Reusing Precisions for Efficient Regression Verification

(Published in Proc. ESEC/FSE 2013, ACM.)

Dirk Beyer

Stefan Löwe

Evgeny Novikov

Andreas Stahlbauer

Philipp Wendler

UNIVERSITÄT PASSAU

ISP RAS

UNIVERSITÄT PASSAU

Regression

“We need this new feature, now!”

Risk of introducing bugs when changing source code

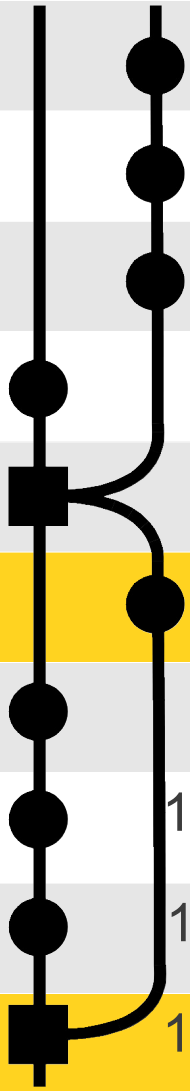
→ **Regression ~~Testing~~**
Verification

Linux Driver Verification

Picture: (c) Sean Bonner, 2013

Real-World Example

Revision	Commit Message	Safe?
3	Implement button detection support	✘
4	Free MICDET IRQ on error during probe	✘
5	fix typos in extcon-arizona	✘
6	Use bypass mode for MICVDD	✘
7	Merge tag ’driver-core-3.6’ of ...	✘
8	unlock mutex on error path in ...	✔
9	remove use of devexit	✘
10	remove use of devinit	✘
11	remove use of devexit p	✘
12	Merge tag ’pull req 20121122’ of ...	✔



High Resource Consumption!

Software Verification is expensive

Verifying all **safety properties** for
all **entry points** of all **revisions** of a software ...

```
        200 000 revisions
           * 10 properties
            * 5 entry points
  ---------------------------
= 10 000 000 verification tasks
            * 5 seconds/verification task
```

≈ 580 days

… is **really expensive**

Reuse of Verification Results

Drawbacks of existing approaches

- Too **large**: space on disk, time for loading
- Too **sensitive** to changes between revisions
- Too **complex**: modification of the verification algorithm

➞ **Reuse** the **“precision”**

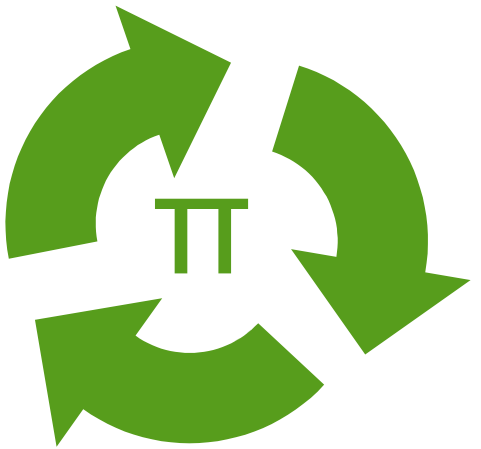
Precision Π

Defines the **level of abstraction** within an abstract domain:

> **Information** that an abstraction-based analysis has **to track** to prove a property.

Advantages of Reusing Precisions

- **No modification** of the verification algorithm
- **Easy to extract** from model checkers
- **Small** memory footprint
- **Low sensitivity** to changes in the input programs



Examples for Precision

- Predicate Analysis $\pi = \{a > 0, k == 1 \wedge e == 0\}$

  Set of predicates used to compute boolean abstractions

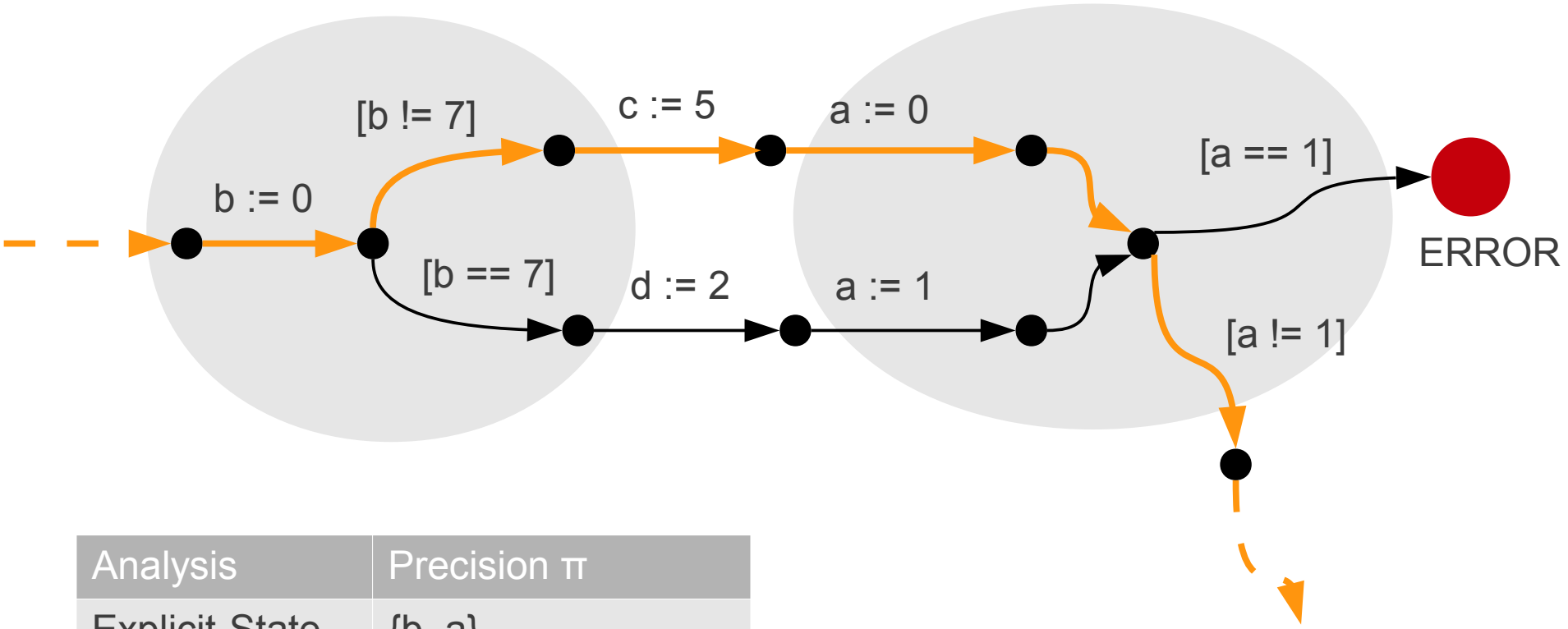
- Explicit-State Analysis $\pi = \{a, k, e\}$

