Bounded Model Checking of Software Code

Dániel Szekeres, Csanád Telbisz

Based on: slides by Tamás Tóth

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Critical Systems Research Group

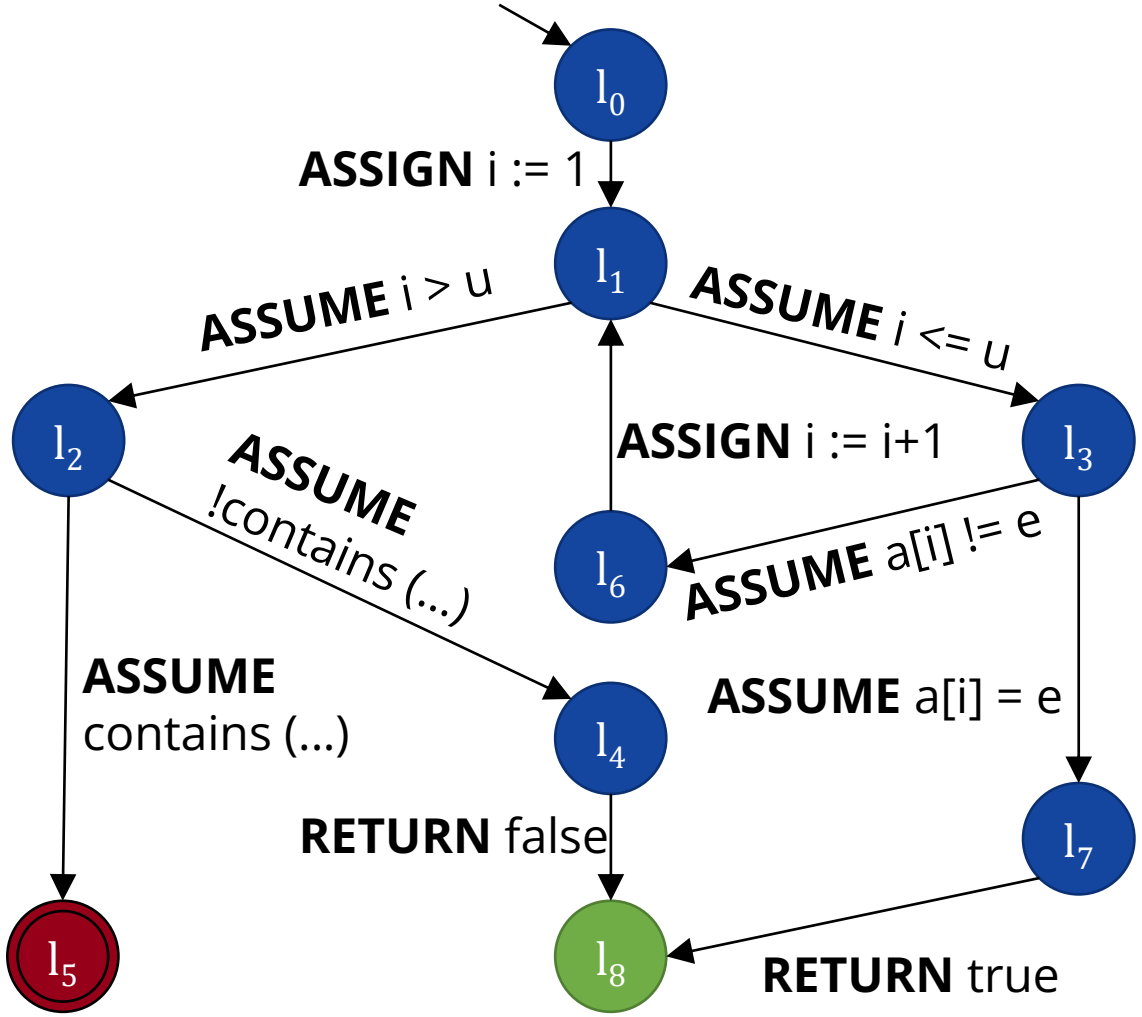
Source code with assertions

```
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;
}
```

