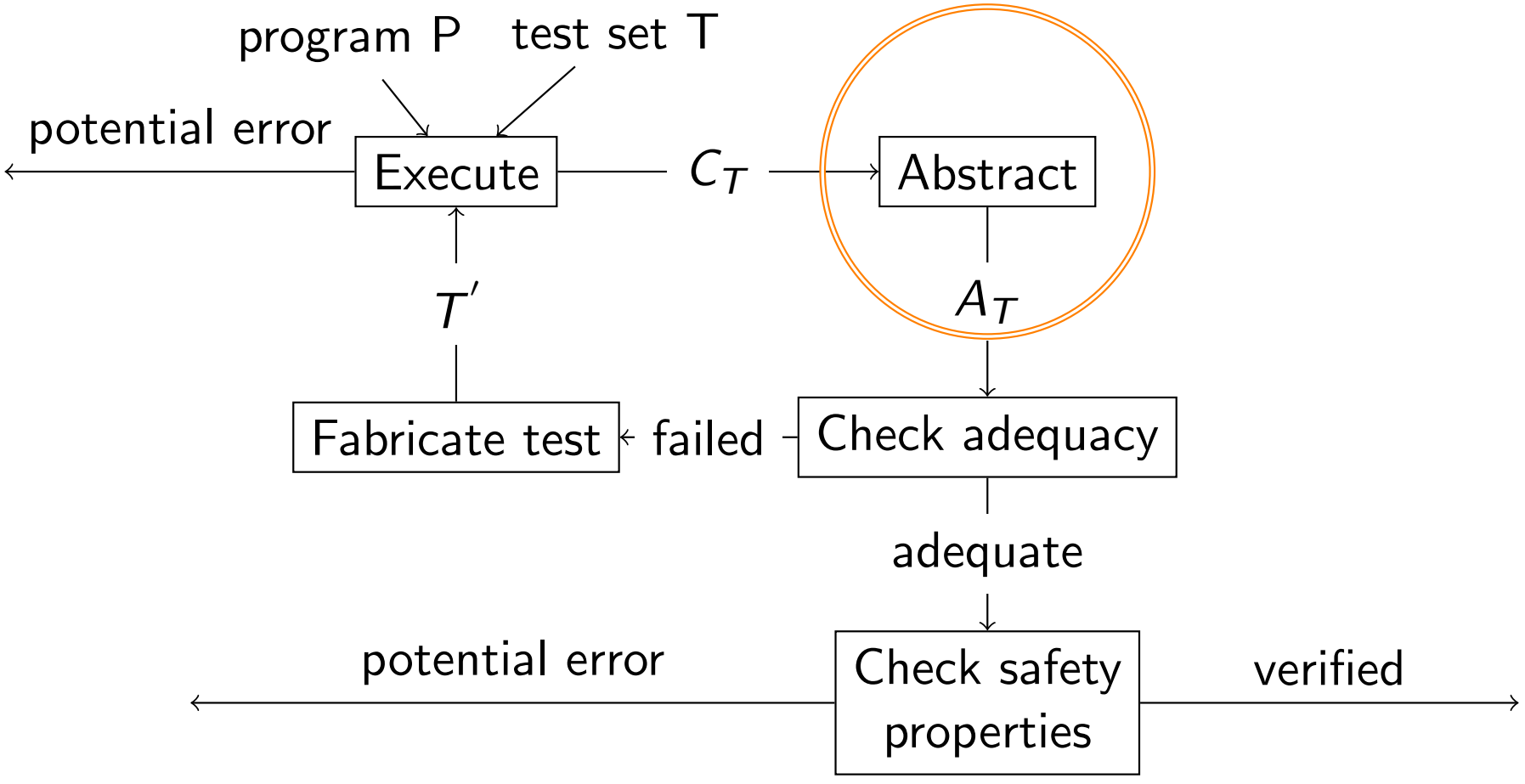
- Program P
  fixedFoo
- Test set T

$$\{\underbrace{(pc = l, x = 2, y = 0, px = NULL)}_{fixedFoo(2,0)},$$
$$\underbrace{(pc = l, x = 6, y = 0, px = NULL)}_{fixedFoo(6,0)},$$
$$\underbrace{(pc = l, x = 11, y = 0, px = NULL)}_{fixedFoo(11,0)}\}$$

