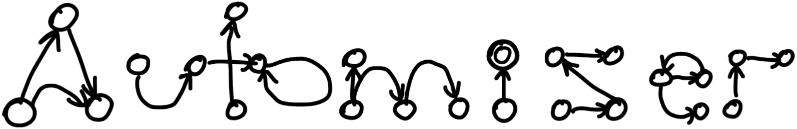
ULTIMATE



automata-based software verification

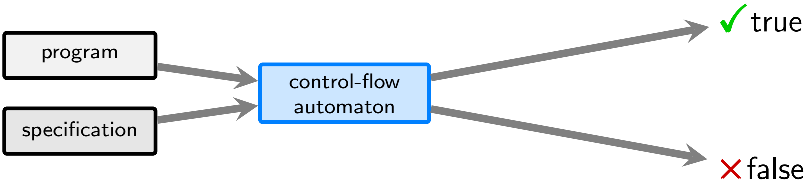
for

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

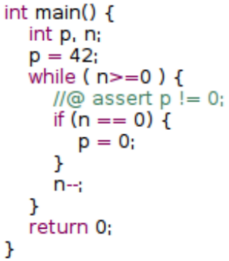
2025 competition team: **Marcel Ebbinghaus Matthias Heizmann,**
Manuel Bentele, Daniel Dietsch, Dominik Klumpp,
Frank Schüssele, Andreas Podelski

decomposition

of verification problem

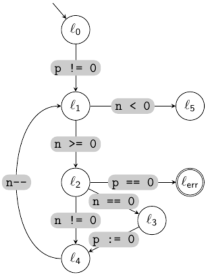


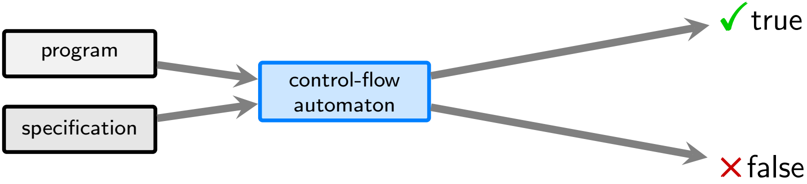
code + specification



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

CFA

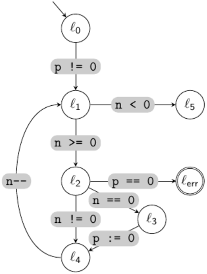




code + specification

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

CFA

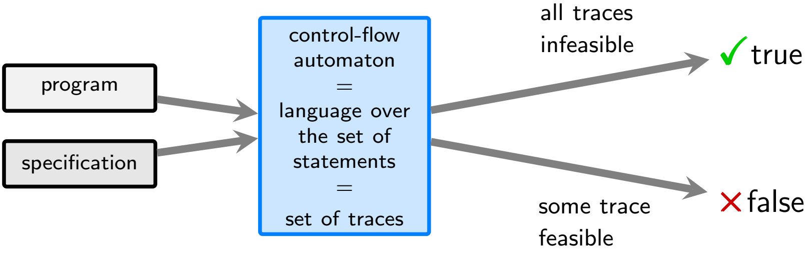


Alphabet

$\Sigma = \{$ `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

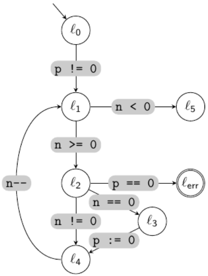
some trace feasible

✗ false

code + specification

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

CFA



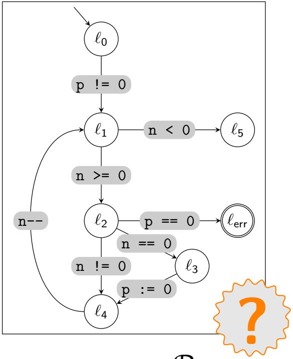
Alphabet

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

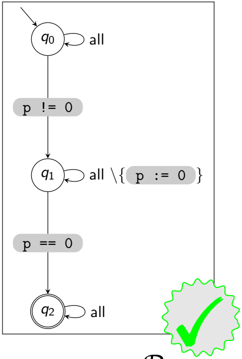
Some trace

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

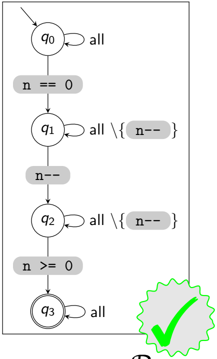
Decomposition: Example



program $\mathcal{P}$



program $\mathcal{P}_1$

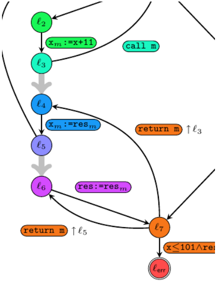


program $\mathcal{P}_2$

$$\mathcal{P} \quad \subseteq \quad \mathcal{P}_1 \cup \mathcal{P}_2$$

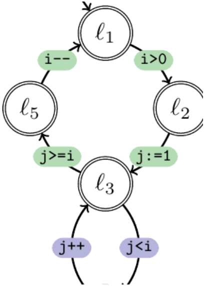
interprocedural
analysis

visibly pushdown
automata



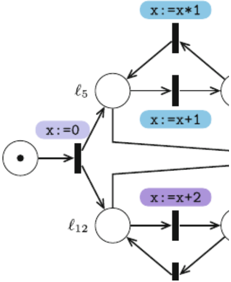
termination
analysis

Büchi
automata



concurrent
programs

bounded
Petri nets



x:=x*1
ℓ_5
x:=x+1
x:=0
x:=x+2
ℓ_{12}