contains: *e* can be found in *a* between indices *l* and *u*

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

Control Flow Automaton



- **Variables**
- **Locations**: state of execution
  - Final location: successful termination
  - Error location: erroneous termination
    → this is what we are searching for
- **Edge**: represents the application of a statement in the program
- **Statements** on the edges:
  - **ASSUME**: for conditions and assertions
  - **ASSIGN**: for changing variable values
  - **RETURN**: ASSIGN to a special return variable
    → not present in the implementation
- Multiple statements could be on a single edge (not used in the implementation)

BMC for CFA models

- Naive implementation (original BMC):
  single transition formula, additional location variable

  $$(loc == 1 \wedge stmt_1 \wedge loc' == 2) \vee (loc == 2 \wedge stmt_2 \wedge loc' == 3) \vee \cdots$$

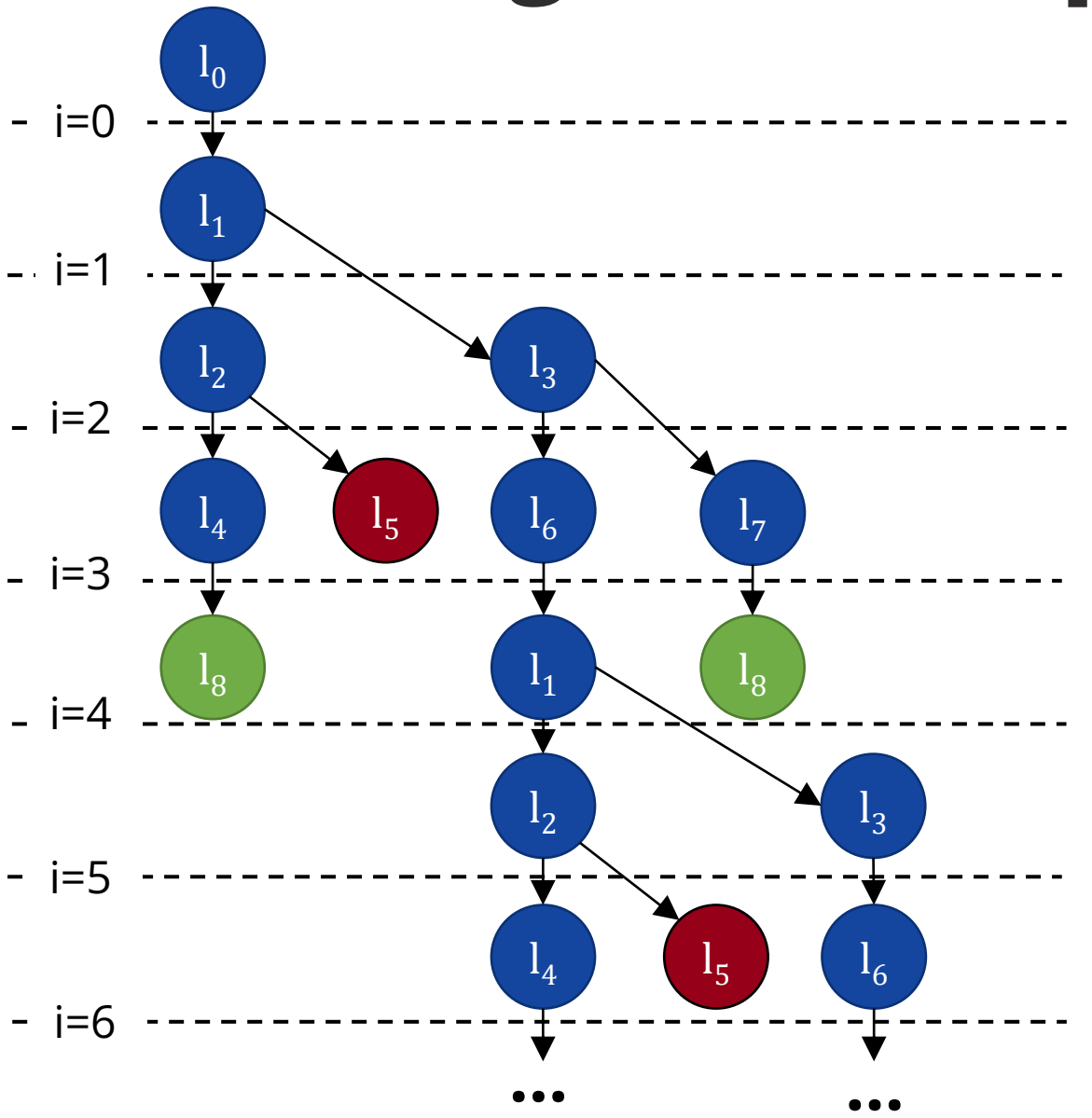
- This is inefficient: the control flow is encoded in a variable, the solver has to figure out which statement sequences even exist