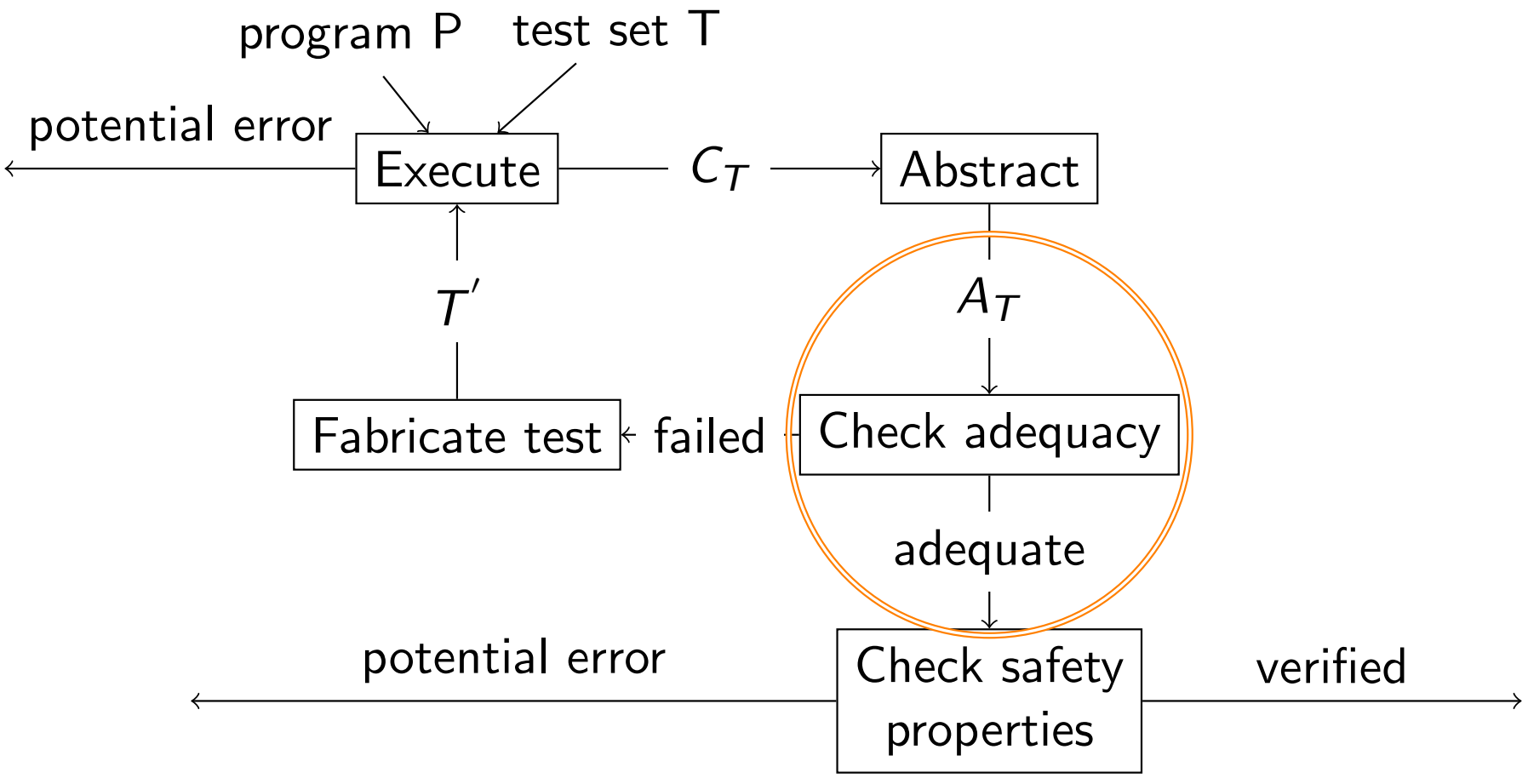
Run until check adequacy



Run until check adequacy

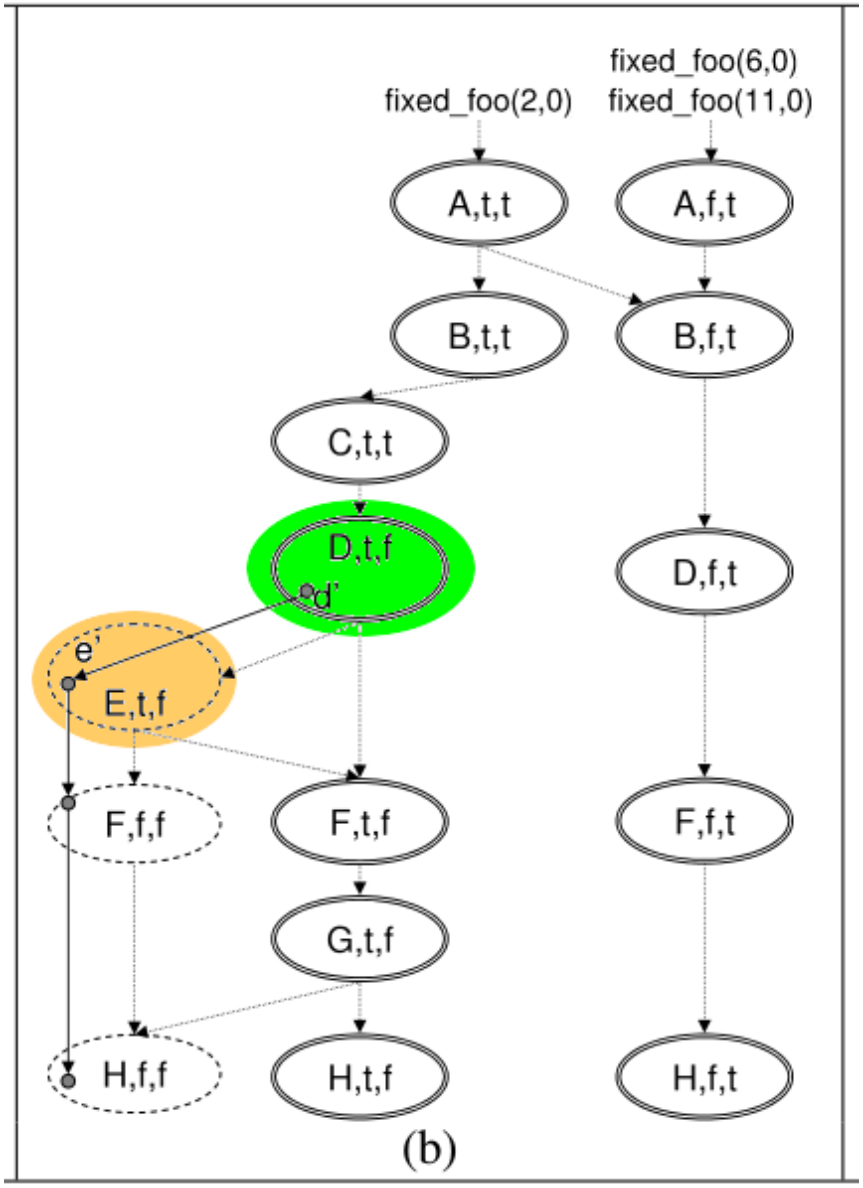


Run until check adequacy



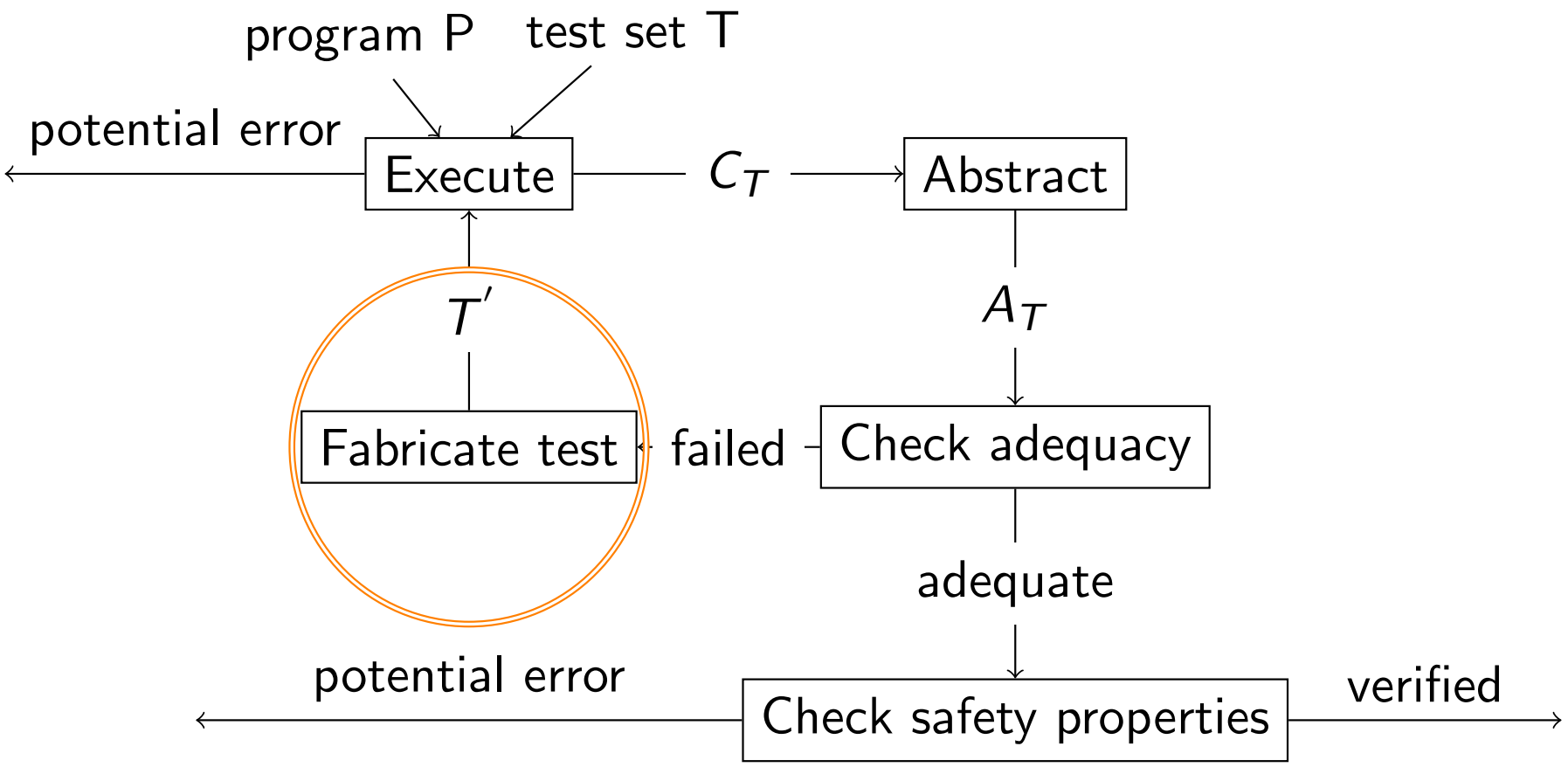
Algorithm: fixedFoo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 5 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```



(Figure 2(b), G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Fabricate test



