Program Verification I.

Critical Architectures Laboratory

Tamás Tóth

totht@mit.bme.hu

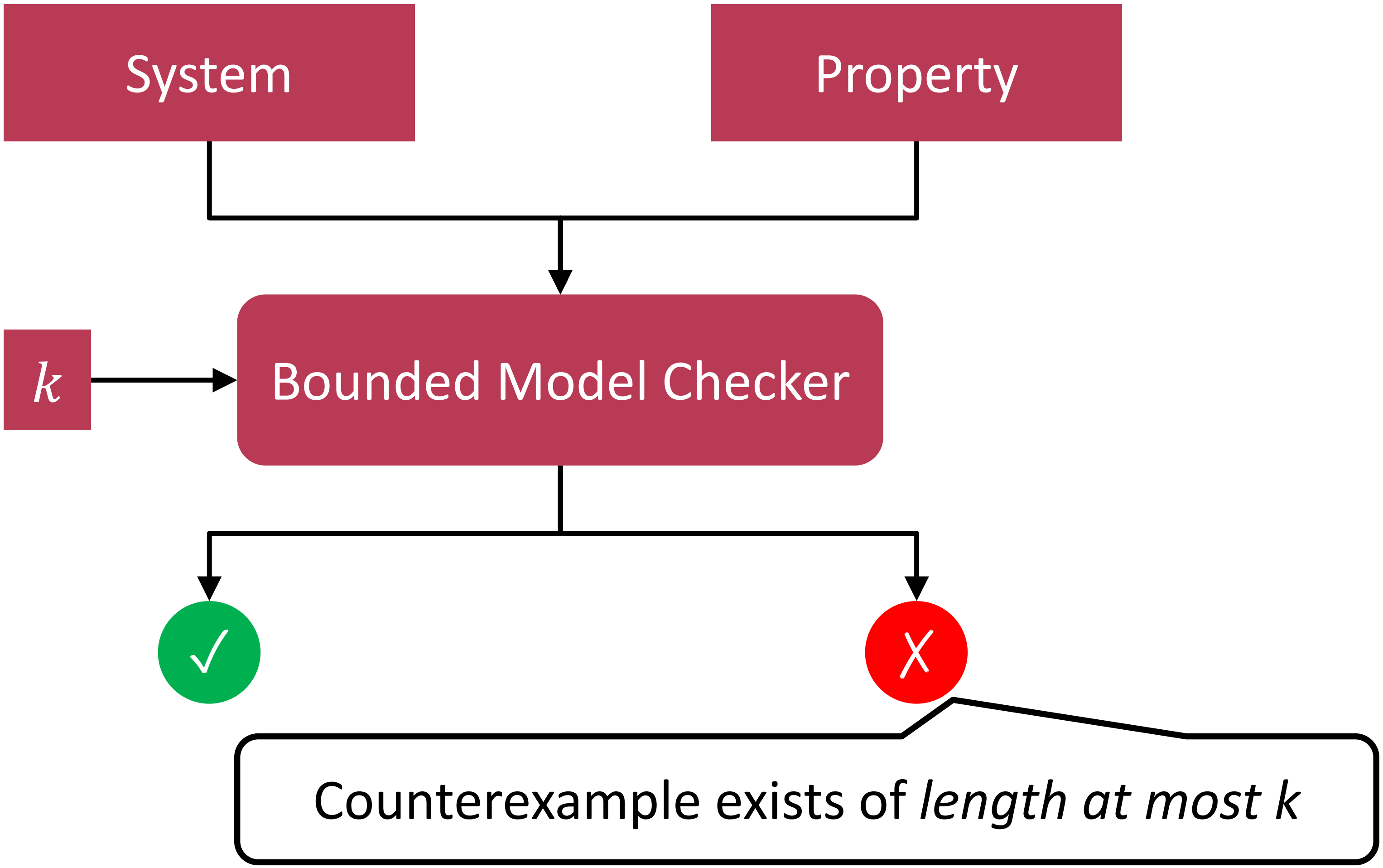
Budapest University of Technology and Economics
Fault Tolerant Systems Research Group