BMC for CFA models

Better solution:

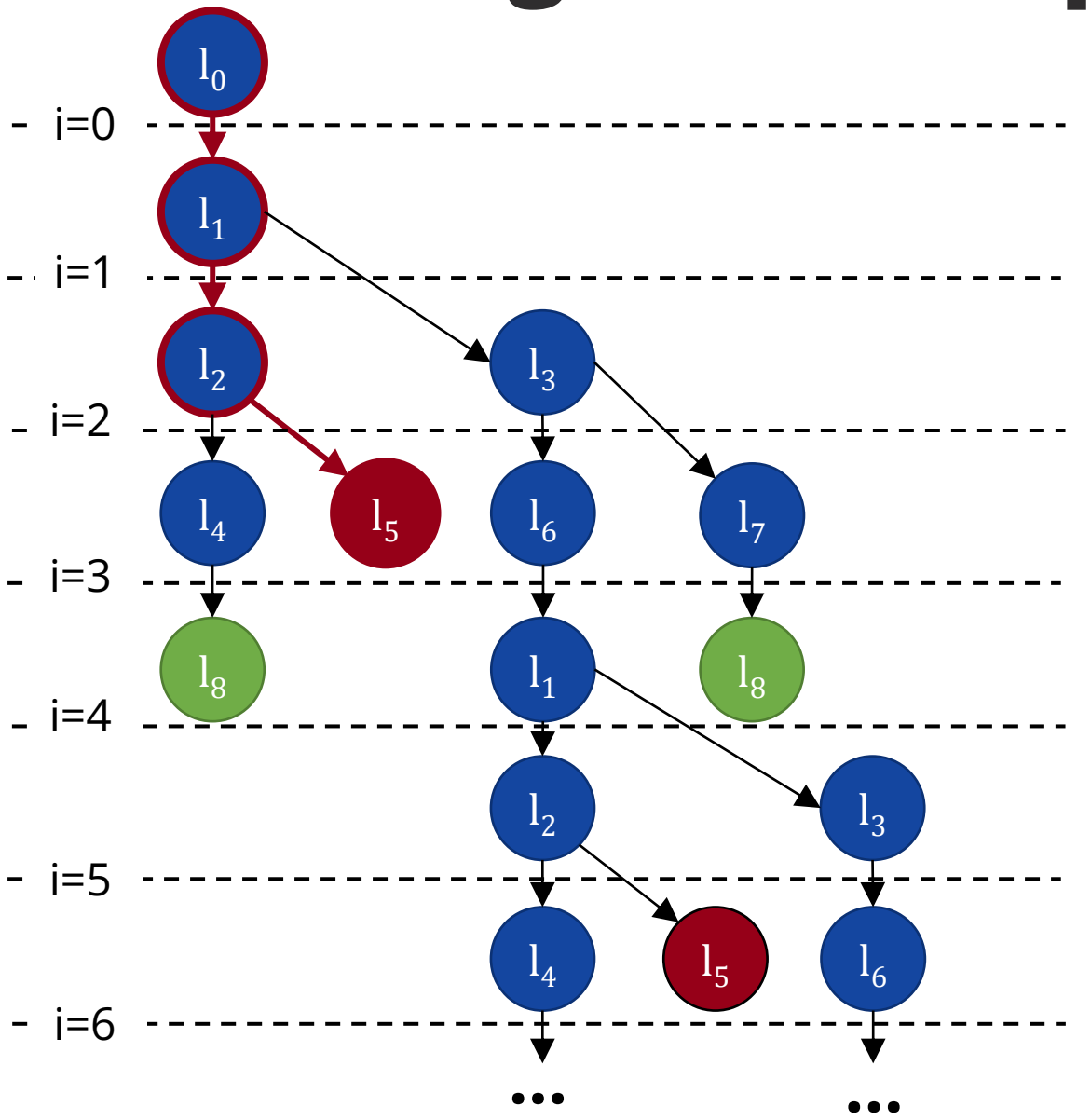
- First **search for error paths** of length *i* only taking the *control flow* into account (ignoring variable values → **no SMT solver needed**)
- If one is found: use the **SMT solver** to check whether **that path** is actually **feasible** (considering the variables, only one specific sequence → no location variable and control flow encoding needed)
- No feasible error path → $i := i + 1$
- Until i reaches a specified bound *k*