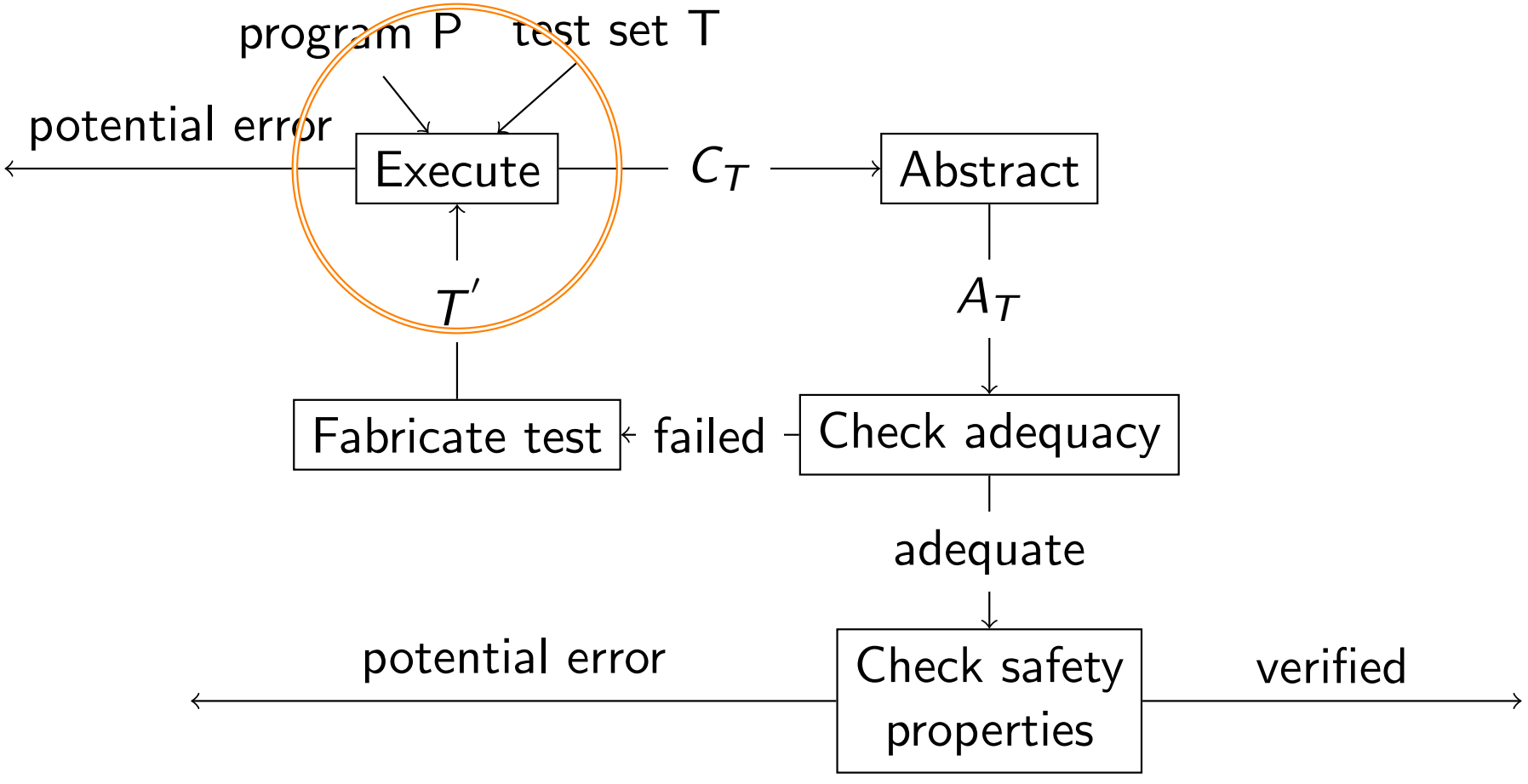
Fabricate test

Algorithm: fixedFoo(int x, int y)