  Set of variables for which the explicit value has to be tracked

- Shape Analysis $\pi = \{p1, p2\}$

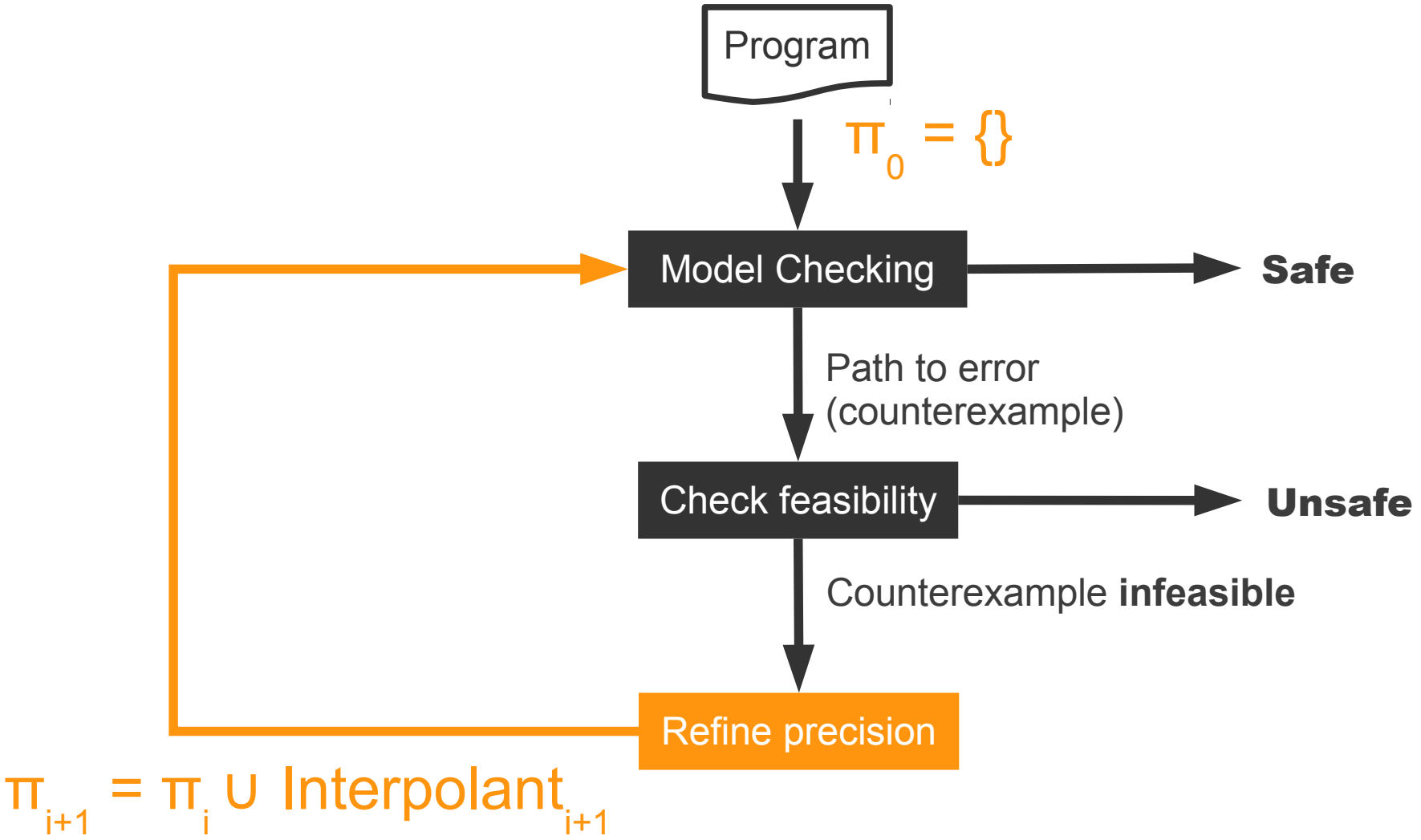
  Set of pointer variables to track

Example

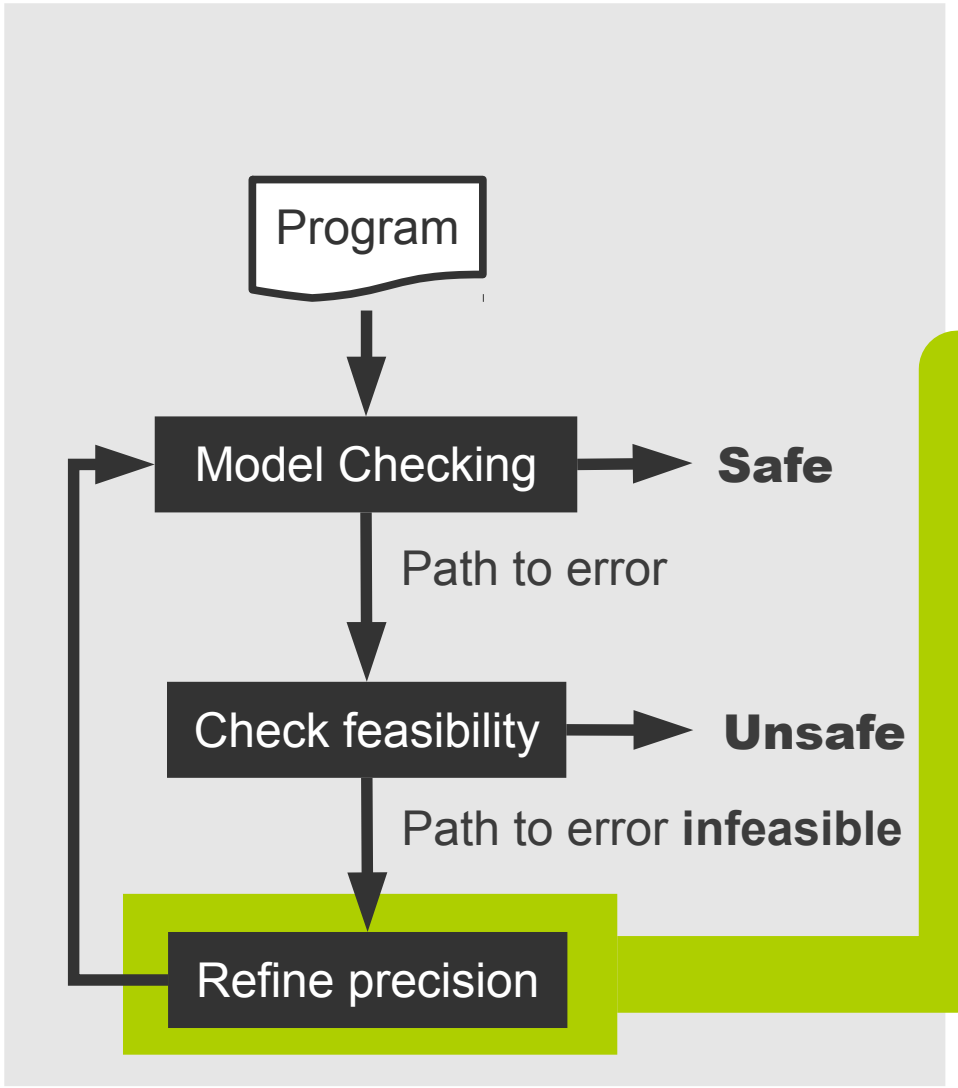


Analysis	Precision π
Explicit-State	{b, a}
Predicate	{b == 7, a == 1}

CEGAR


Program
$\pi_0 = \{\}$
Model Checking
Safe
Path to error
(counterexample)
Check feasibility
Unsafe
Counterexample **infeasible**
Refine precision
$\pi_{i+1} = \pi_i \cup \text{Interpolant}_{i+1}$