INTRODUCTION

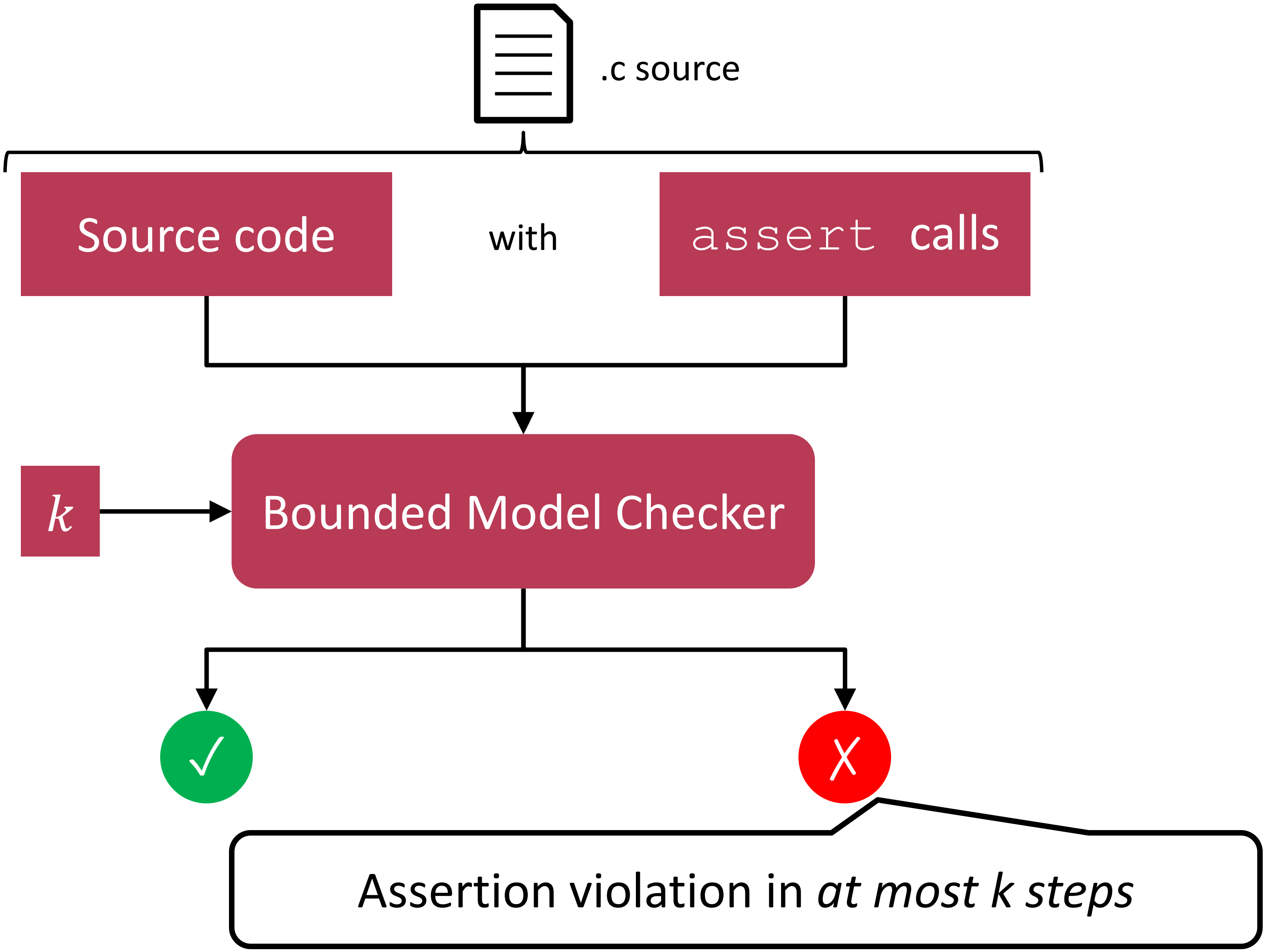
Topic of the Lab Session:

Implement a simple bounded model checker for a restricted fragment of the C programming language