Searching for error paths



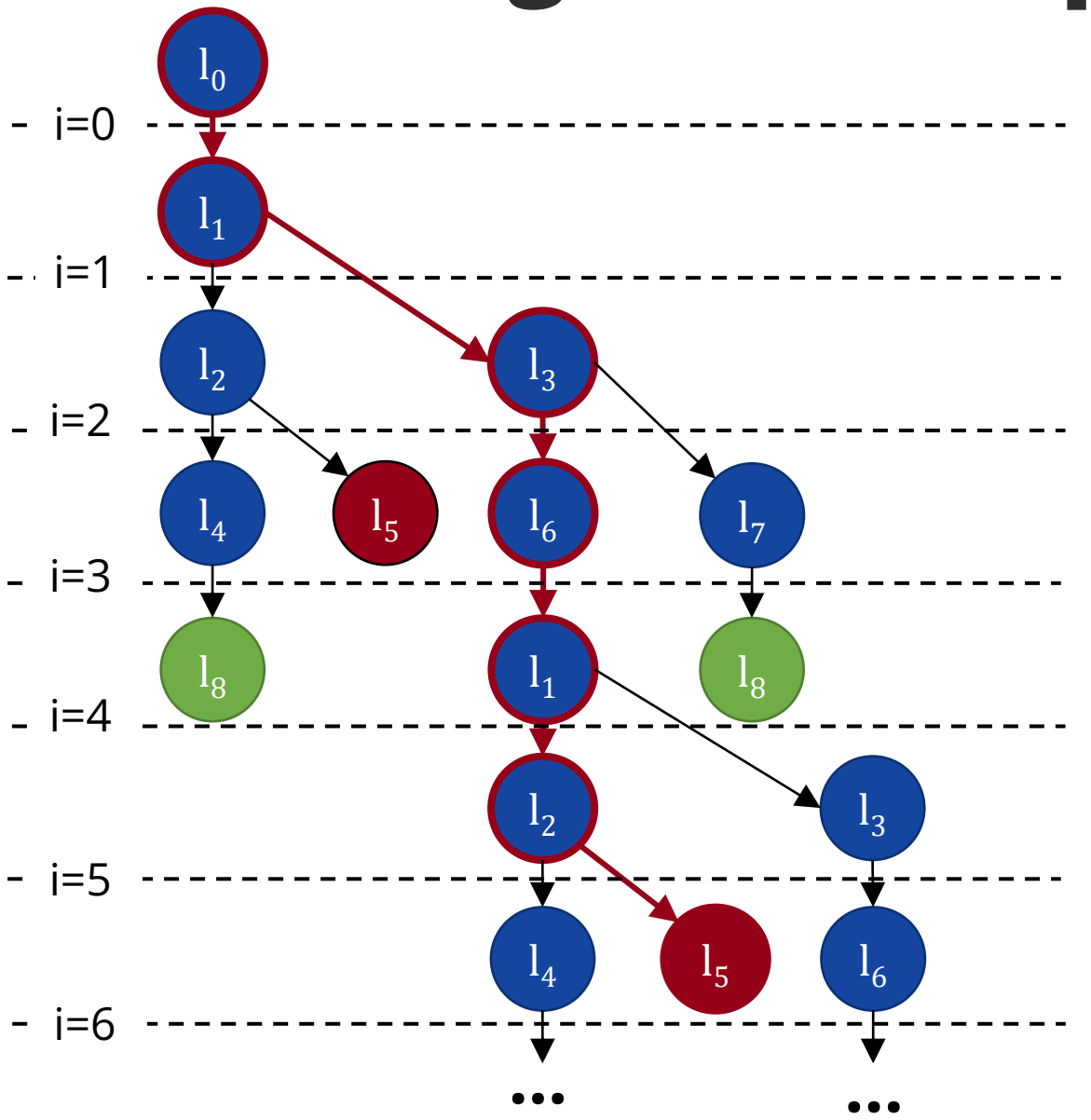
- **BFS-like search**
  - Disadvantage of DFS: early unrolling of loops
  - Search in the *state space of the program*, not in the *nodes of the CFA* → visited locations can be visited again