Costs of one (more) Iteration

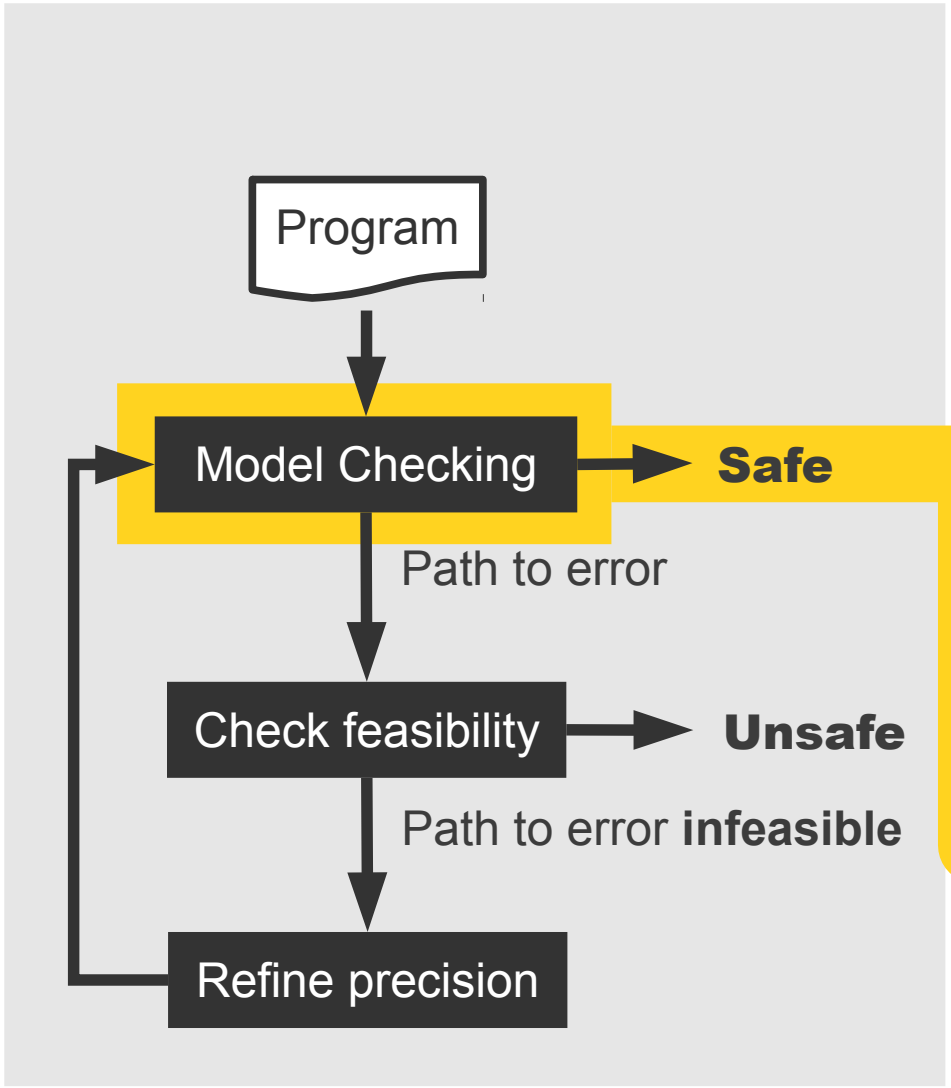


Interpolation for refining the precision of relevant program locations

Recomputing affected **abstract states**

Cut abstract reachability graph on pivot state

Costs of one (more) Iteration

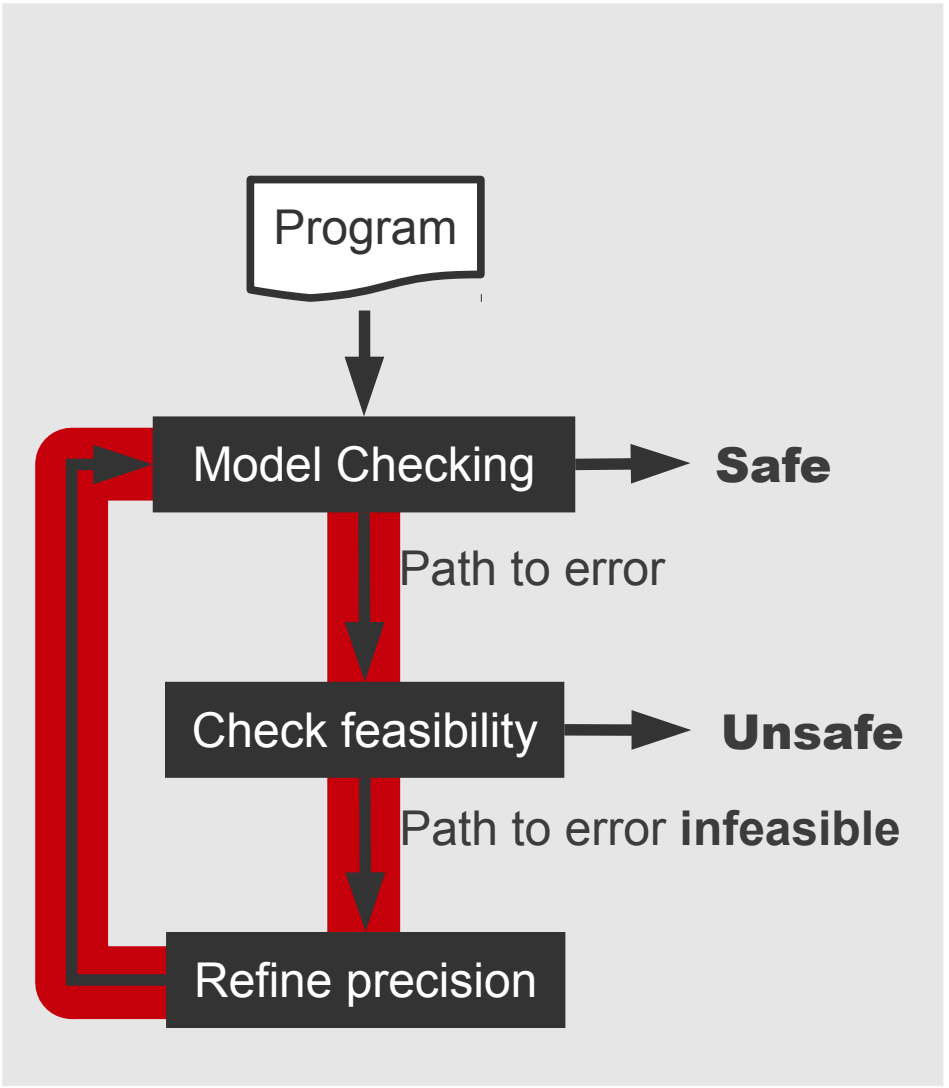


Interpolation for refining the precision of relevant program locations

Recomputing affected **abstract states**

Cut abstract reachability graph on pivot state

Costs of one (more) Iteration