Bounded Model Checking

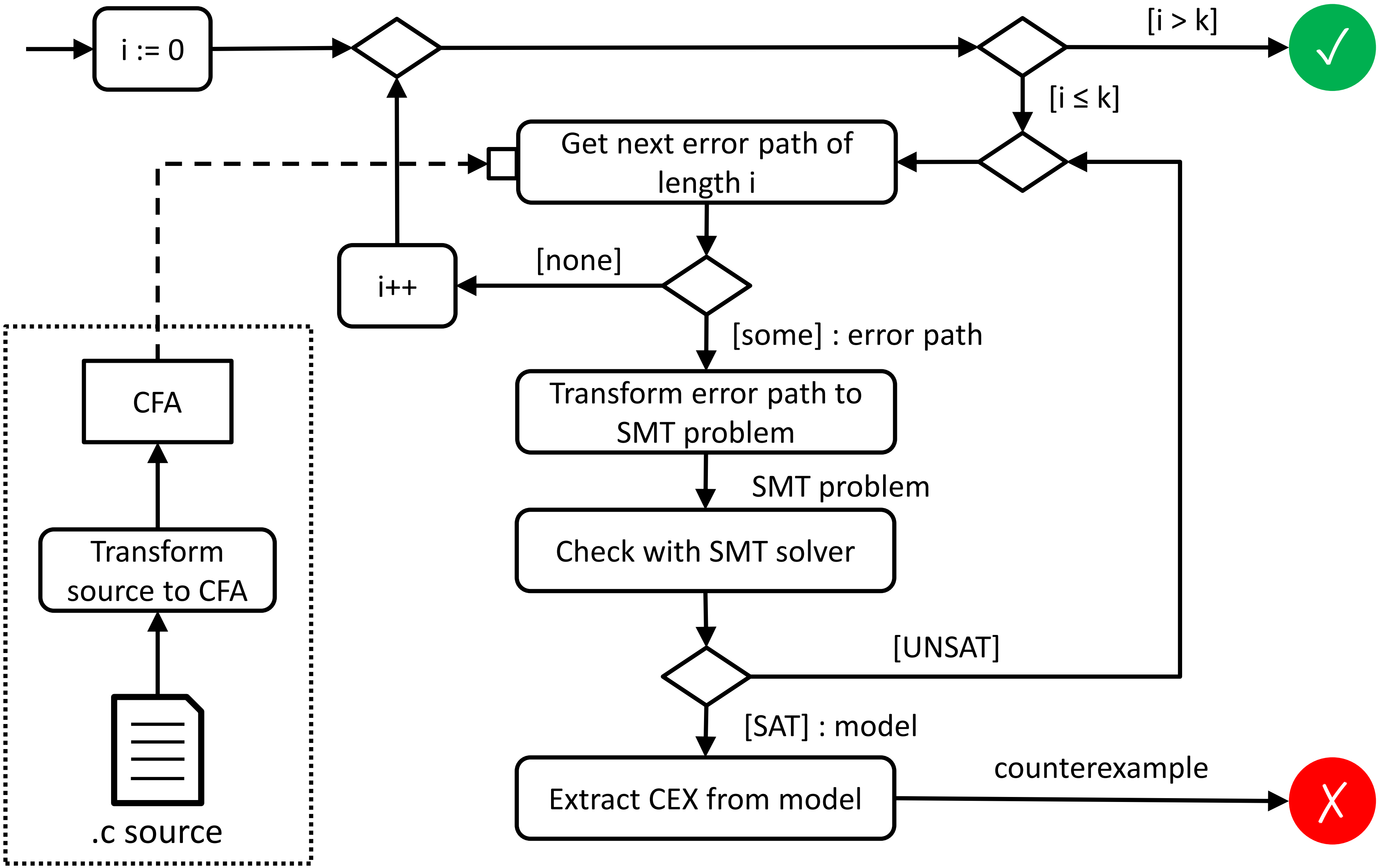


BMC for Programs



VERIFICATION WORKFLOW

BMC Workflow

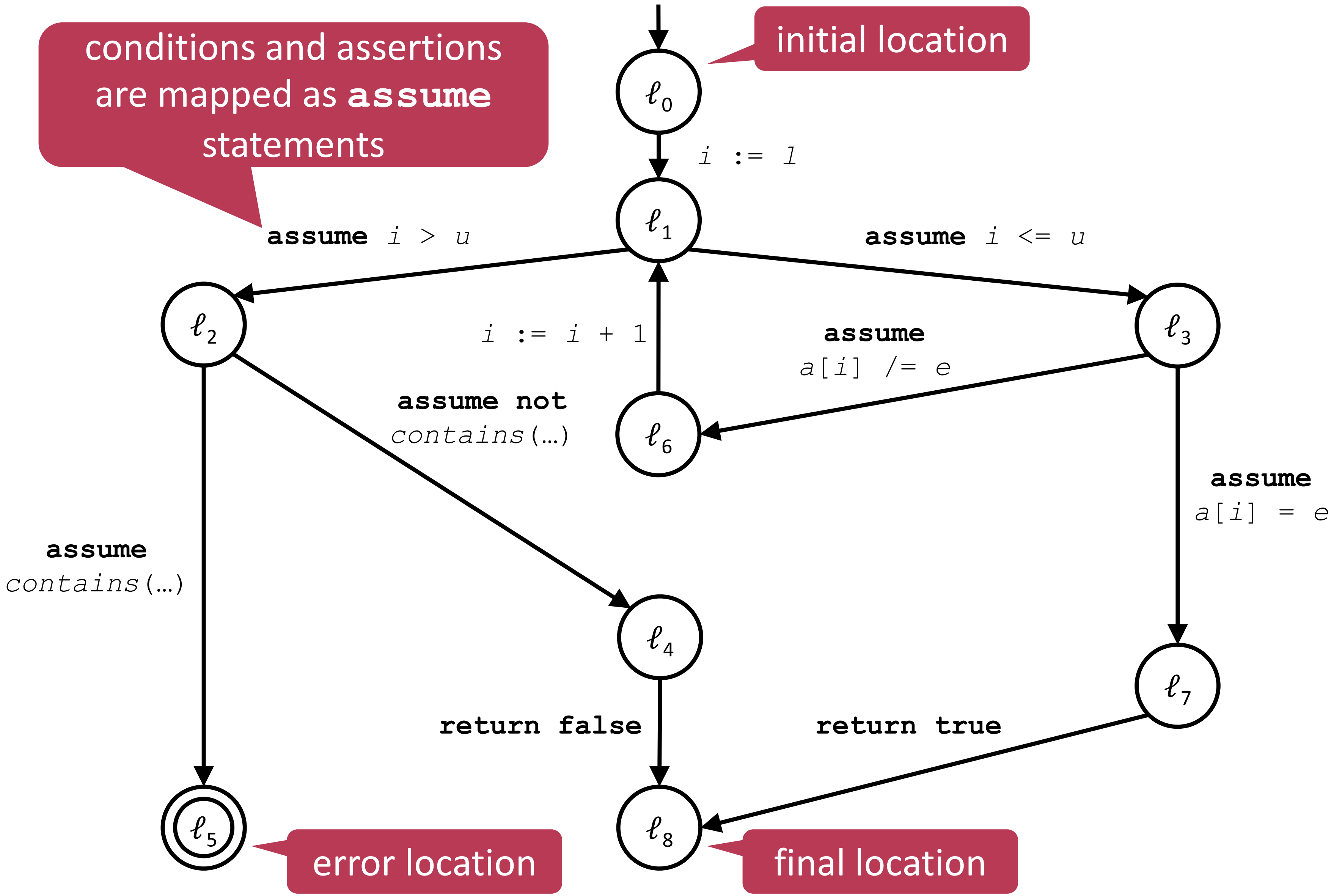


Source code with Assertions

