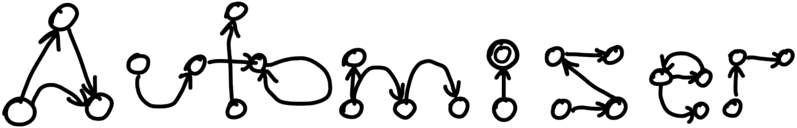
ULTIMATE

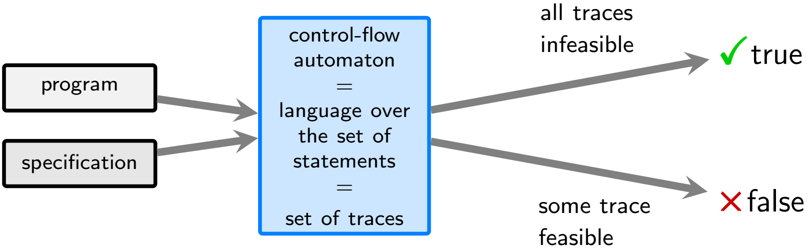


automata-based software verification

for

non-reachability, **memory safety**, **termination**, **overflows**, **race detection**

2024 competition team: **Matthias Heizmann, Manuel Bentele, Daniel Dietsch, Xinyu Jiang, Dominik Klumpp, Frank Schüssele, Andreas Podelski**



program

specification

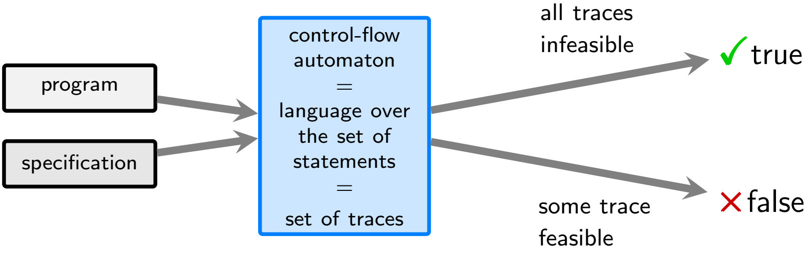
control-flow automaton = language over the set of statements = set of traces

all traces infeasible

✓ true

some trace feasible

✗ false



program

specification

control-flow automaton = language over the set of statements = set of traces

all traces infeasible

✓ true

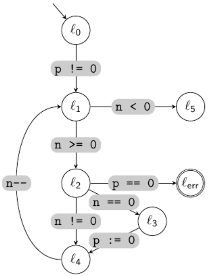
some trace feasible

✗ false

code + specification

```
int main() {
    int p, n;
    p = 42;
    while ( n>=0 ) {
        //@ assert p != 0;
        if (n == 0) {
            p = 0;
        }
        n--;
    }
    return 0;
}
```

CFA

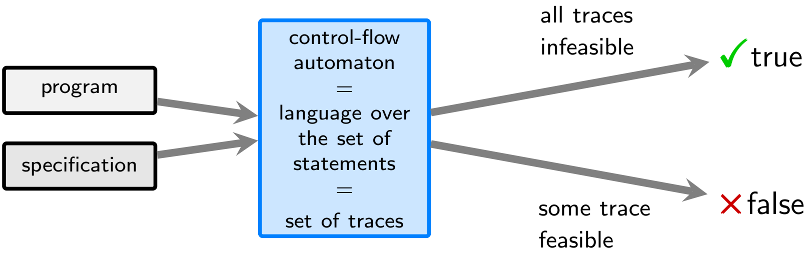


Alphabet

$\Sigma = \{$ `p != 0`, `n >= 0`, `n == 0`, `p := 0`, `n != 0`, `p == 0`, `n--`, `n < 0`, $\}$

Some trace

`p != 0` `n >= 0` `p == 0`



program

specification

control-flow automaton = language over the set of statements = set of traces

all traces infeasible

✓ true

some trace feasible

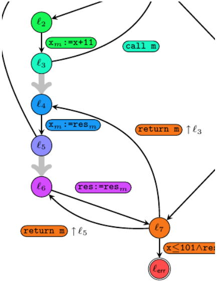
× false

Verification Algorithm

- pick trace π from L
- analyze feasibility of π
- generalize from π to set of traces Π
- subtract Π from L
- repeat until language L is empty

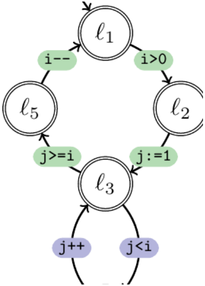
interprocedural
analysis

visibly pushdown
automata



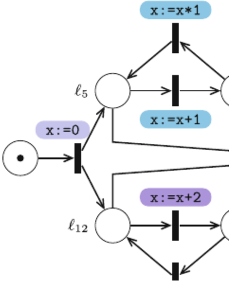
termination
analysis

Büchi
automata



concurrent
programs

bounded
Petri nets


x:=x*1
ℓ_5
x:=x+1
x:=0
ℓ_{12}
x:=x+2