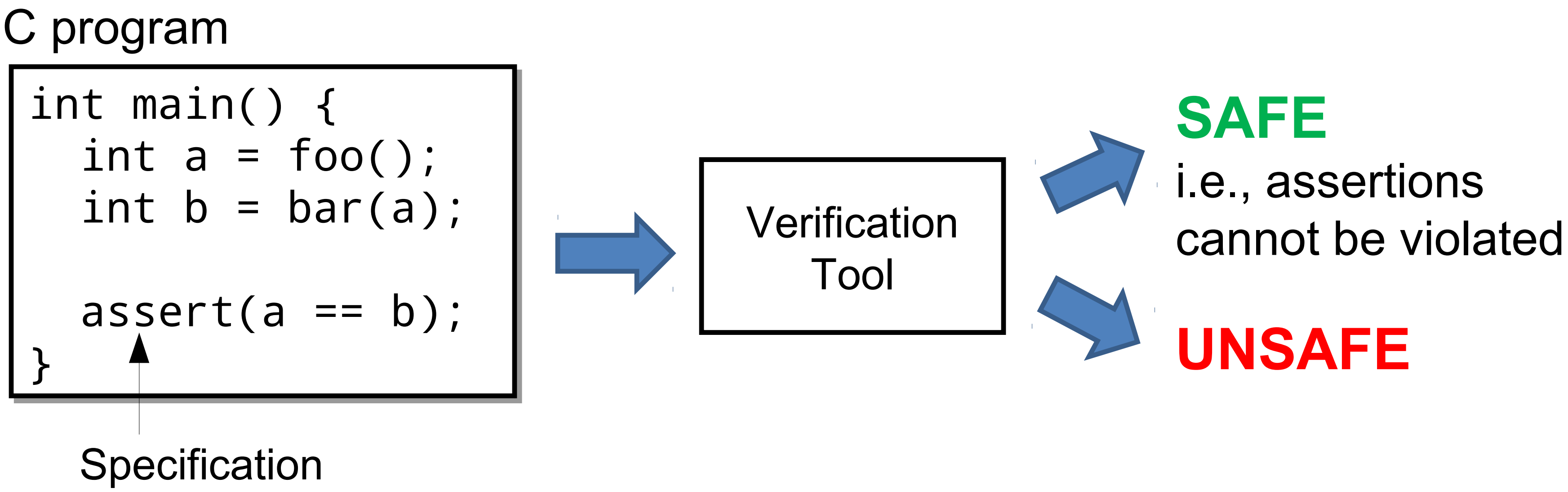
Conditional Model Checking

Philipp Wendler

Joint work with Dirk Beyer,
Tom Henzinger, Erkan Keremoglu

UNIVERSITÄT
PASSAU

Software Verification



C program

```
int main() {
  int a = foo();
  int b = bar(a);

  assert(a == b);
}
```

Specification

Verification Tool

SAFE

i.e., assertions cannot be violated

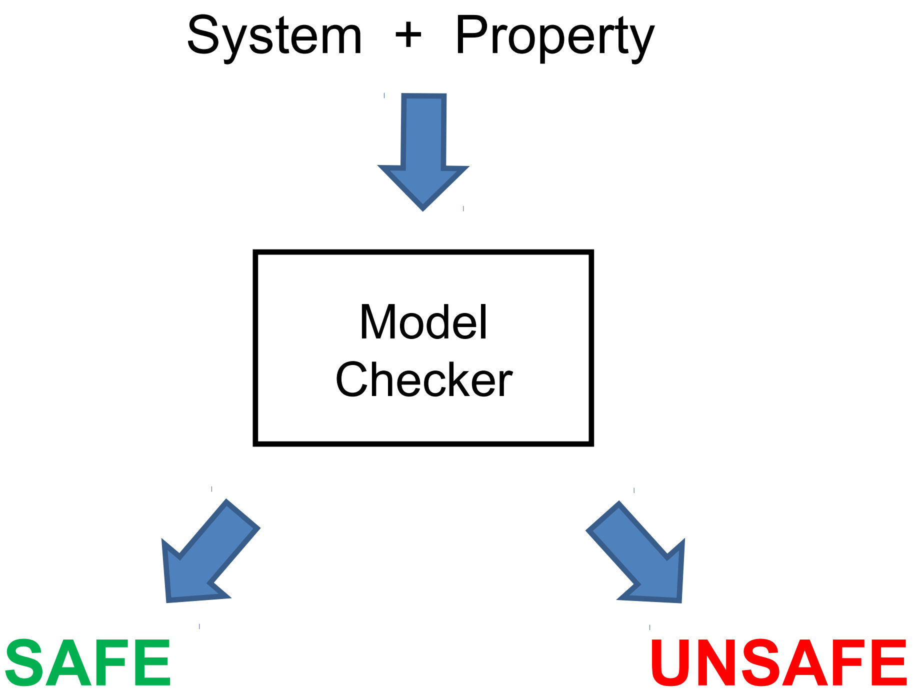
UNSAFE

Problem:
Single Analysis not Effective

```
void main() {
  if (nondet_int()) {
    int i;
    for (i = nondet_int(); i < 1000000; i++) {
      // ...
    }
    assert(i >= 1000000);

  } else {
    int x = 5;
    int y = 6;
    int r = x * y;
    assert(r >= x);
  }
}
```