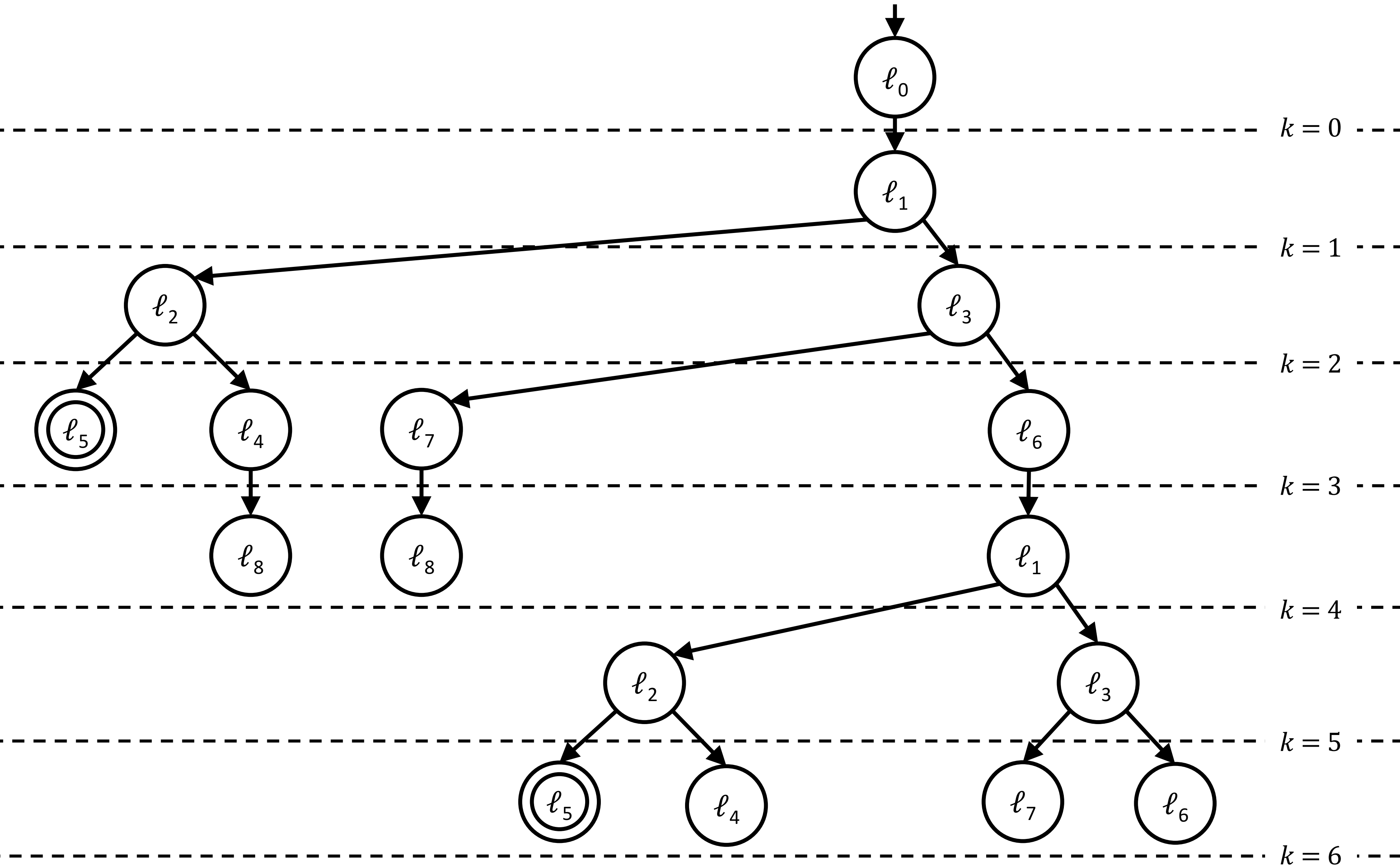
```
bool linearSearch(int[] a, int l, int u, int e) {

  for (int i = l; i <= u; i++) {
    if (a[i] == e) {
      return true;
    }
  }

  assert(!contains(a, l, u, e));

  return false;
}
```