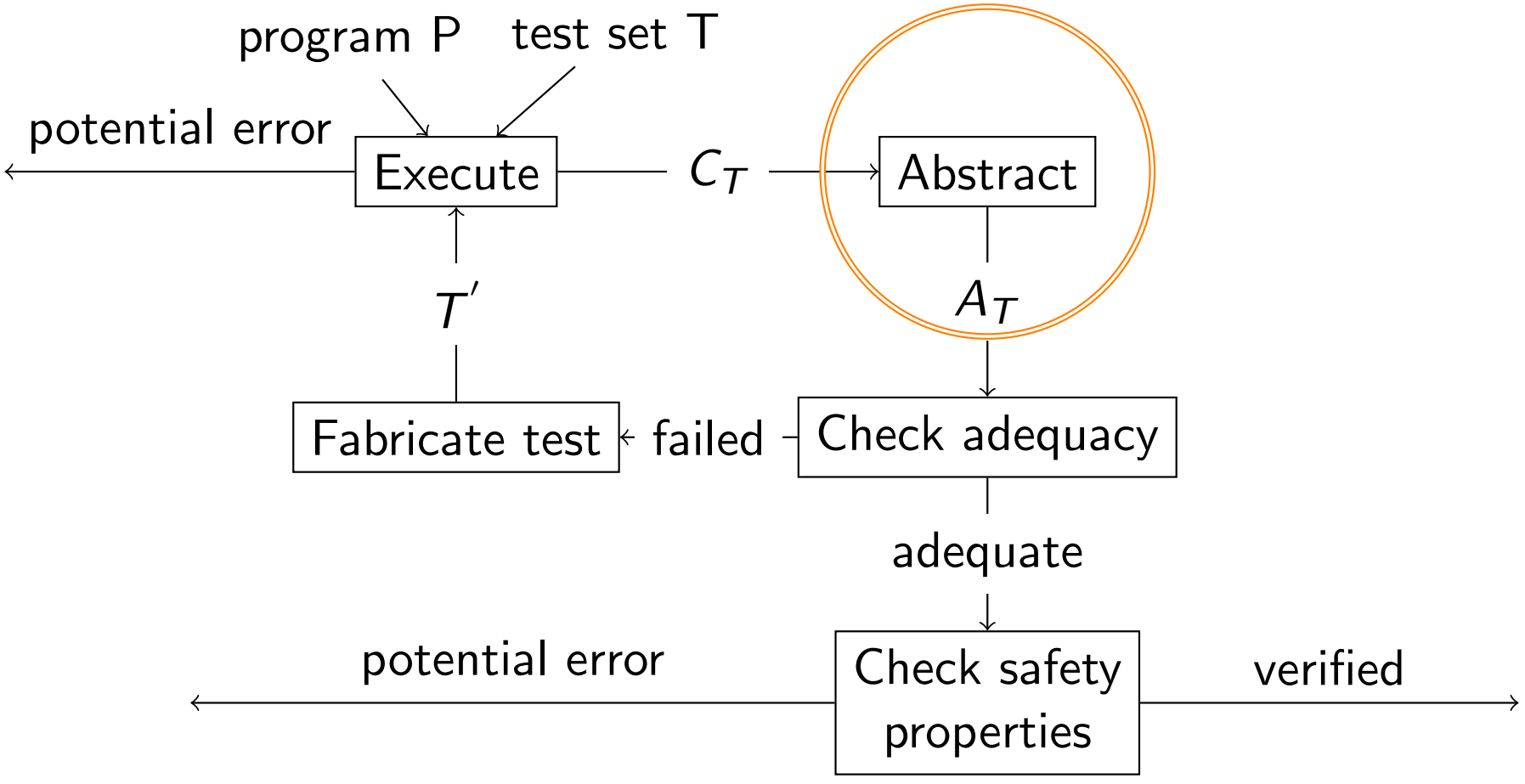
```
I  int * px = NULL;
A  x = x + 1;
B  if x < 5 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```

Fabricate tests
(pc=D, x=4, y=0, px=&y)
(pc=E, x=4, y=0, px=&y)
Remark: Both tests are not reachable, since px never assigned to &y.

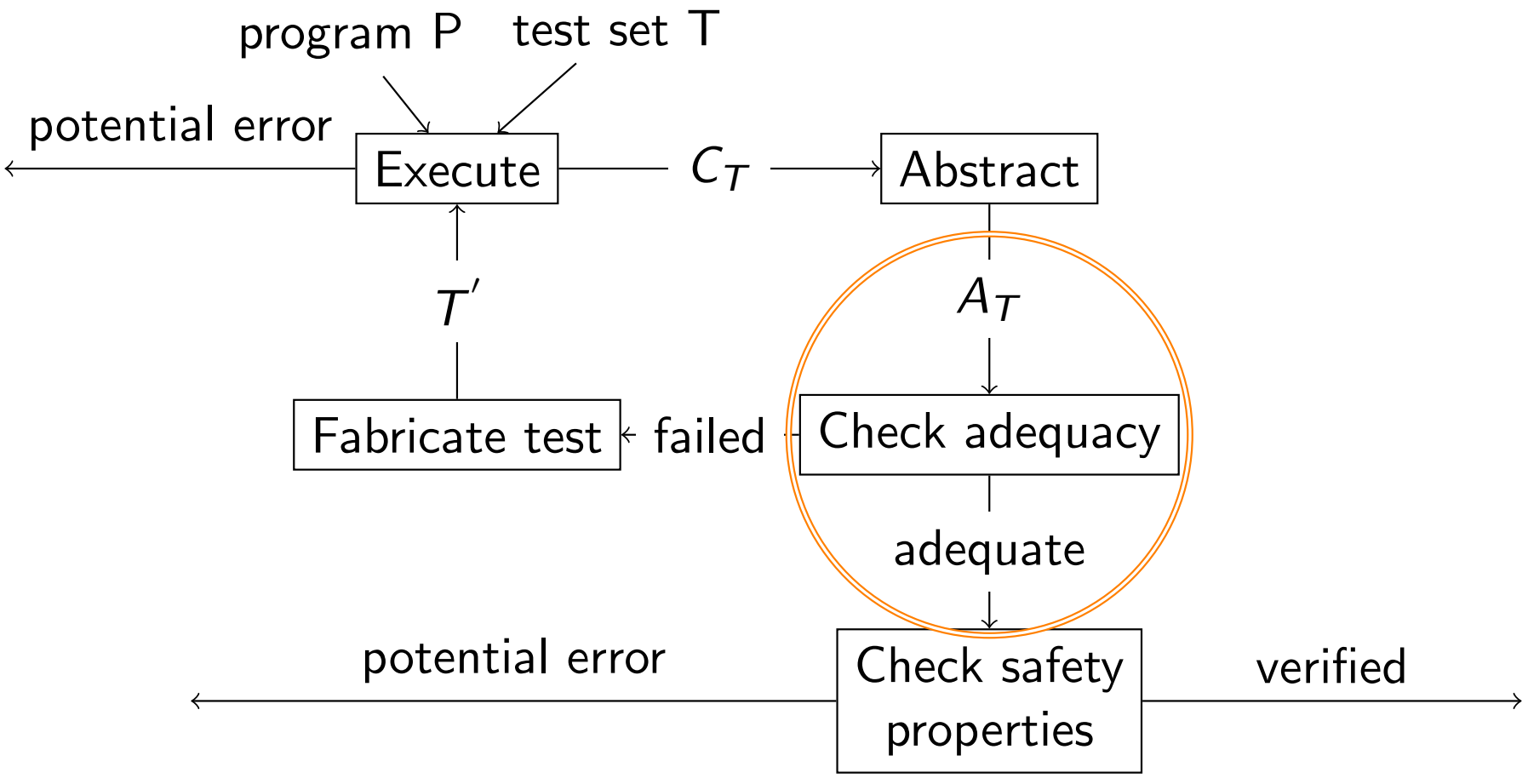
Run until check adequacy

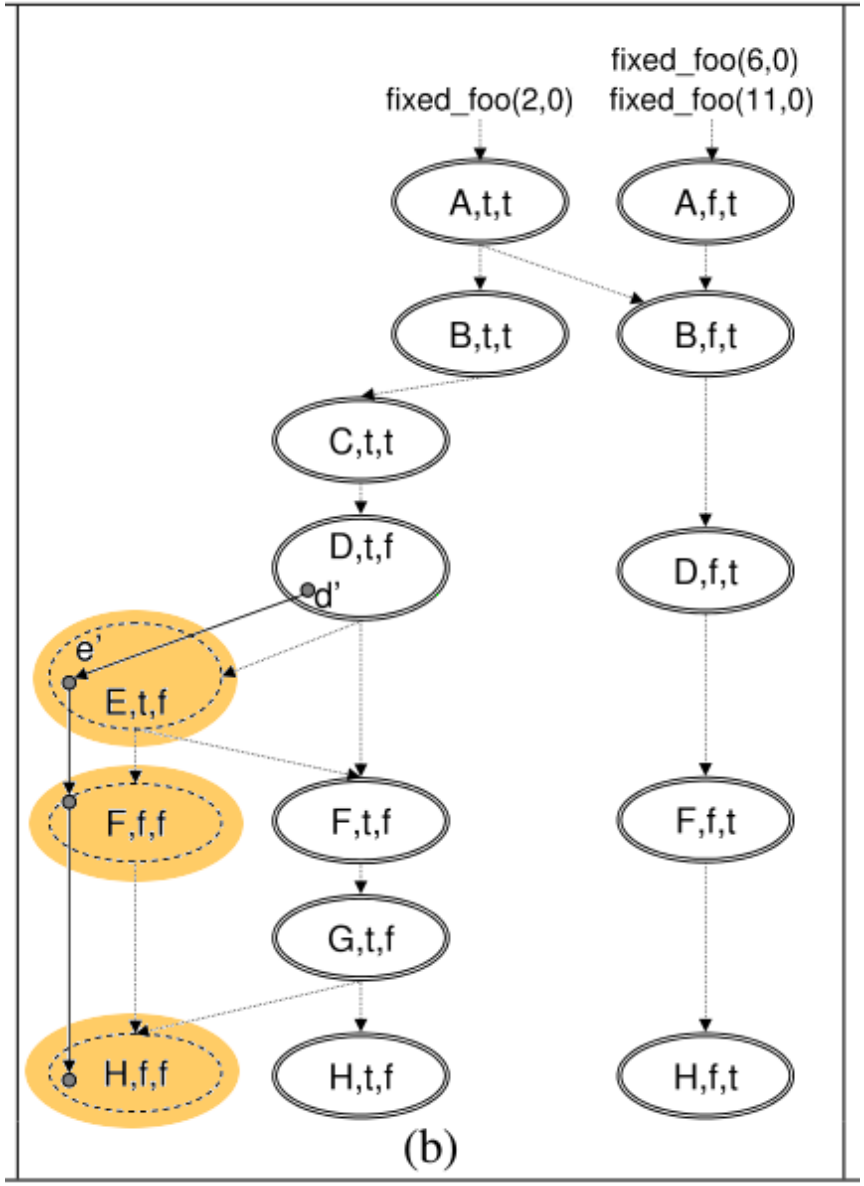


Run until check adequacy



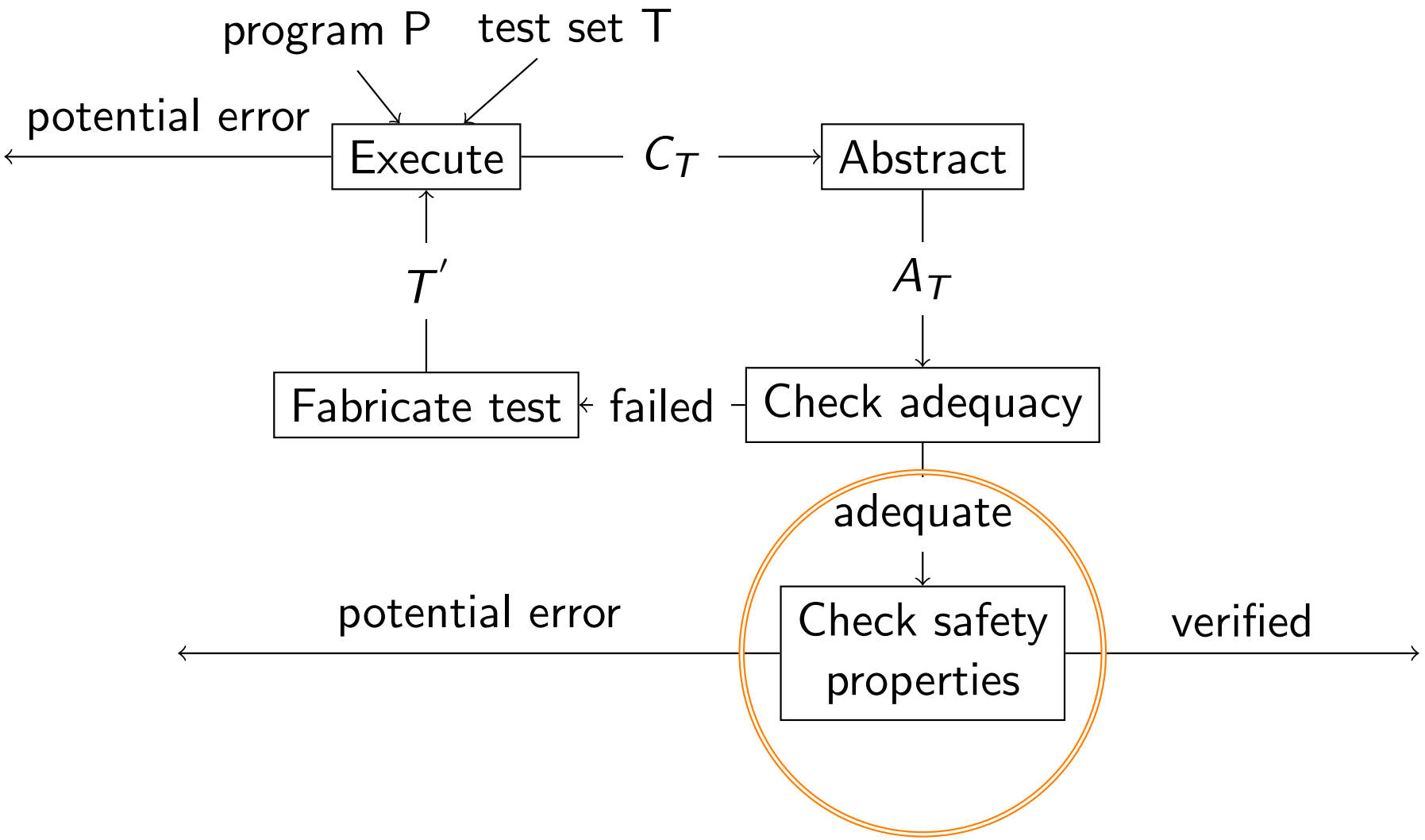
Run until check adequacy