Model Checking



System + Property

Model Checker

SAFE

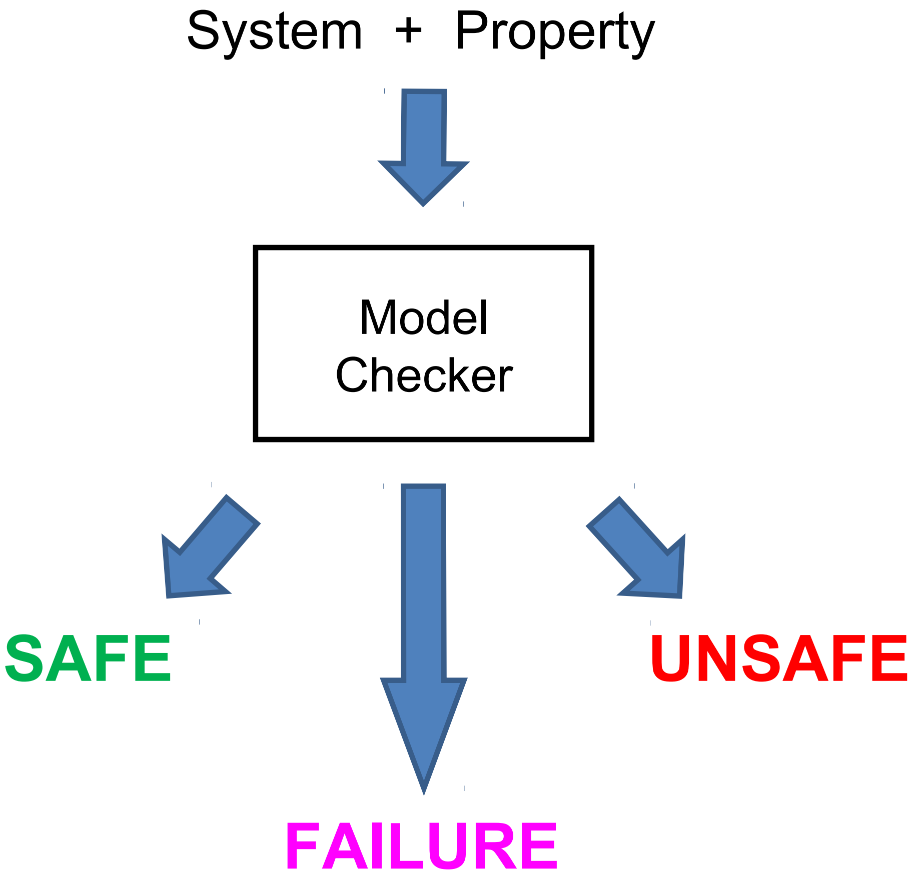
UNSAFE

ACM Turing Award 2007

- Edmund Clarke
- Allen Emmerson
- Joseph Sifakis

Invention: “Model Checking”

Classic Model Checking



System + Property

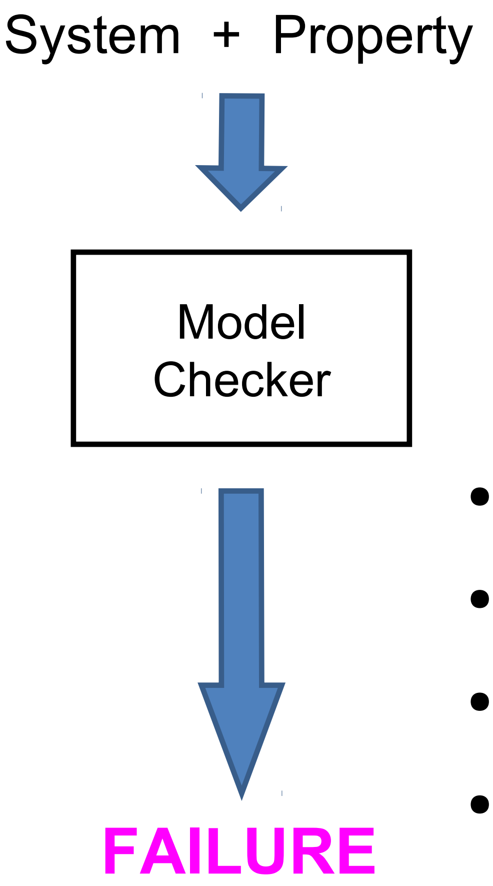
Model Checker

SAFE

FAILURE

UNSAFE

Classic Model Checking



System + Property

Model Checker

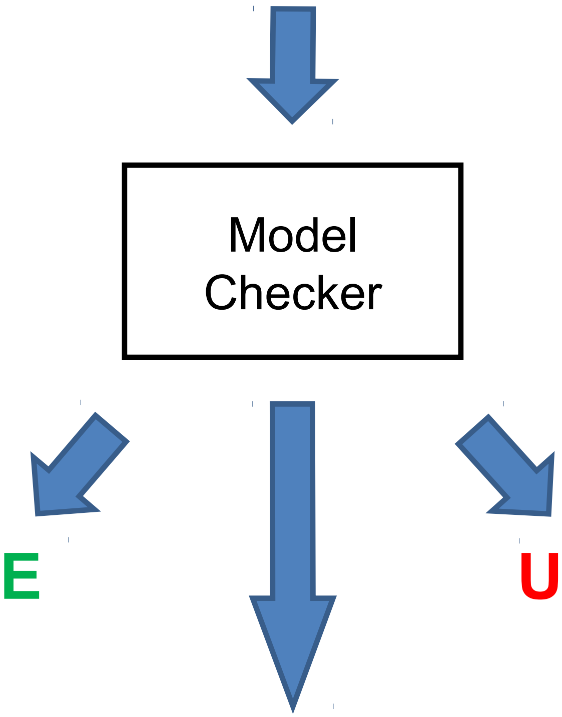
FAILURE

- Timeout
- Out of memory
- Crash of component
- Operand exception

Enormous amounts of resources **wasted**!