`assert()` calls mark a requirement at the given point of control flow

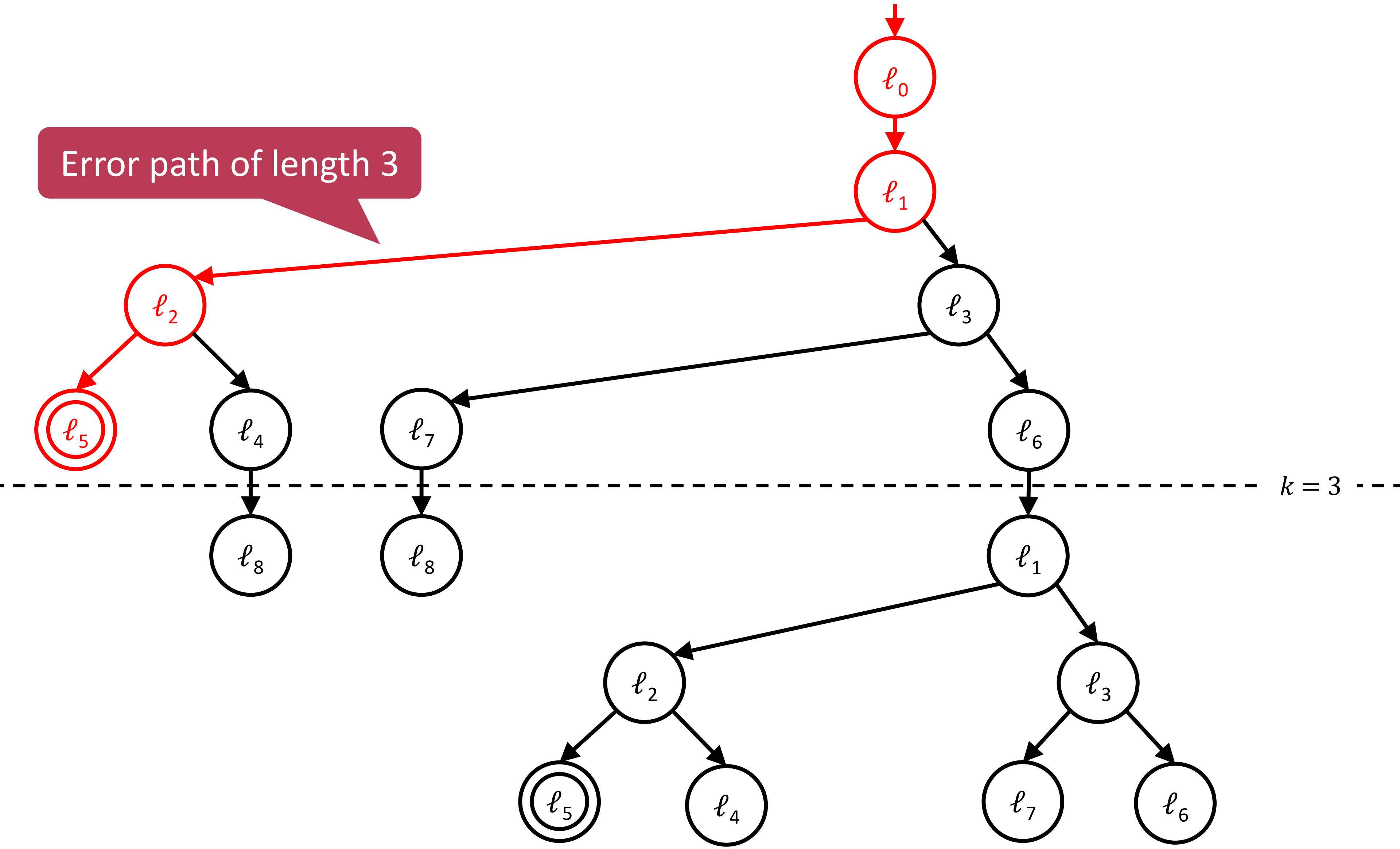
Control Flow Automata (CFA)