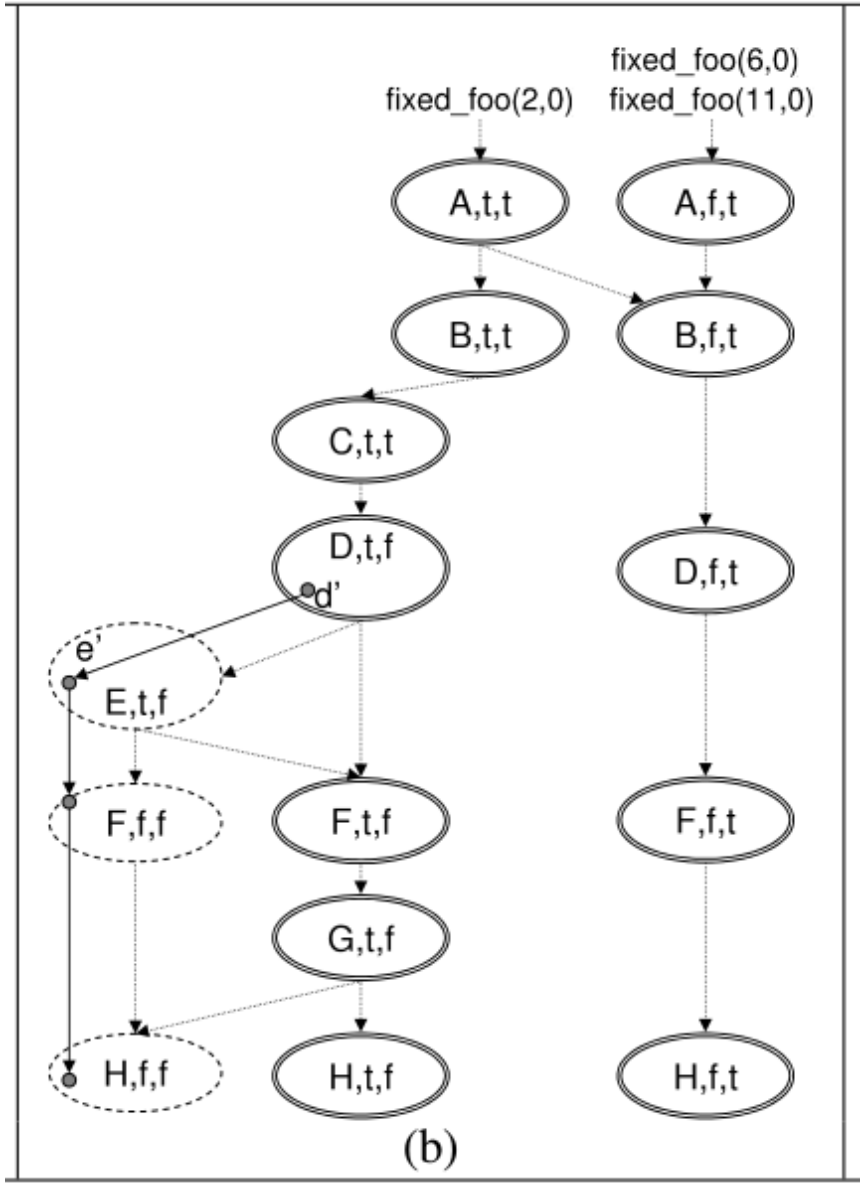


(Figure 2(b), G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Check safety properties



Error state $(G,t,t) \notin$



(Figure 2(b), G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Proof

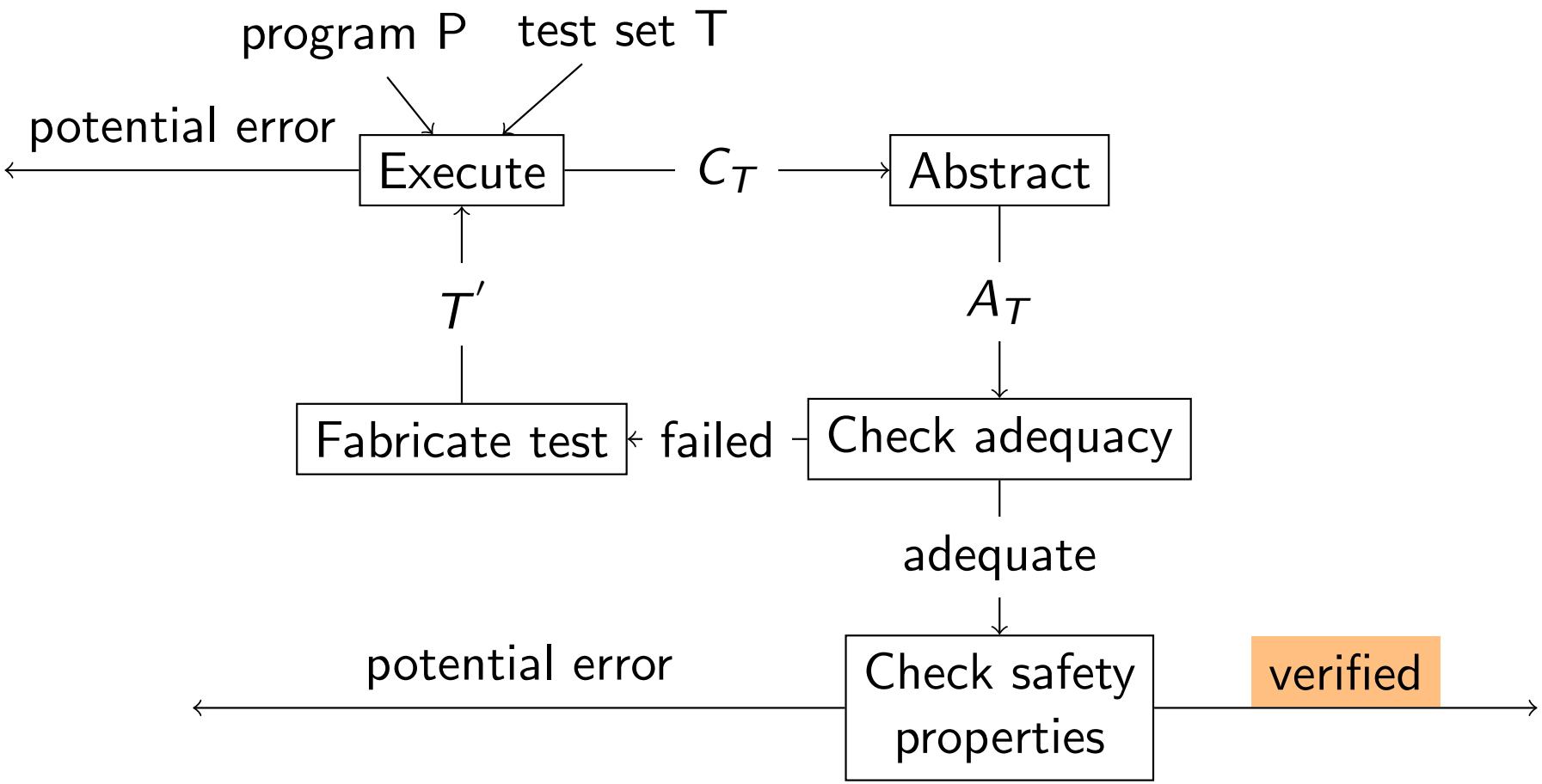

program P
test set T
potential error
Execute
C_T
Abstract
T'
A_T
Fabricate test
failed
Check adequacy
adequate
potential error
Check safety properties
verified

Table of Contents

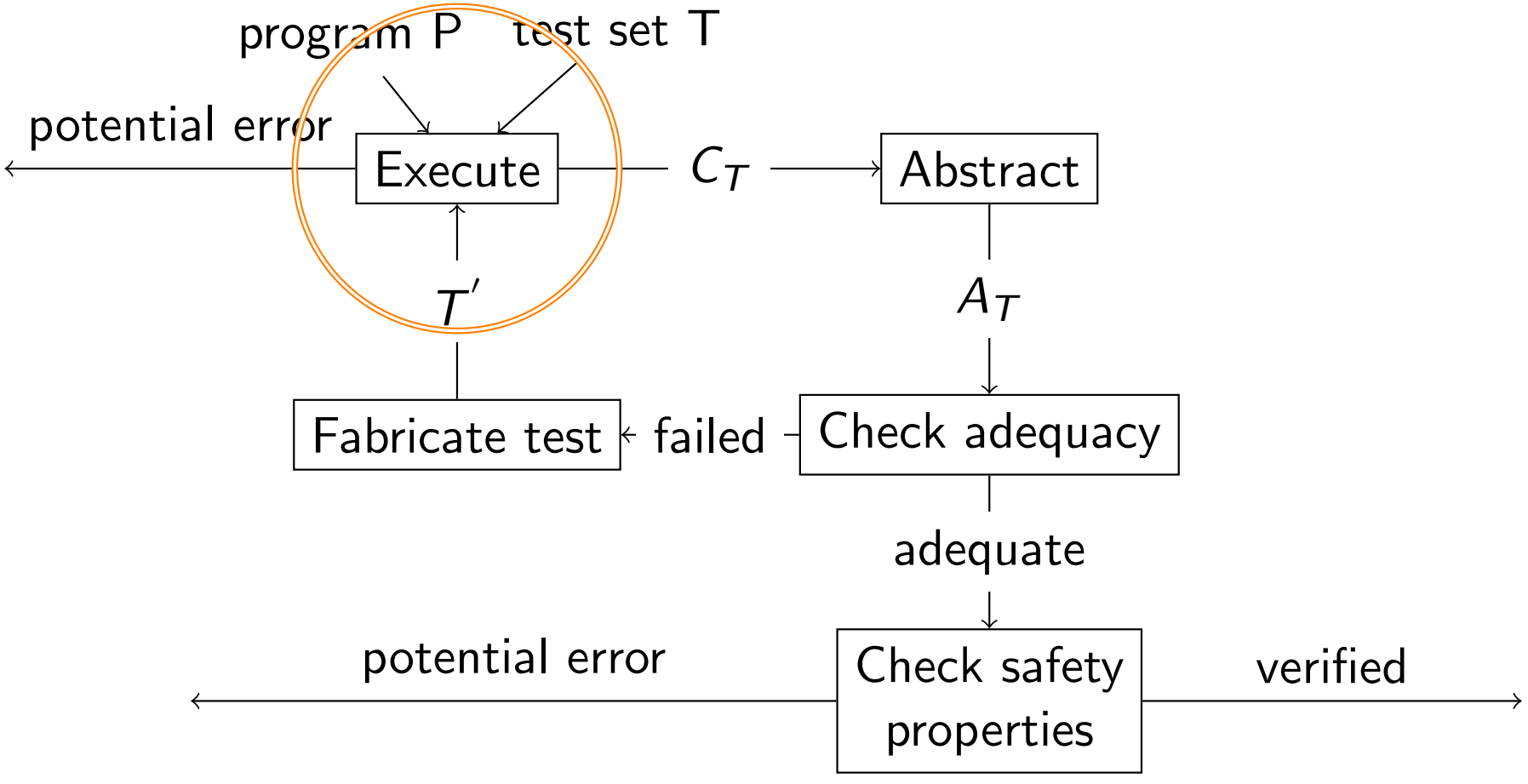
Input