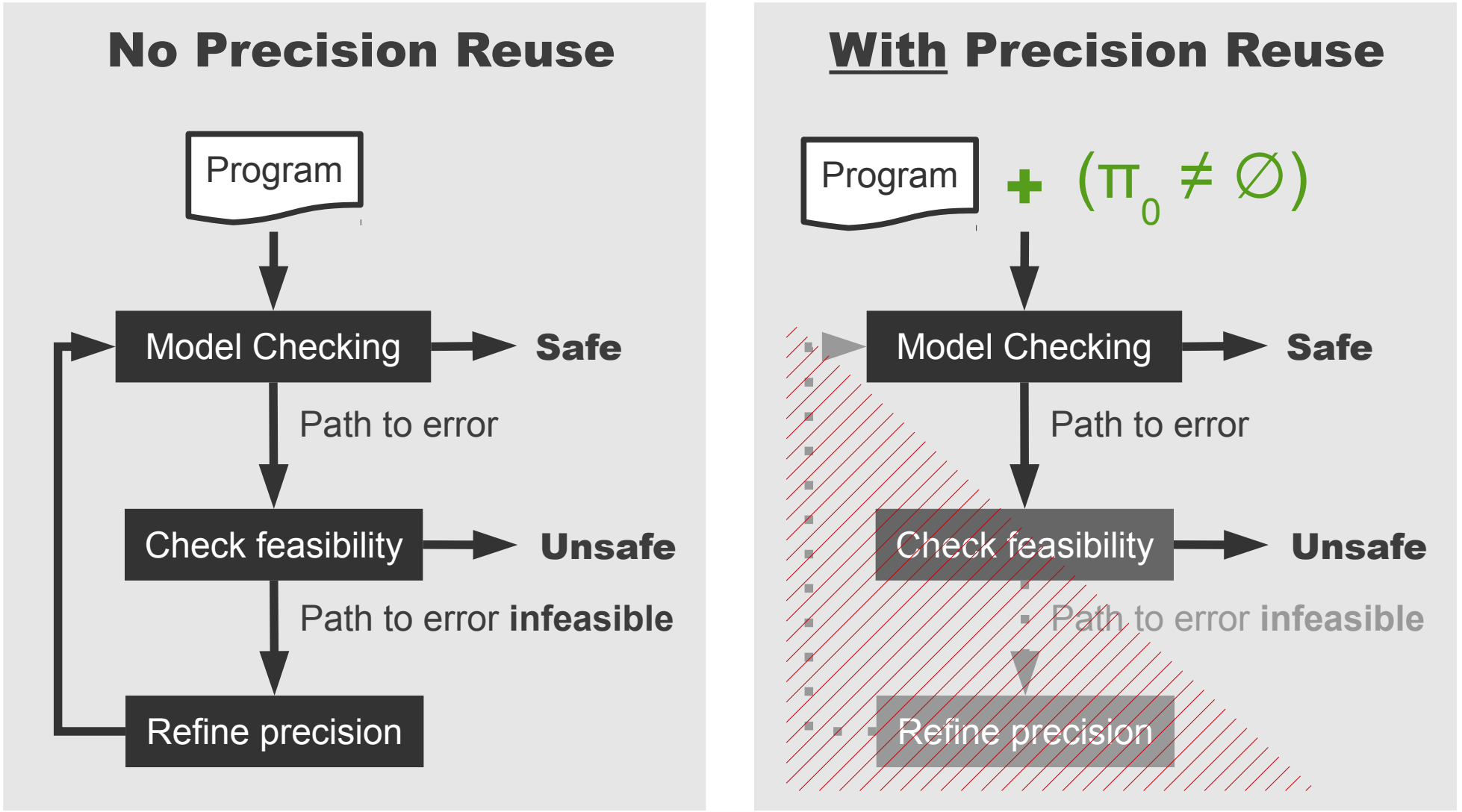


Interpolation for refining the precision of relevant program locations

Recomputing affected **abstract states**

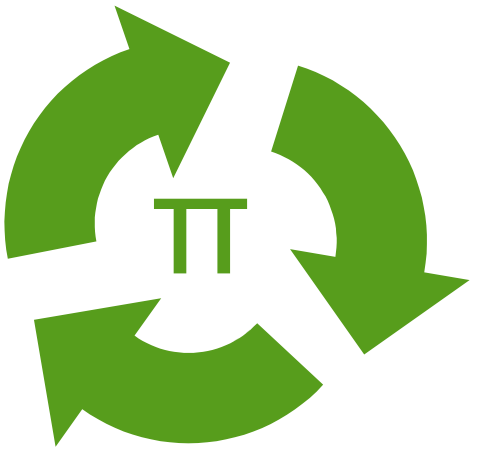
Cut abstract reachability graph on pivot state

Costs of one (more) Iteration



Advantages of Reusing Precisions

- **No modification** of the verification algorithm
- **Easy to extract** from model checkers
- **Small** memory footprint
- **Low sensitivity** to changes in the input programs