Classic Model Checking

System + Property



Model Checker

Unsound
(overflows)

→ False negatives
(missed bugs = disaster)

Incomplete
(non-linear operations)

→ False positives
(false alarms = noise)

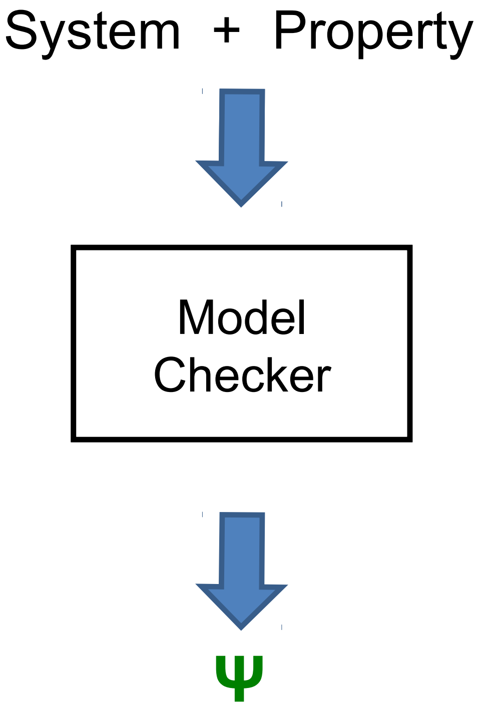
SAFE

UNSAFE

FAILURE

Conditional Model Checking

Conditional Model Checking



System + Property

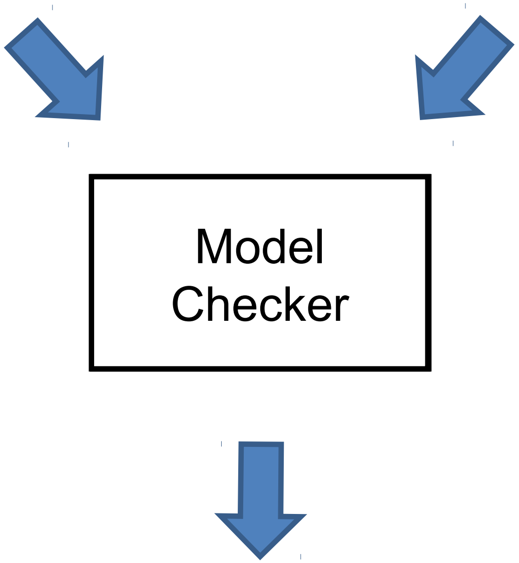
Model Checker

Ψ

("SAFE under Condition Ψ")

Examples:	- **Ψ =** true:	previous SAFE
	- **Ψ =** false:	previous UNSAFE
	- general:	condition for safety

Conditional Model Checking



System + Property

Condition Ψ_0

Directs the analysis
to parts to analyze

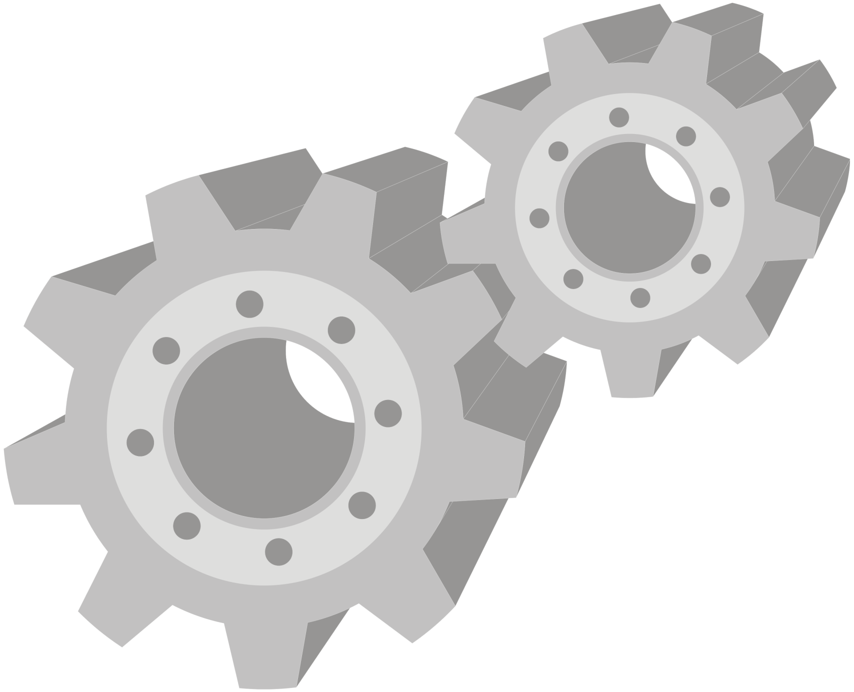
Model
Checker

Ψ

(“SAFE under Condition Ψ”)

Examples:	- **Ψ =** true:	previous SAFE
	- **Ψ =** false:	previous UNSAFE
	- general:	condition for safety

Applications of
Conditional Model Checking



Back to Our Example

```
void main() {
  if (nondet_int()) {
    int i;
    for (i = nondet_int(); i < 1000000; i++) {
      // ...
    }
    assert(i >= 1000000);

  } else {
    int x = 5;
    int y = 6;
    int r = x * y;
    assert(r >= x);
  }
}
```

Back to Our Example

To show:

$$M \models \Phi$$

In this case:

$$\Phi = \Phi_1 \;\&\; \Phi_2$$

with Φ_1 = “loop is correct”

and Φ_2 = “multiplication is correct”

Idea

- Verify Φ_1 (“loop is correct”)