- Program P
  fixedFoo
- Test set T

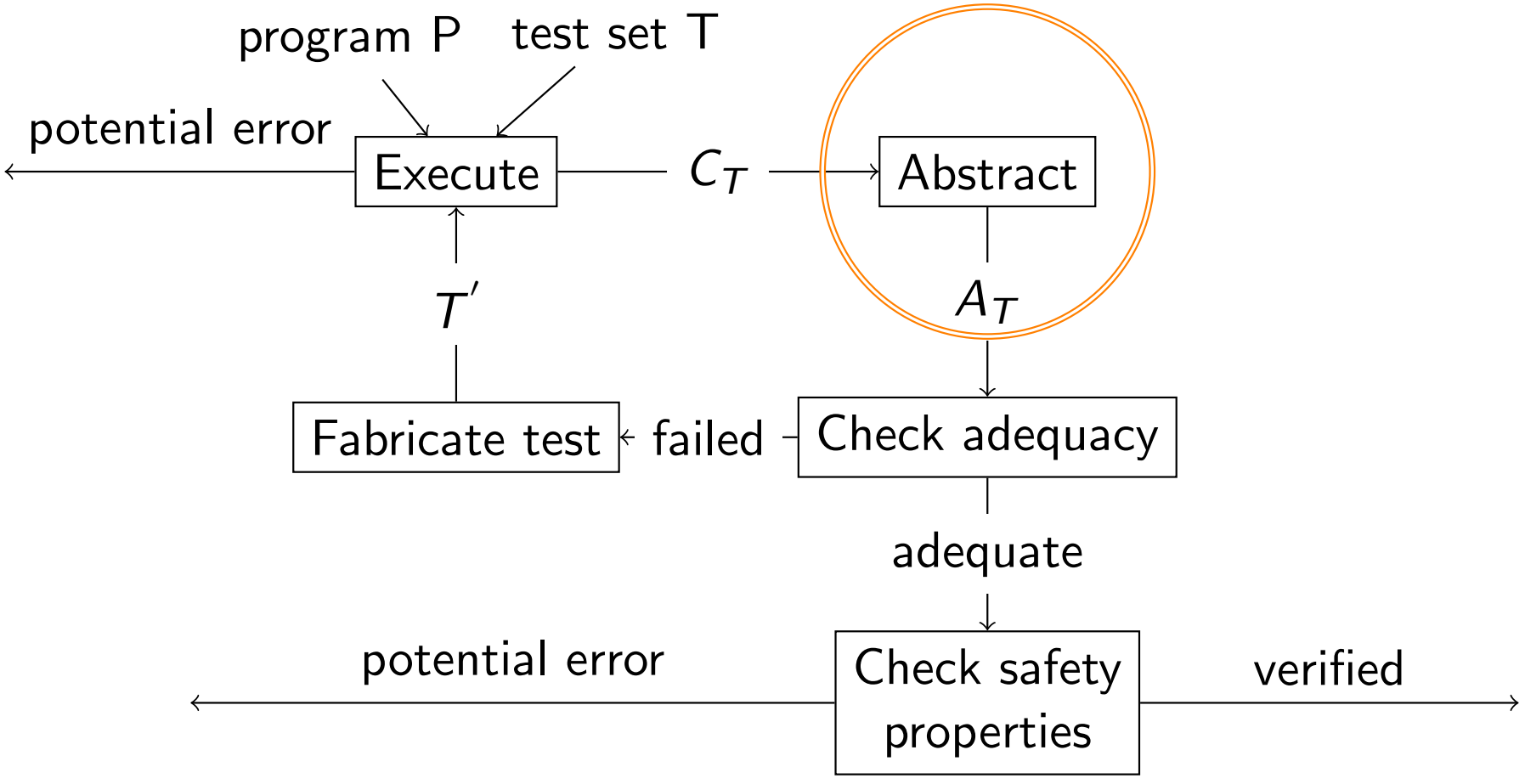
$$\{\underbrace{(pc = l, x = 2, y = 0, px = NULL)}_{fixedFoo(2,0)},$$
$$\underbrace{(pc = l, x = 6, y = 0, px = NULL)}_{fixedFoo(6,0)},$$
$$\underbrace{(pc = l, x = 11, y = 0, px = NULL)}_{fixedFoo(11,0)}\}$$

- abstraction function:
  $\alpha = \{(pc, x < 10, px = NULL) | (pc, x, y, px) \in C\}$

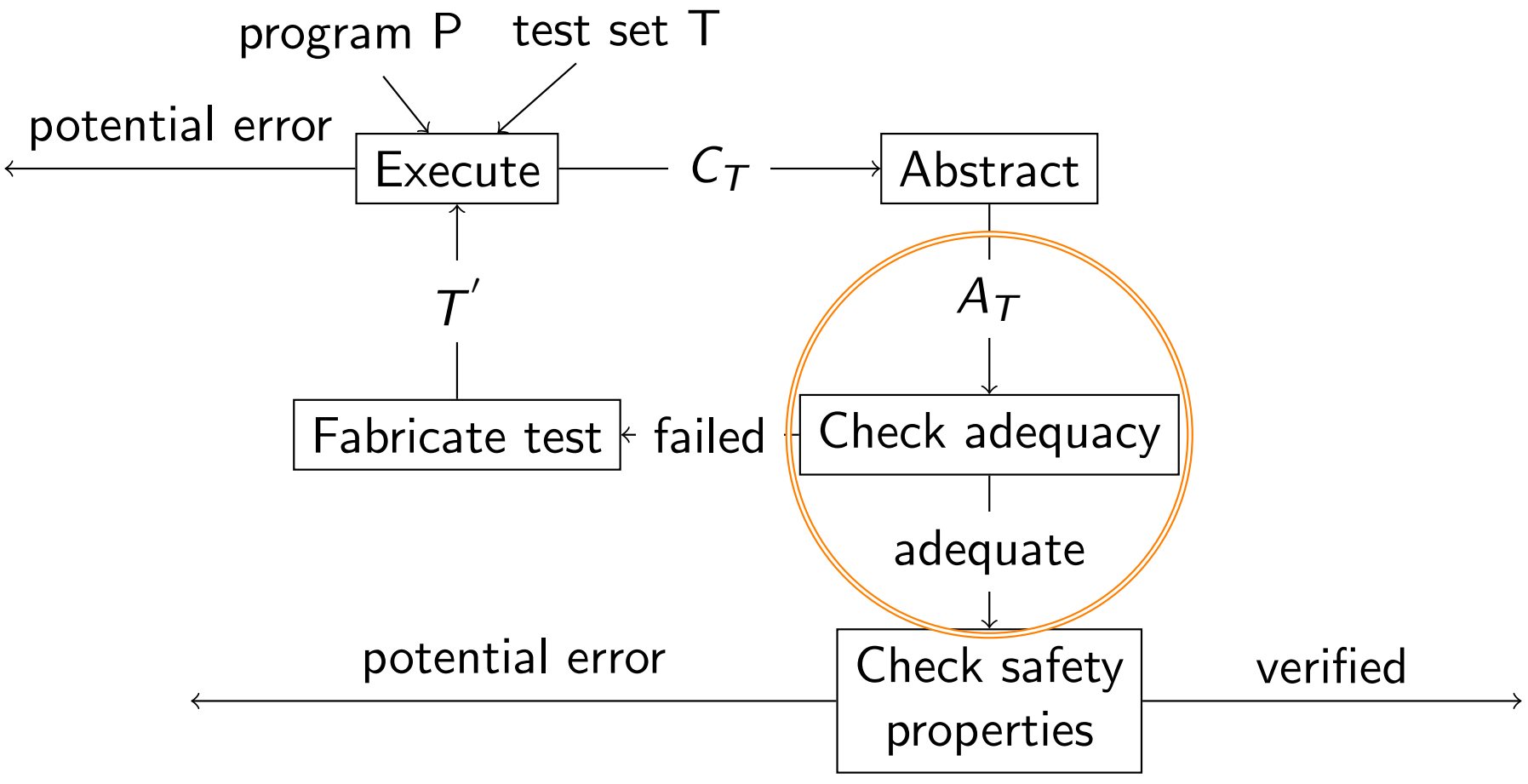
Run until check adequacy



Run until check adequacy

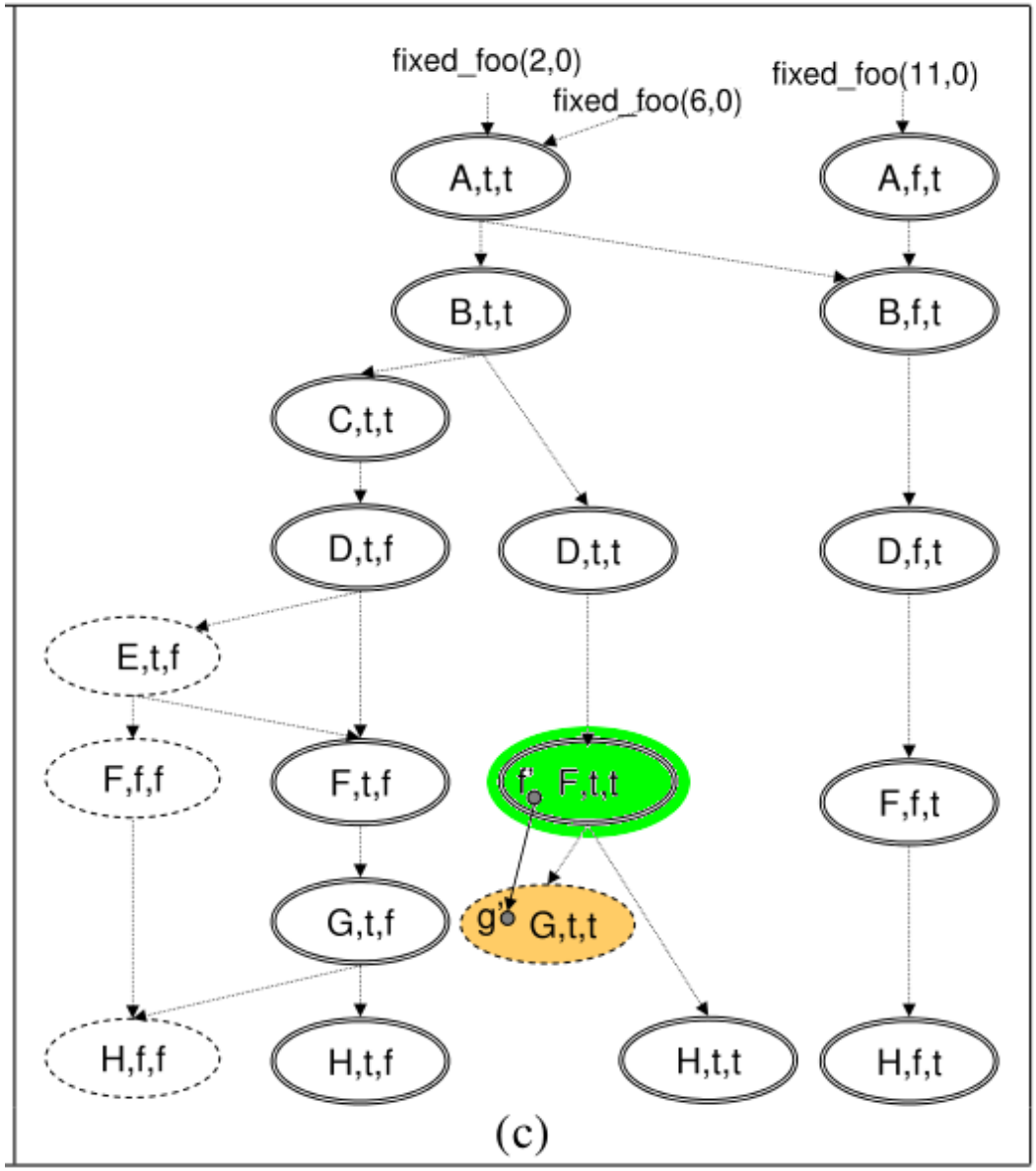