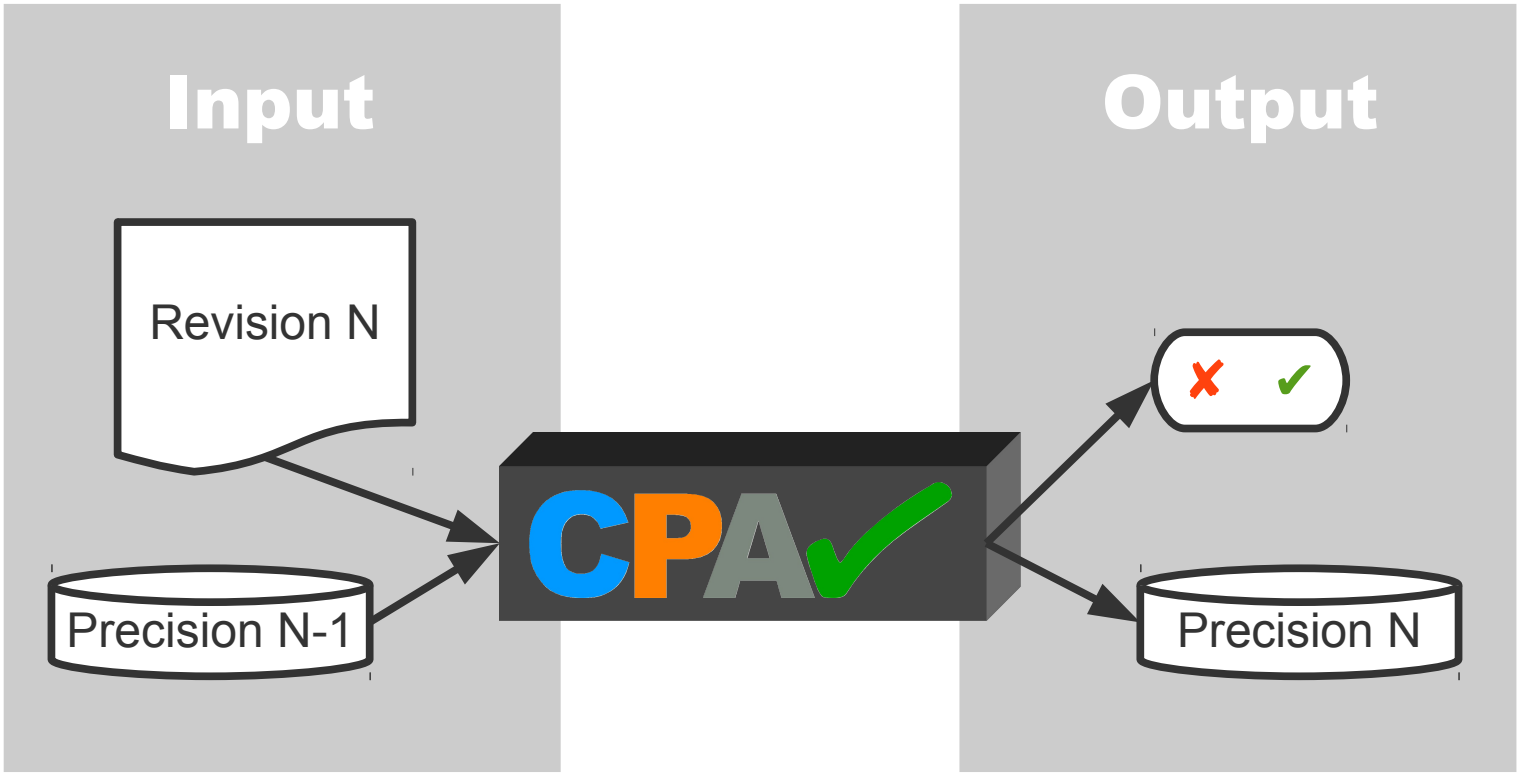


Implementation

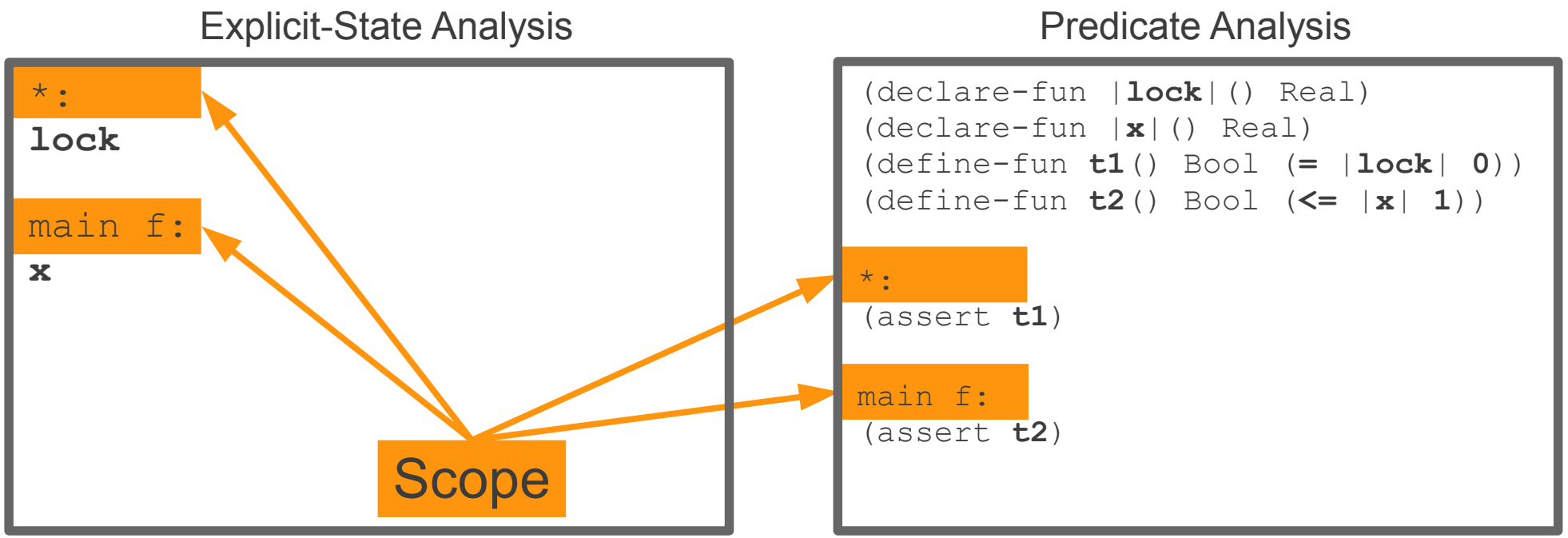
http://cpachecker.sosy-lab.org

- Implemented in **CPAchecker**
  - Predicate Analysis
  - Explicit-State Analysis
- Common to both analyses:
  - Lazy abstraction
  - CEGAR
  - Construct an abstract reachability graph