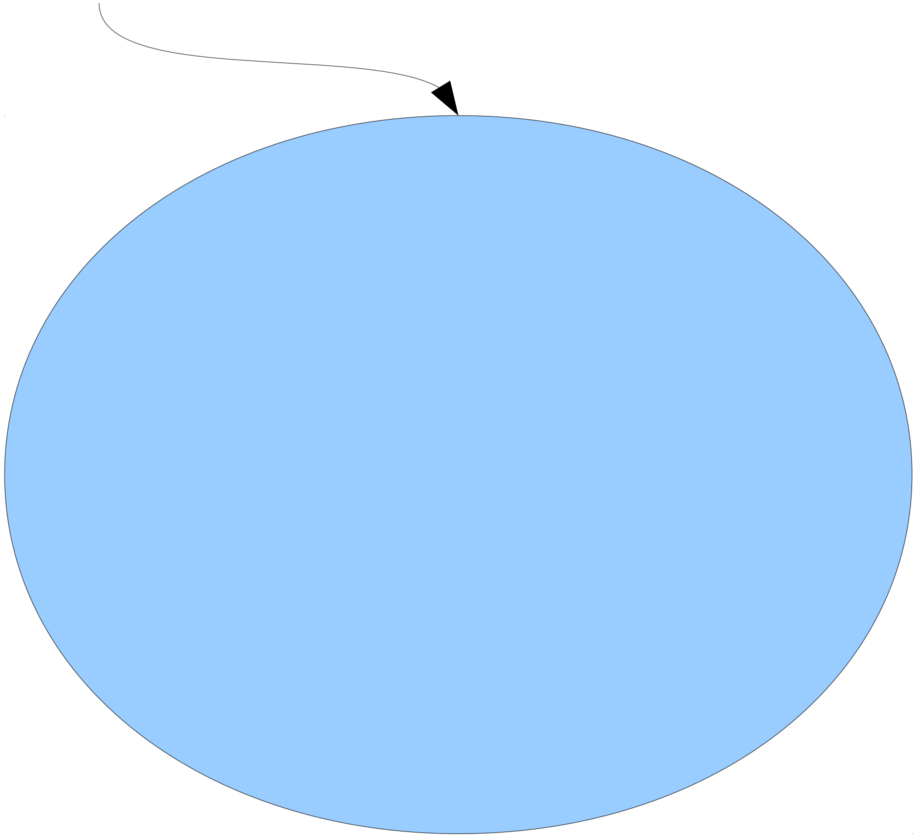
  → use predicate analysis
- Verify Φ_2 (“multiplication is correct”)

  → use explicit-state analysis
- Final result: Φ verified

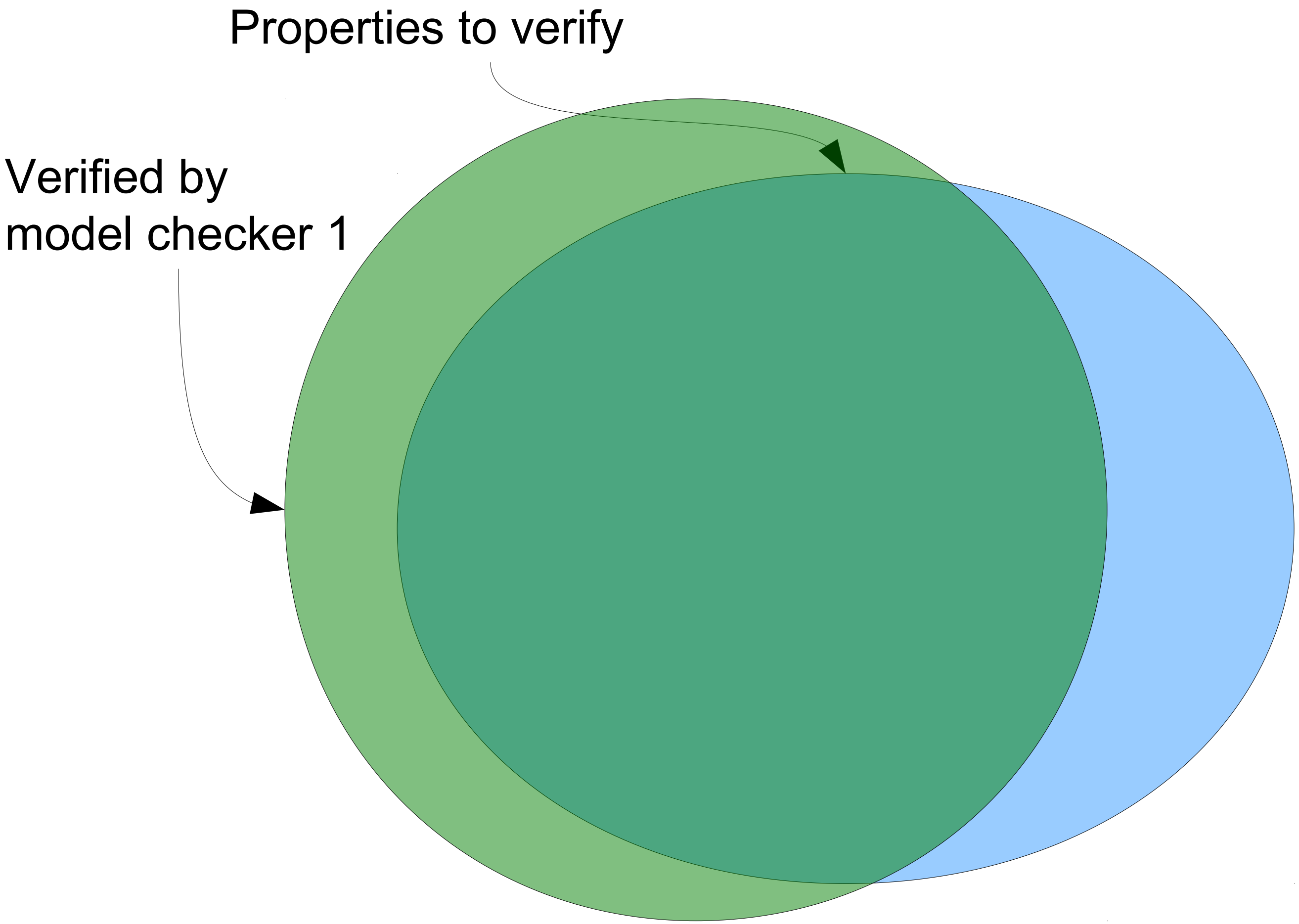
Using CMC with Input Conditions

- Tell model checker what to verify
- In our example:
  - For conditional model checker 1: verify Φ_1
  - For conditional model checker 2: verify Φ_2
  - Full verification possible