Run until check adequacy



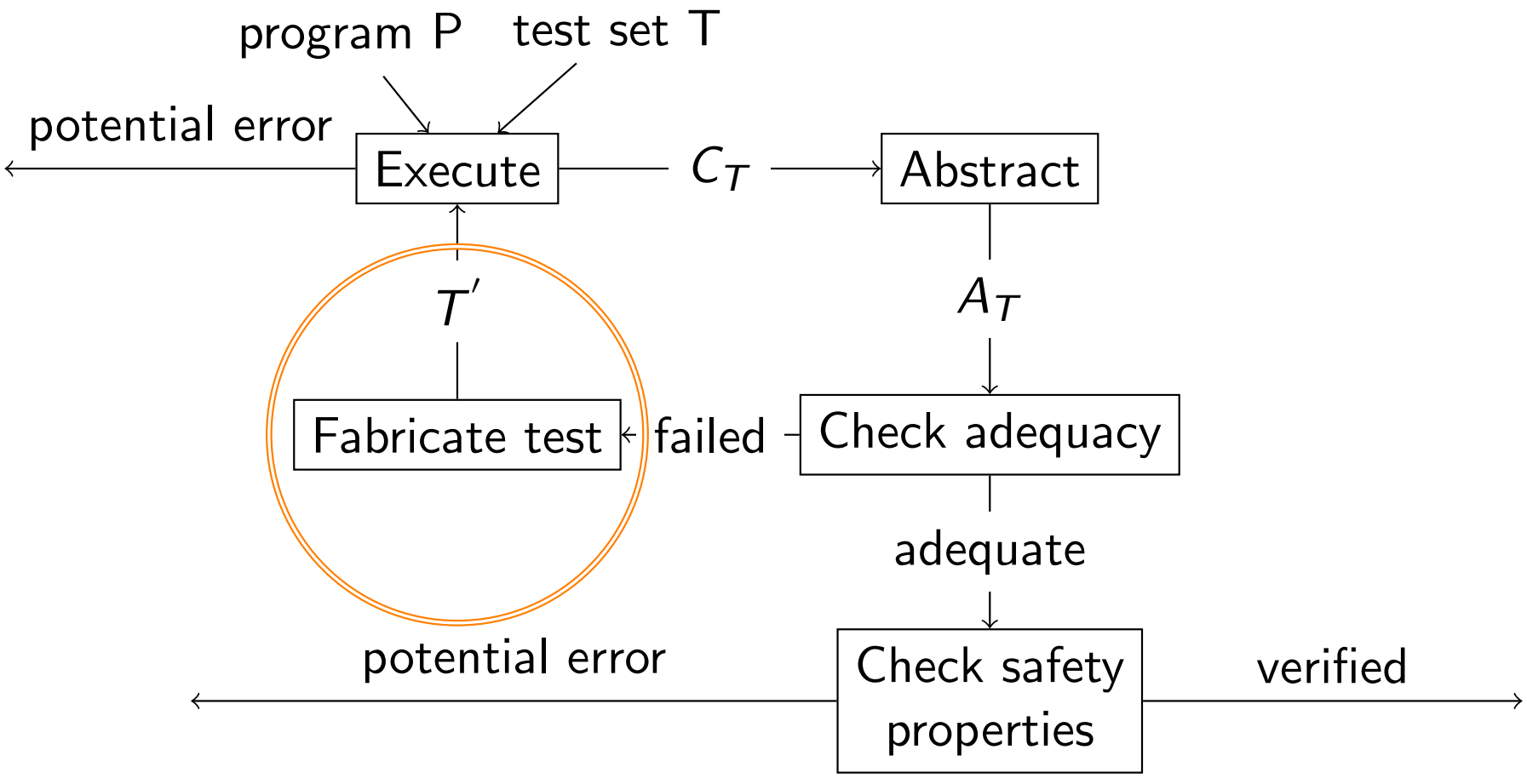
Algorithm: fixedFoo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 5 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```



(Figure 2(c), G. Yorsh , T. Ball, and M. Sagiv , 2006.)

Fabricate test



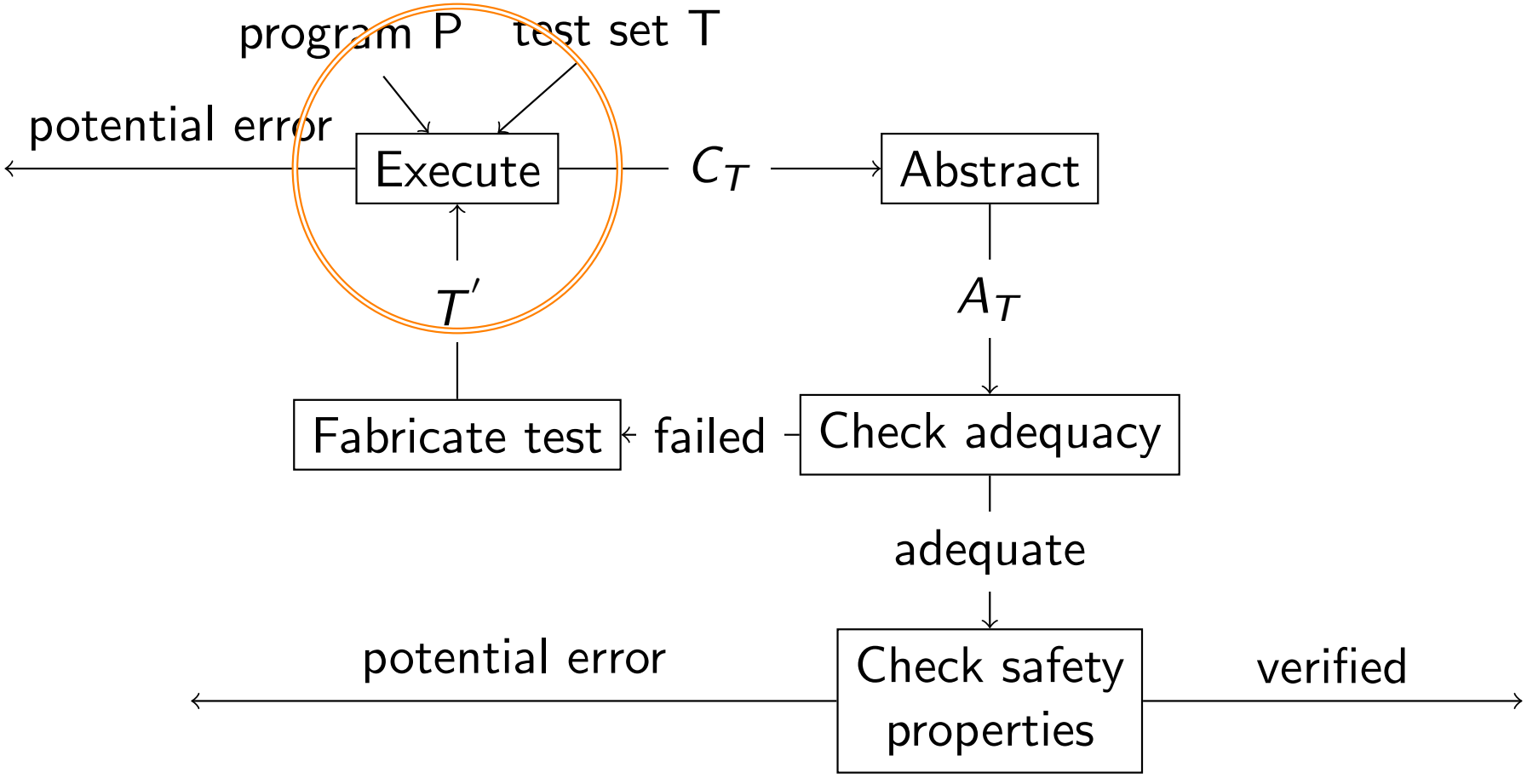
Fabricate test

Algorithm: fixedFoo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 5 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5 then
G  |   *px = *px + 1;
```