Workflow

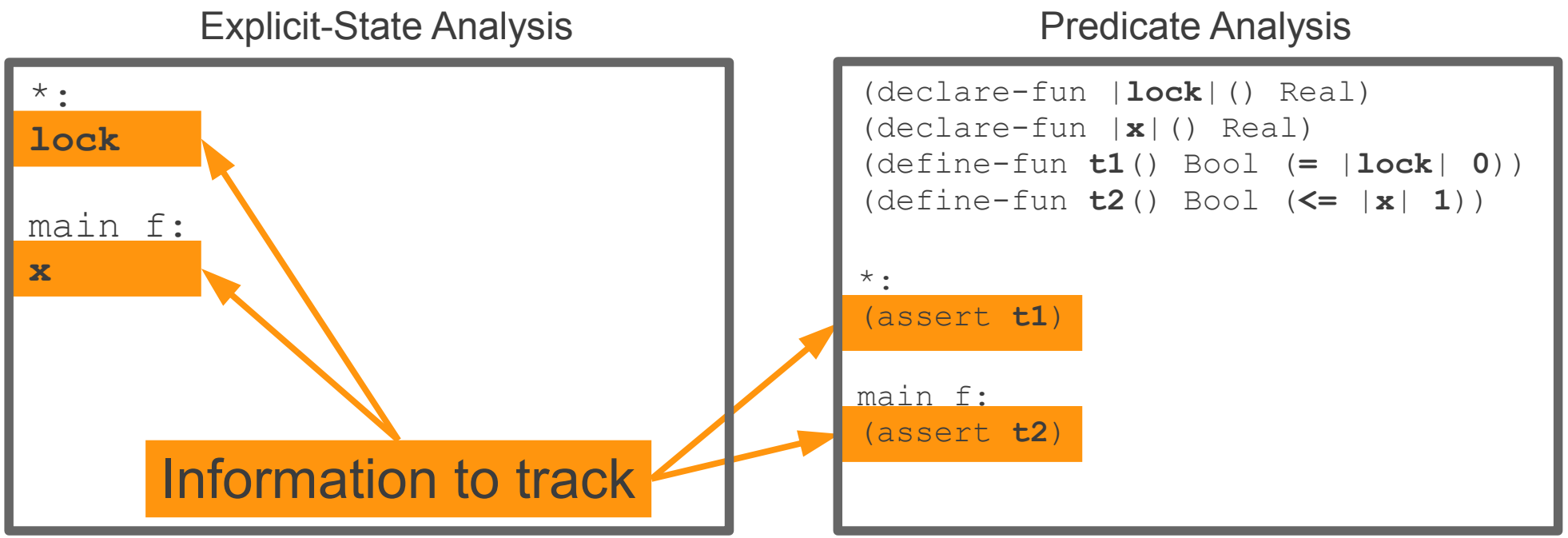


Storing Precisions



Really simple! **Dump the precision** if you have it!

Storing Precisions



Really simple! **Dump the precision** if you have it!

Storing Precisions

Explicit-State Analysis

```
*:
lock

main f:
x
```

Predicate Analysis

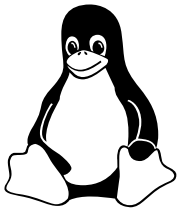
```
(declare-fun |lock|() Real)
(declare-fun |x|() Real)
(define-fun t1() Bool (= |lock| 0))
(define-fun t2() Bool (<= |x| 1))

*:
(assert t1)

main f:
(assert t2)
```

Global declarations and definitions

Really simple! **Dump the precision** if you have it!

Benchmark Suite

- Derived from industrial code (Linux kernel)
  - **4193** verification problems
  - **59** Linux device drivers
  - **1119** revisions
    spanning more than **5 years** of development