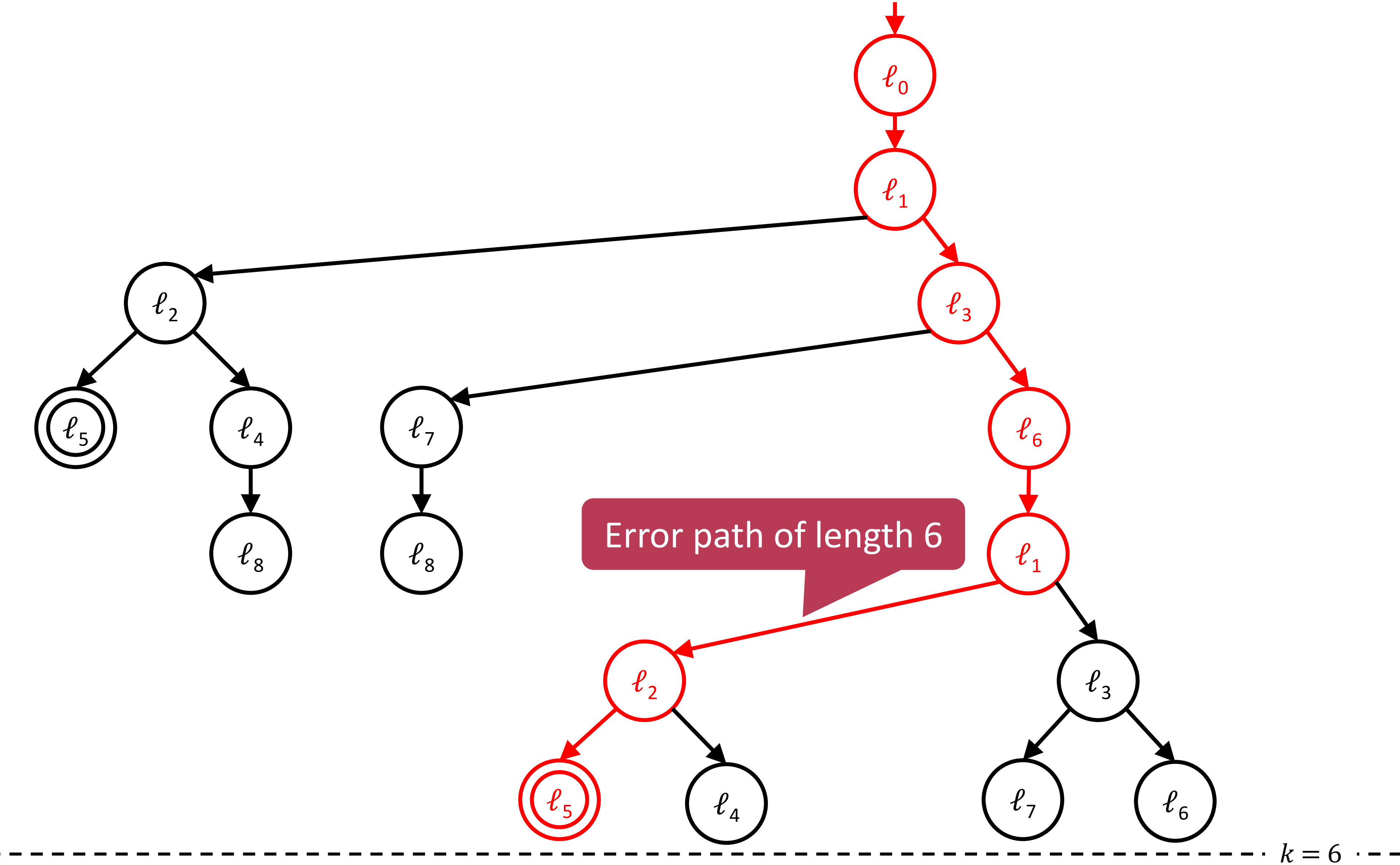
(Bounded) Unwinding of a CFA



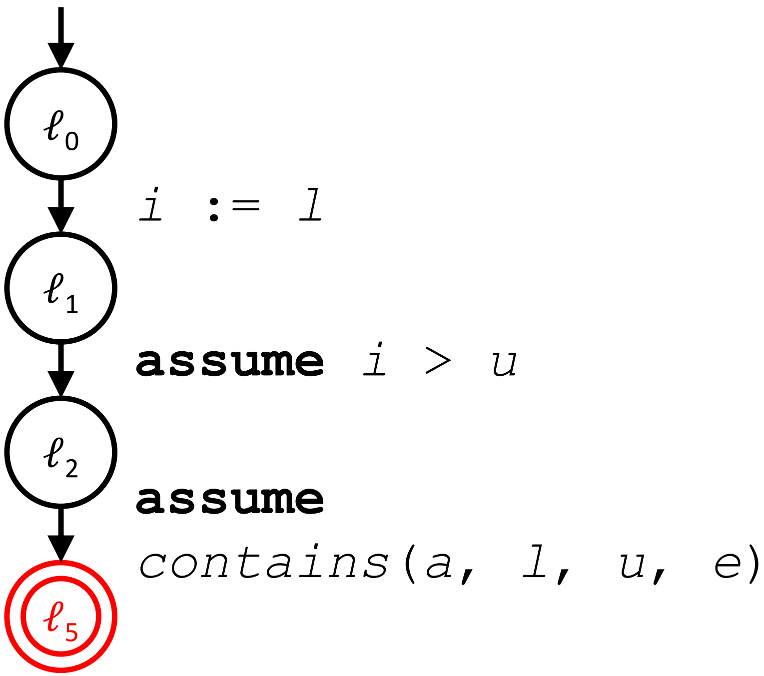
(Bounded) Unwinding of a CFA



(Bounded) Unwinding of a CFA



Error Paths



Error path

```
i := l;
assume i > u;
assume
   contains(a, l, u, e)
```

Simple program representing the error path: contains only assignments and assumptions

Checking error paths

Program path

```
i := l;
assume i > u;
assume exists (j : integer) :
   (j >= l and j < u and a[j] = e)
```

can be taken for some inputs *a*, *l*, *u*, *e*

iff

SMT problem

$$i_0 = l$$
$$i_0 > u$$
$$\exists(j : Int) : (j \geq l \wedge j < u \wedge a[j] = e)$$

is satisfiable.