Fabricate tests
(pc=F, x=4, y=0, px=NULL)
(pc=G, x=4, y=0, px=NULL)
Remark: Both tests are not reachable,
since px = &x $\Leftrightarrow$ $x < 5$.

Execute


program P
test set T
potential error
Execute
C_T
Abstract
T'
A_T
Fabricate test
failed
Check adequacy
adequate
potential error
Check safety properties
verified

Execute (pc=F, x=4, y=0, px=NULL)

Algorithm: fixedFoo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 5 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5                          // (pc=F, x=4, y=0, px=NULL)
   then
G  |   *px = *px + 1;                // (pc=G, x=4, y=0, px=NULL)
```

Execute (pc=F, x=4, y=0, px=NULL)

Algorithm: fixedFoo(int x, int y)

```
I  int * px = NULL;
A  x = x + 1;
B  if x < 5 then
C  |   px = &x;
D  if px == &y then
E  |   x = x + 1;
F  if x < 5                          // (pc=F, x=4, y=0, px=NULL)
   then
G  |   *px = *px + 1;                // (pc=G, x=4, y=0, px=NULL)
```

$\Longrightarrow$ Null pointer dereference error

False error

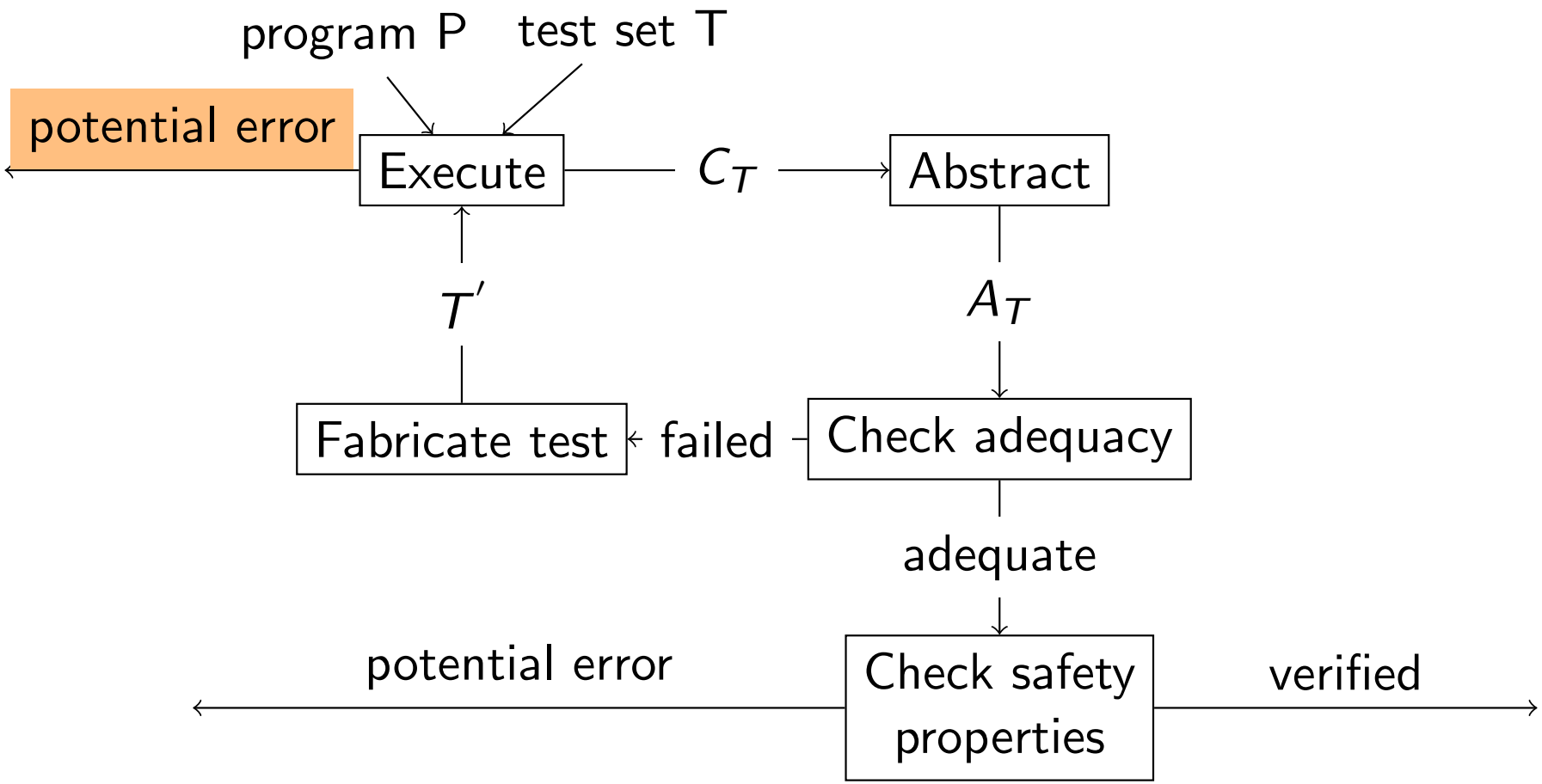

program P
test set T
potential error
Execute
C_T
Abstract
T'
A_T
Fabricate test
failed
Check adequacy
adequate
potential error
Check safety properties
verified

Table of Contents

Conclusion

- Given an abstraction function and safety property the procedure can proof the absence of errors in programs.
- Approach cannot distinguish between false and real errors like a dynamic approach.
- Idea of presented approach: Combining the pros of dynamic and static program analysis.

Conclusion

- Given an abstraction function and safety property the procedure can proof the absence of errors in programs.
- Approach cannot distinguish between false and real errors like a dynamic approach.
- Idea of presented approach: Combining the pros of dynamic and static program analysis.

Is this true ?

Conclusion
question