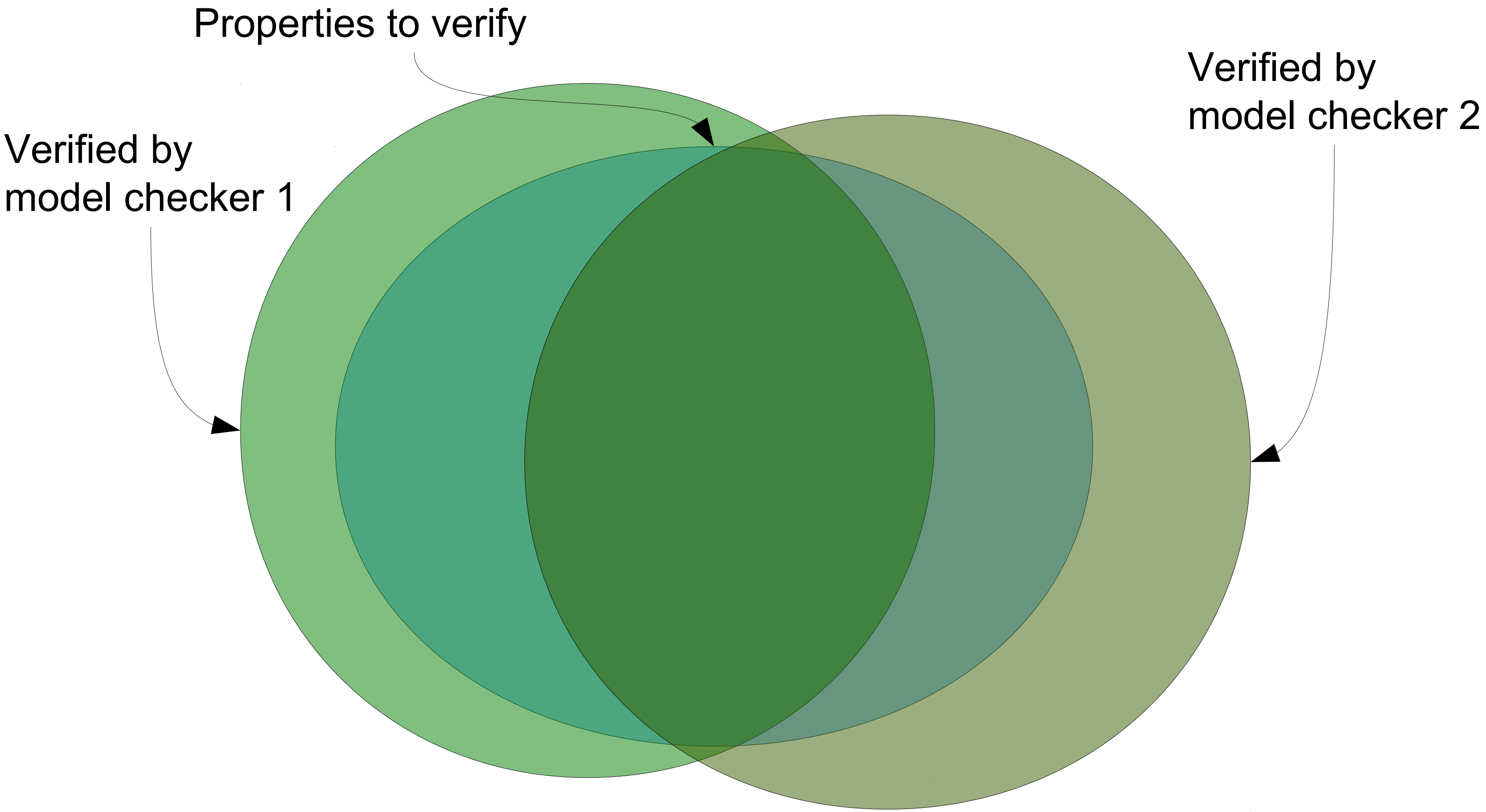
More General:


Properties to verify

More General:



More General:



Properties to verify

Verified by model checker 1

Verified by model checker 2

Further Input Conditions

- Limit resources
  - Time
  - Memory
  - Model Checker will not crash, but terminate itself and give useful result
- Restrict the search
  - Loop bounds (a.k.a. “bounded model checking”)
  - Path length
  - Time spent on path
  - ...

Output Conditions

- Dump partial result if analysis didn't finish
  - Output cond. summarizes what could be verified
- Explicitly state assumptions used by MC
  - Example: “variable `x` does not overflow”
- Purpose:
  - Give information to the user
  - Verify condition with other methods (testing, manual proofs, …)
  - Comparison of checkers (weaker output condition is better)

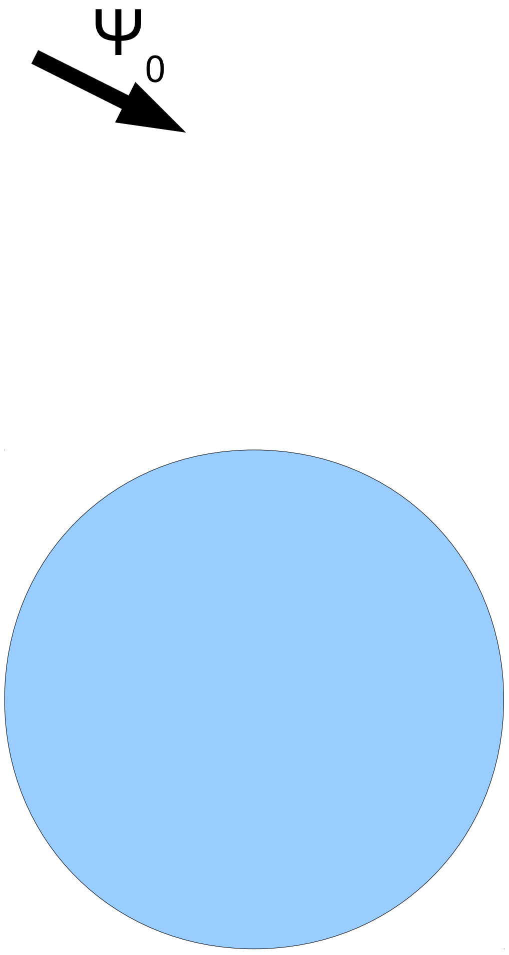
Sequential Composition

- In our example,
  we told the model checkers what to verify
- Now let them find out automatically!
- Conditional model checker 1 verifies what it can verify
- Conditional model checker 2 verifies remaining parts