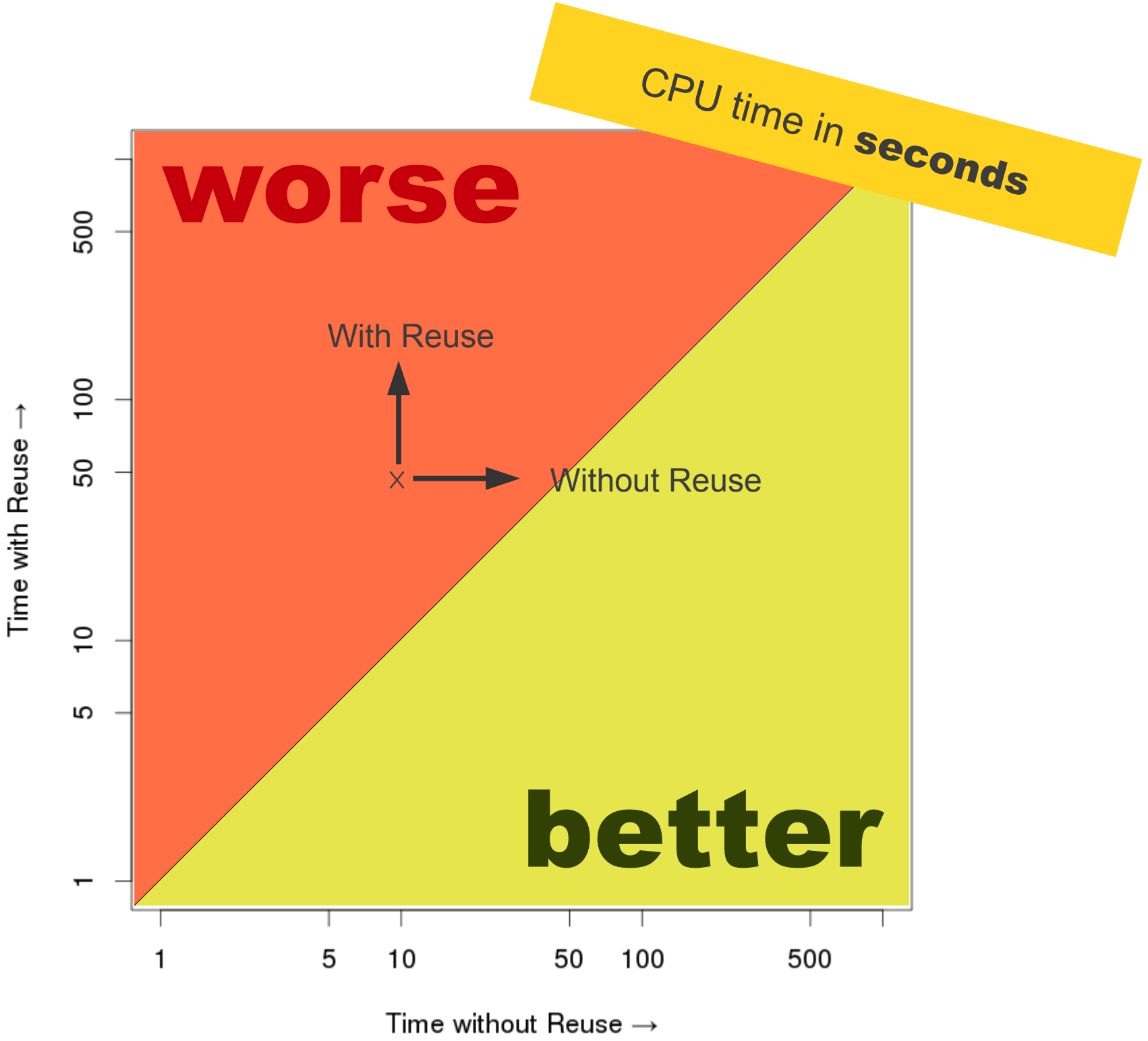
- Publicly available

http://sosy-lab.org/~dbeyer/cpa-reuse/

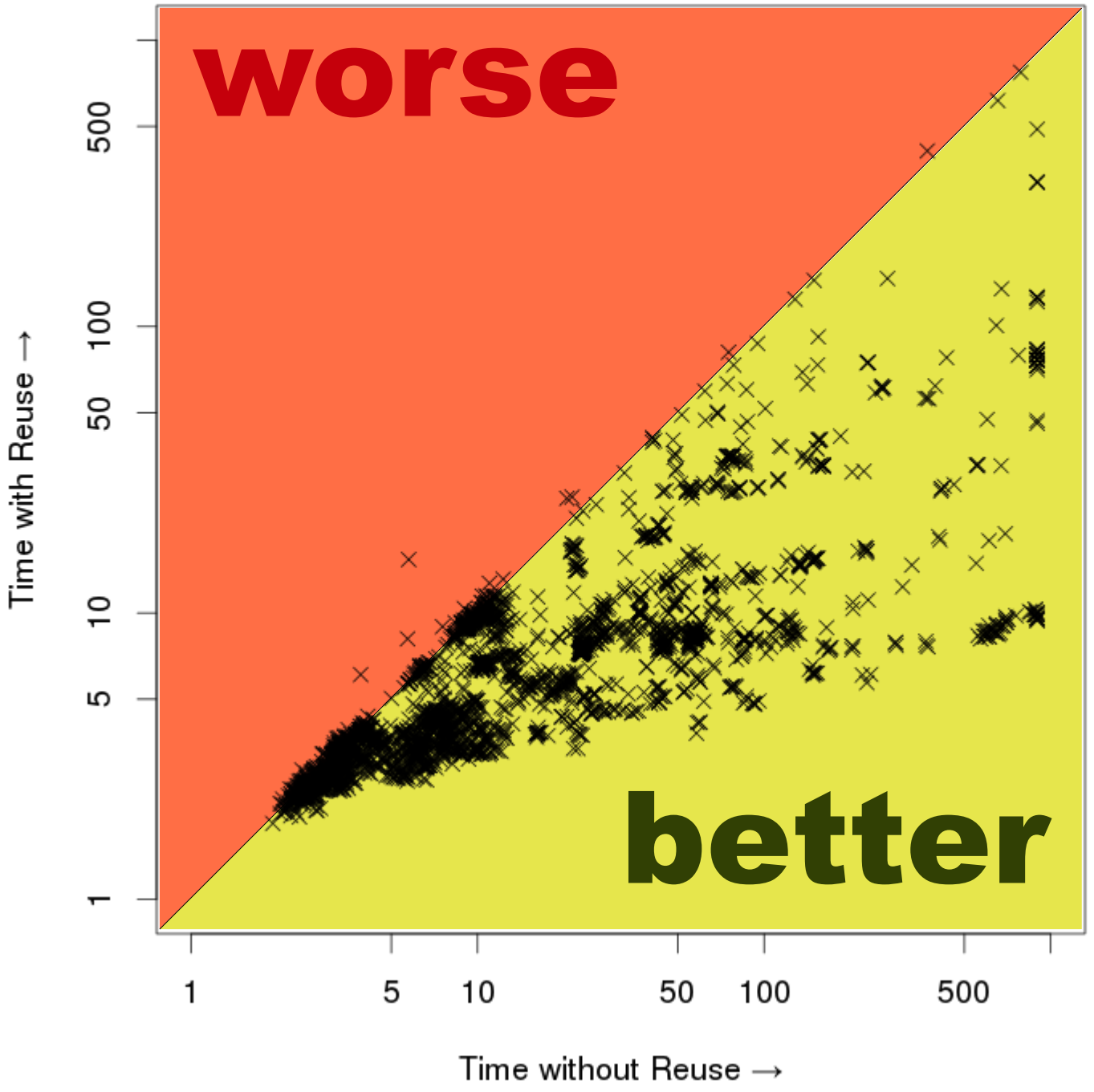
Benchmark Setup

- Processor: Intel i7 3.4 GHz Quad Core
- Time limit: **15 minutes**
- Memory limit: **15 GB**