Searching for error paths



- **BFS-like search**
  - Disadvantage of DFS: early unrolling of loops
  - Search in the *state space of the program*, not in the *nodes of the CFA* → visited locations can be visited again

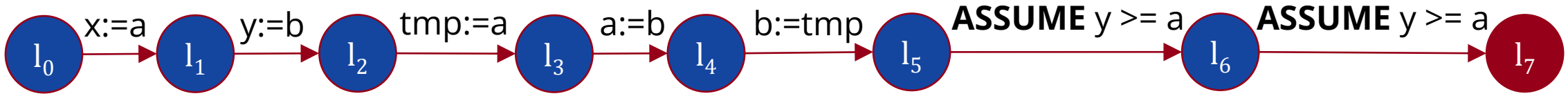
Searching for error paths



- **BFS-like search**
  - Disadvantage of DFS: early unrolling of loops
  - Search in the *state space of the program*, not in the *nodes of the CFA* → visited locations can be visited again

Feasibility checking through SMT

- An error path is found in the CFA → we need to check whether variable values can be chosen such that the execution follows that path




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

Encode variable "versions" using indices (Single Static Assignment)

$$x_0 = a_0$$
$$y_0 = b_0$$
$$tmp_0 = a_0$$
$$a_1 = b_0$$
$$b_1 = tmp_0$$
$$y_0 \geq a_1$$
$$x_0 \geq b_1$$

SMT problem: $x_0 = a_0 \wedge y_0 = b_0 \wedge tmp_0 = a_0 \wedge a_1 = b_0 \wedge b_1 = tmp_0 \wedge y_0 >= a_1 \wedge x_0 >= b_1$

Feasibility checking through SMT

SMT problem:

$$x_0 = a_0 \wedge y_0 = b_0 \wedge tmp_0 = a_0 \wedge a_1 = b_0 \wedge b_1 = tmp_0 \wedge y_0 >= a_1 \wedge x_0 >= b_1$$

Satisfiable:
Feasible error path found, SMT model contains the appropriate variable values

Unsatisfiable:
Search for other paths

BMC Workflow - Tasks