Sequential Composition

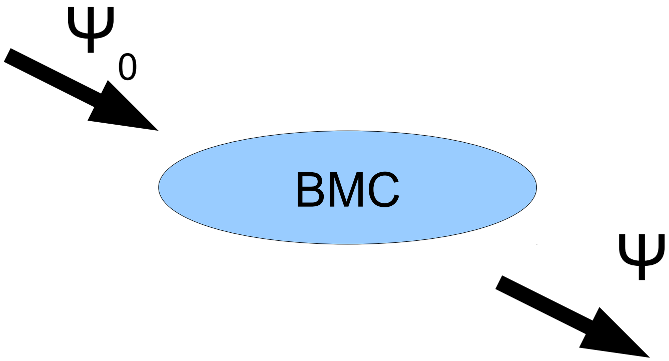
- Use input condition to limit resource usage of first analysis
- Use output condition
as input condition for next model checker
- Iterate until finished (or run out of tools)

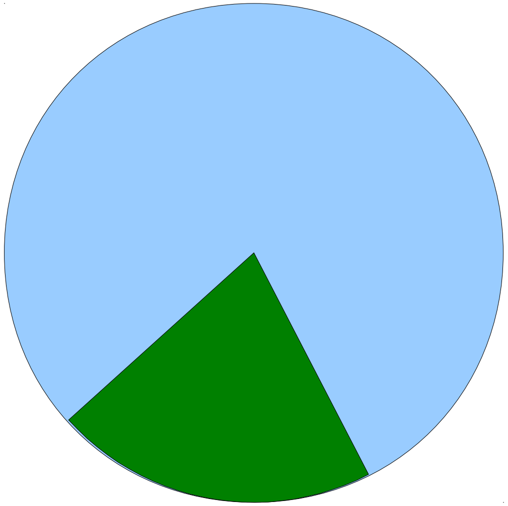
Sequential Composition



Property to verify

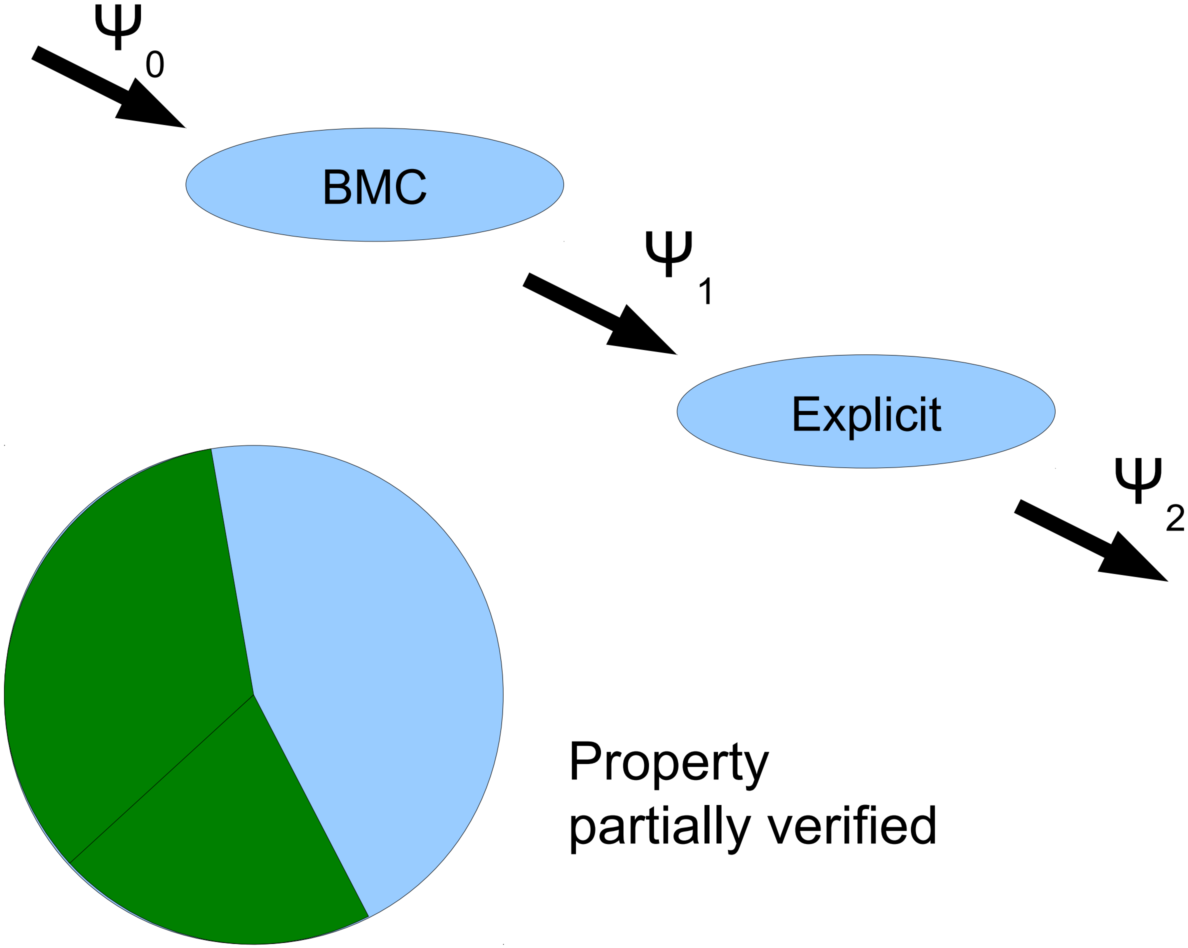
Sequential Composition





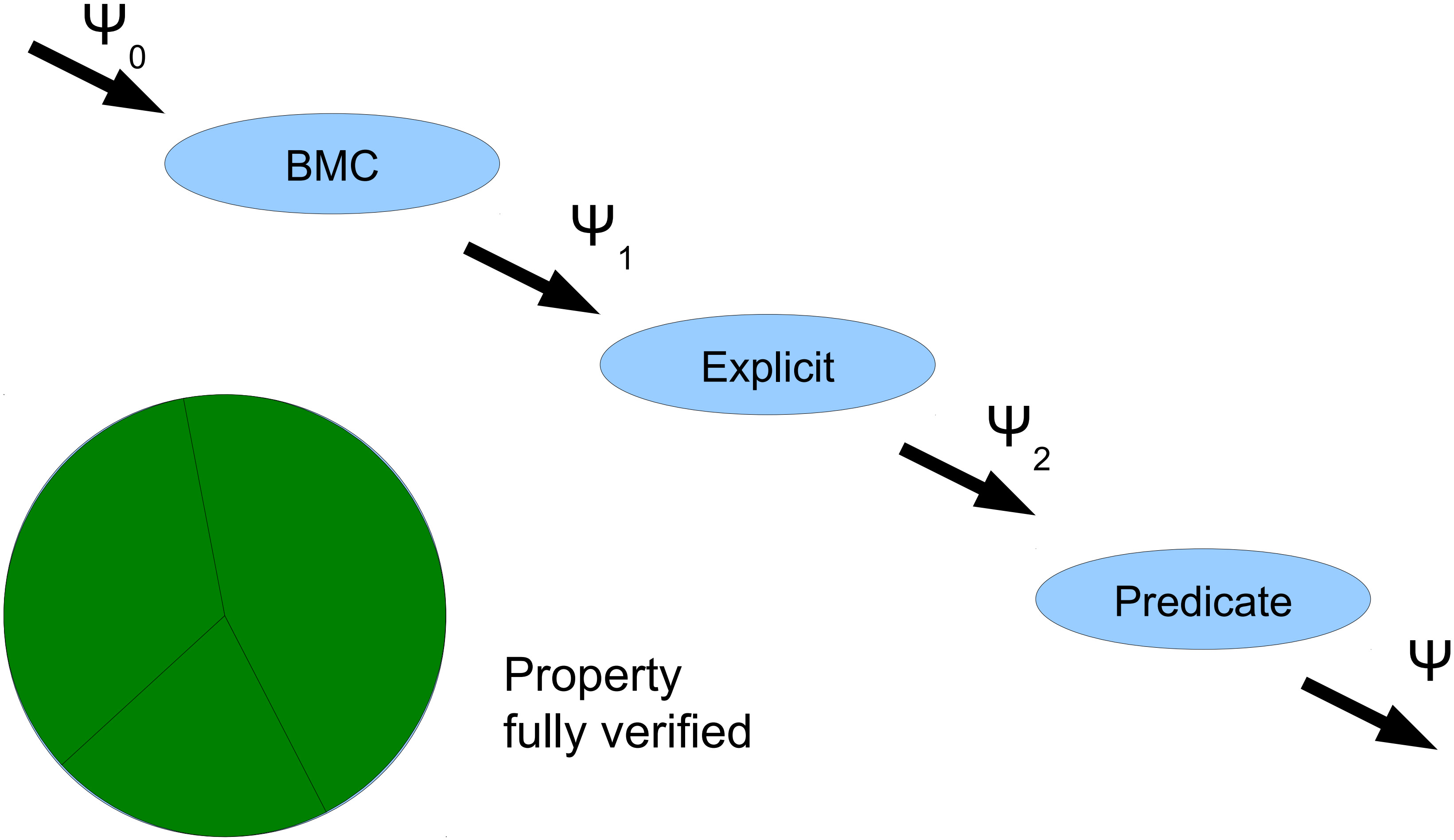
Property
partially verified

Sequential Composition



Property
partially verified

Sequential Composition



Property
fully verified

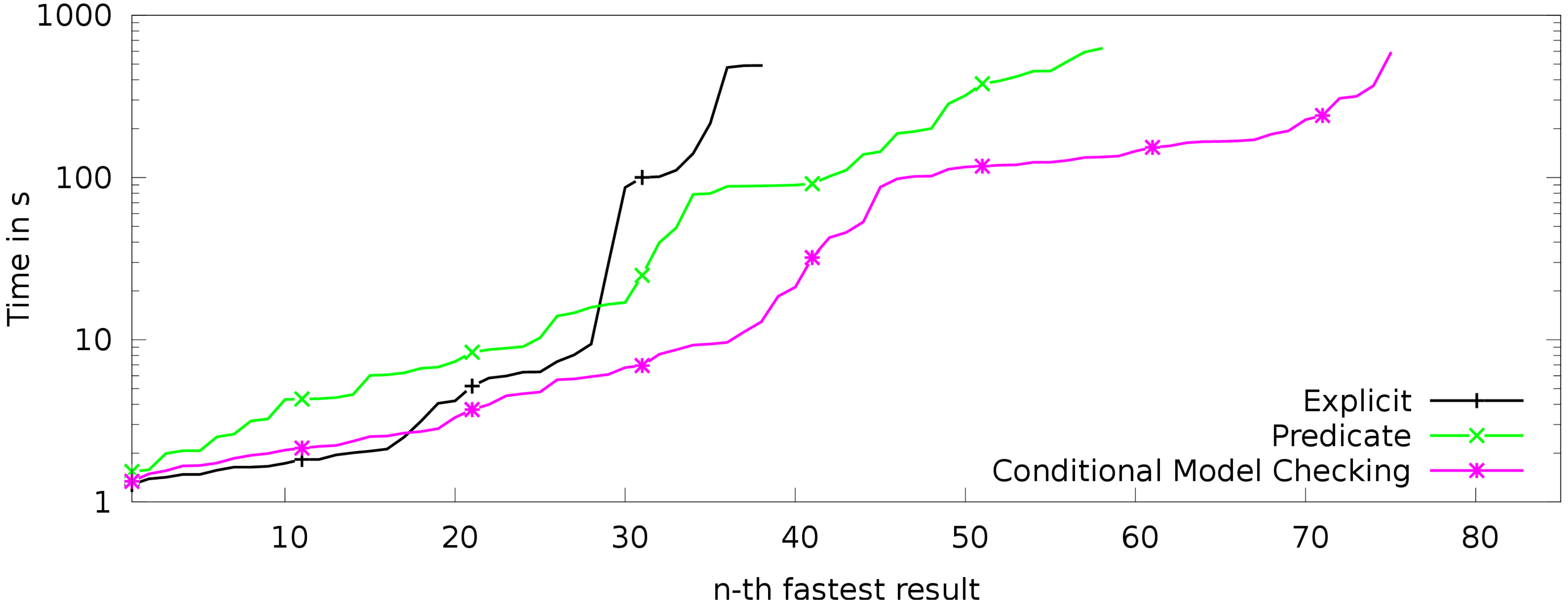
Experiment: Sequential Composition