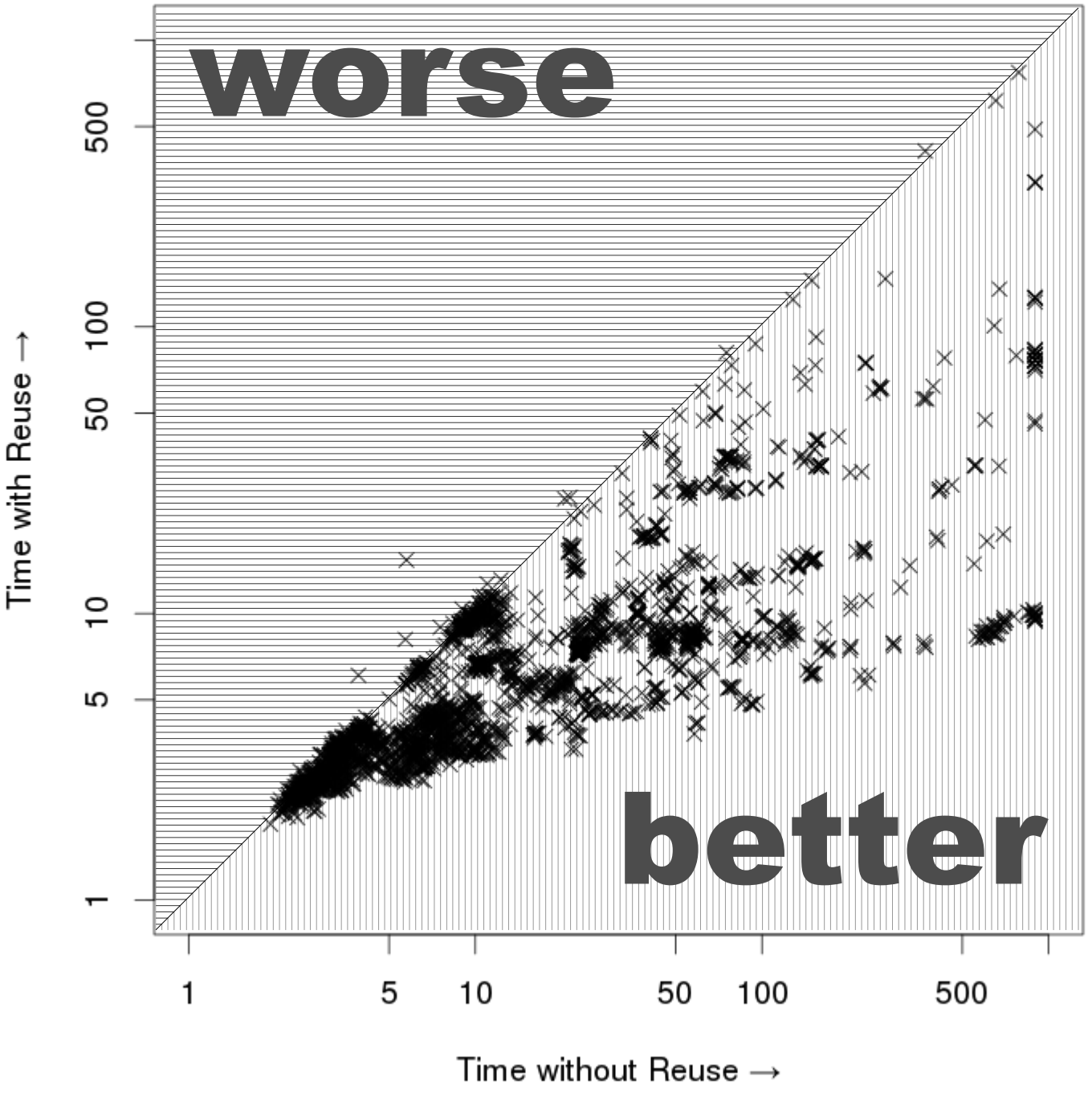
= Setup of the Intl. Competition on Software Verification


CPU time in seconds
worse
With Reuse
Without Reuse
better
Time with Reuse →
1
5
10
50
100
500
Time without Reuse →

Results for Predicate Analysis



Results for Predicate Analysis



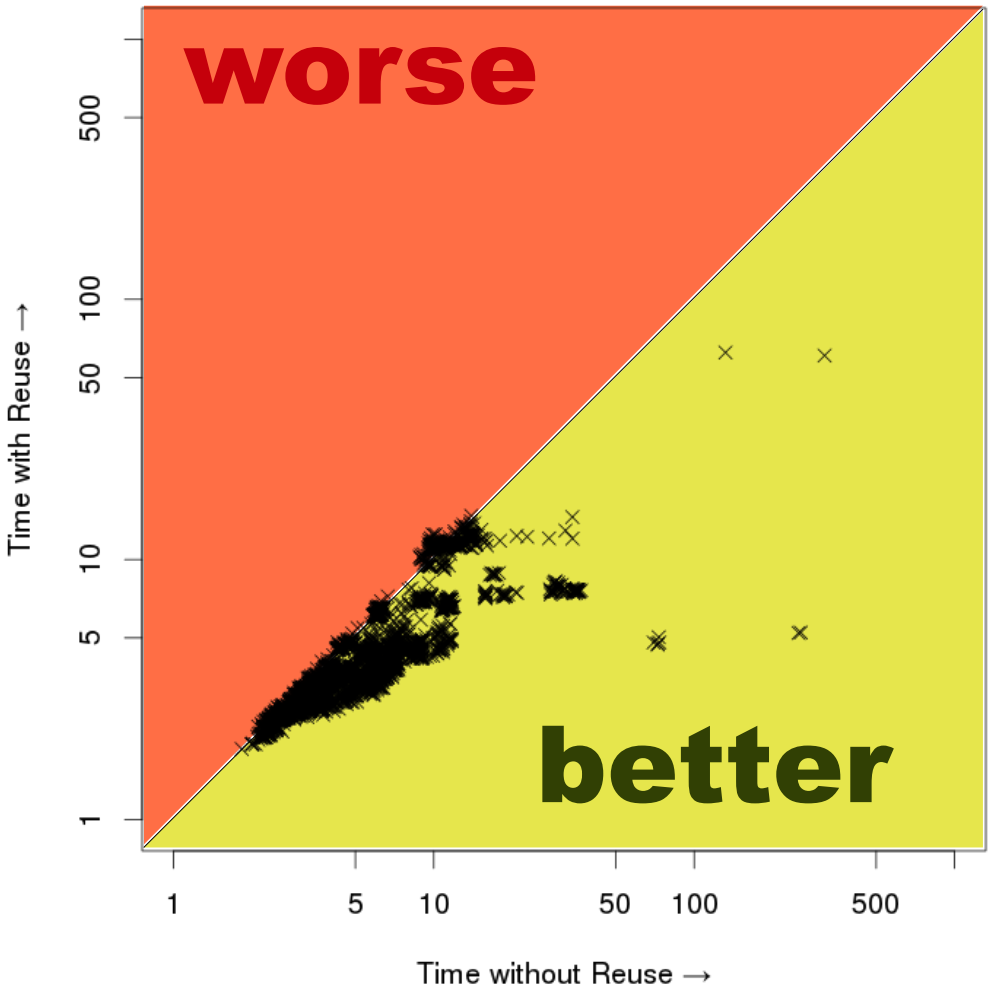
# Tasks	4 193
CPU Time without Reuse	130 000
CPU Time with Reuse	40 000
Speedup	**3.7**
Solved	4 001 **+ 56**

Sensitivity to Changes

Analysis	Revs.	# Tasks	Average Difference (Lines)	CPU Time without Reuse	CPU Time with Reuse	Speedup	Solved
Predicate	All	4 193	688	130 000	40 000	**3.7**	4 001 **+56**
	4th	1 090	1 579	34 000	14 000	**3.2**	1 045 **+12**

→ **Low sensitivity** to changes in the program code

Results for Explicit-State Analysis



Revs.	# Tasks	Different Lines (Average)	CPU Time without Reuse	CPU Time with Reuse	Speedup	Solved
All	4 193	688	27 000	20 000	**1.4**	4 191
4th	1 090	1 579	6 300	5 100	**1.3**	1 090

Conclusion

Precision reuse has a significant positive effect!

- Drastically **improves performance**

  Drastically reduces the number of refinements

- **More** problems can be **solved**
- **Low sensitivity** to changes in the program code