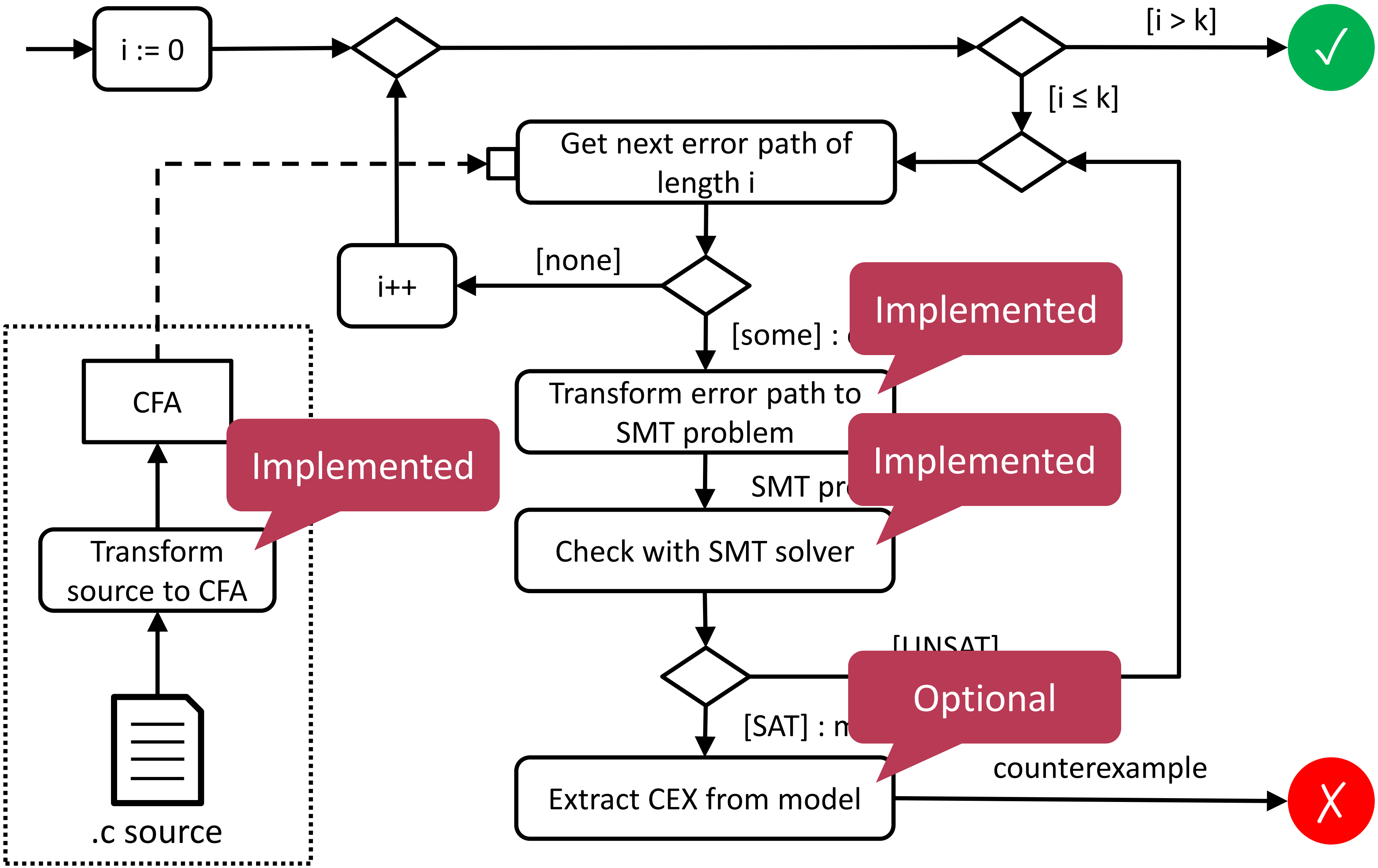
Transforming Statements to SMT

```
x := a
y := b
tmp := a
a := b
b := tmp
assume y >= a
assume x >= b
```

$$
\begin{aligned}
&x_0 = a_0\\
&y_0 = b_0\\
&tmp_0 = a_0\\
&a_1 = b_0\\
&b_1 = tmp_0\\
&y_0 \geq a_1\\
&x_0 \geq b_1
\end{aligned}
$$

- Introduce a fresh constant symbol for the variable in the left-hand side in each assignment
- Refer to the freshest constant symbol accordingly