- Implemented Conditional Model Checking in CPAchecker
- 85 C programs based on “hard” programs of Software Verification Competition 2012
- 15 min time, 15 GB RAM

Experiment: Sequential Composition

- A: Explicit-value analysis
- B: Predicate analysis
- C: Conditional model checking
  - First: explicit-value analysis
    with input condition: time limit = 100s
  - Second: predicate analysis
    with output condition of first analysis
    as input condition

Experiment: Sequential Composition

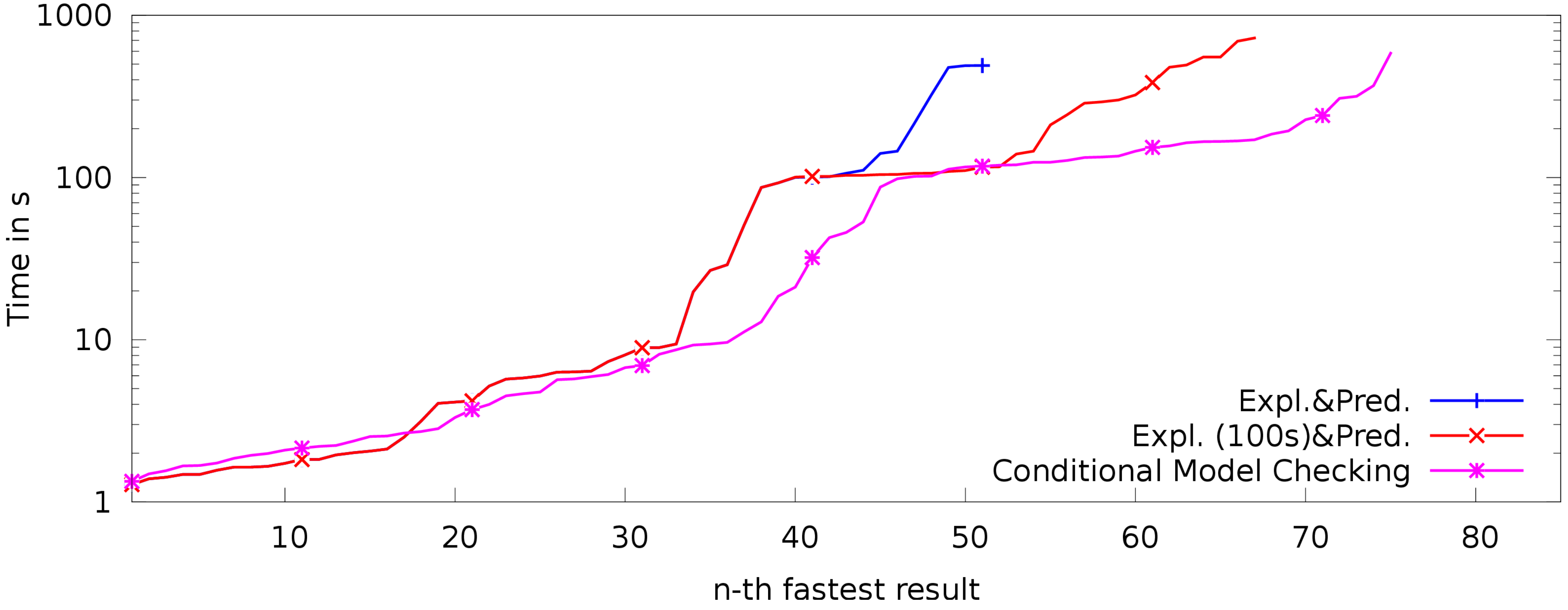


➔ Sequential composition solves more problems and is faster

Experiment: Sequential Composition

- A: Explicit-value analysis ; predicate analysis
- B: Explicit-value analysis ; predicate analysis
  - Input condition for first analysis:
    time limit = 100s
- C: Conditional model checking
  - First: explicit-value analysis
    with input condition: time limit = 100s
  - Second: predicate analysis
    with output condition of first analysis
    as input condition

Experiment: Sequential Composition



- ➔ Using conditional model checking for sequential composition is better

Summary

Conditional Model Checking:

- Terminates with useful results (no crashes)
- Enables partial / compositional verification
- Effective sequential composition (solve harder problems)
- Unified view on existing approaches