BMC Workflow: Tasks



LIST OF QUESTIONS

List of questions

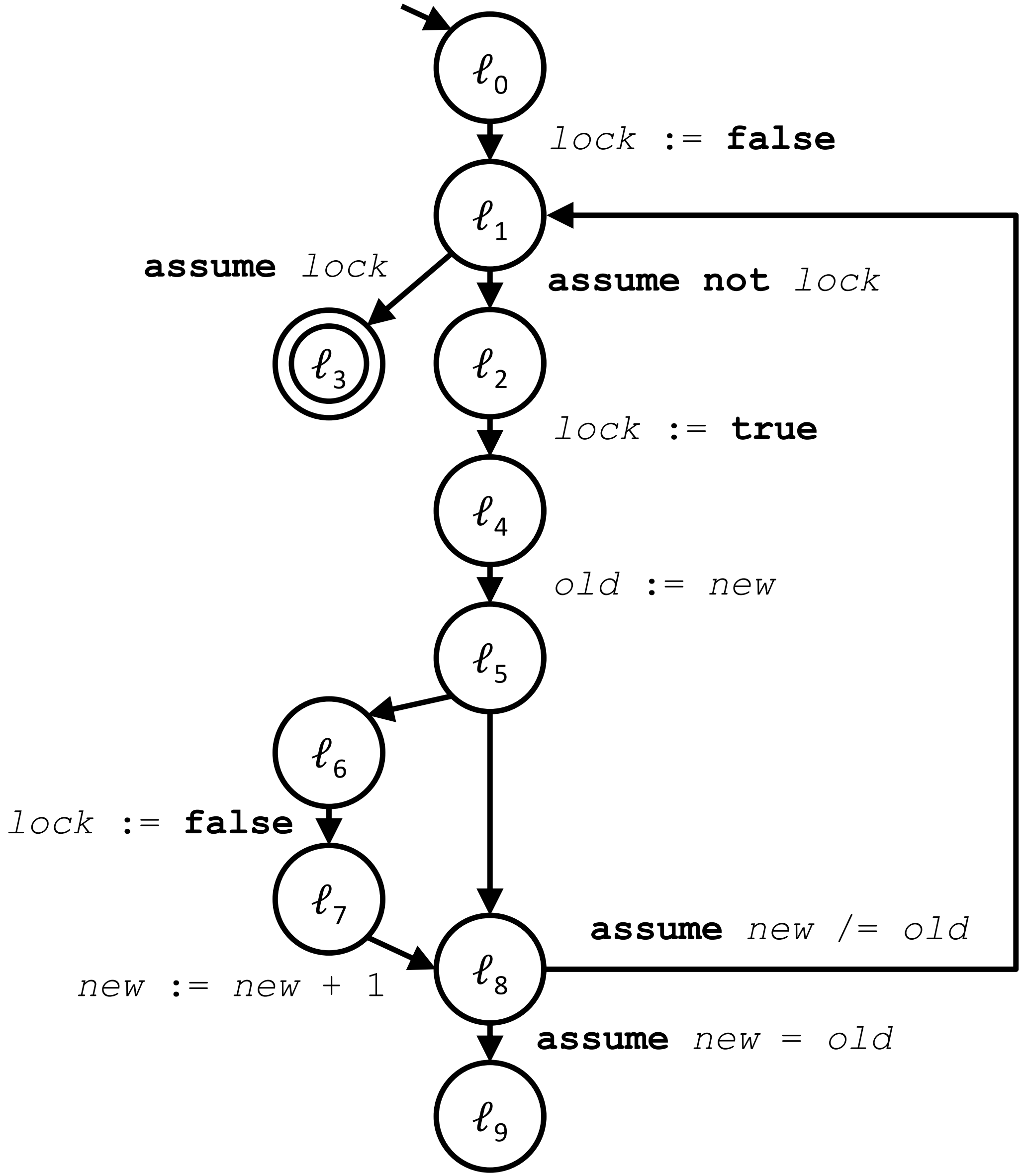
1. Transform the following program to CFA form:

```
int lock = 0;
int old, new;
do {
  assert(!lock);
  lock = true;
  old = new;
  if (nondet_bool()) {
    lock = false;
    new++;
  }
} while (new != old)
```

2. Determine the program paths that represent the three shortest error paths of the program
3. Transform the paths to SMT problems
4. Give an argument for their unsatisfiability

SOLUTIONS

Solution (1)



ℓ0
lock := false
ℓ1
assume lock
assume not lock
ℓ3
ℓ2
lock := true
ℓ4
old := new
ℓ5
ℓ6
lock := false
ℓ7
new := new + 1
ℓ8
assume new /= old
assume new = old
ℓ9

Solution (2)(3)(4)

```
lock := false;
assume lock;
```

$\neg lock_0$
$lock_0$

```
lock := false;
assume not lock;
lock := true;
old := new;
assume new /= old;
assume lock;
```

$\neg lock_0$
$\neg lock_0$
$lock_1$
$old_0 = new_0$
$new_0 \neq old_0$
$lock_1$

```
lock := false;
assume not lock;
lock := true;
old := new;
lock := false;
new := new + 1;
assume new /= old;
assume lock;
```

$\neg lock_0$
$\neg lock_0$
$lock_1$
$old_0 = new_0$
$\neg lock_2$
$new_1 = new_0 + 1$
$new_1 \neq old_0$
$lock_2$