Explicit-State Software Model Checking Based on CEGAR and Interpolation

Dirk Beyer and Stefan Löwe

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

ETAPS Test-of-Time Award 2023

Proc. FASE 2013, doi:10.1007/978-3-642-37057-1_11

Software Verification

```
int a, b, c;
a := 0;
b := a;
c := a;
if(a == 0) {
   a := 1;
}
if(a == -1) {
  assert(0);
}
```

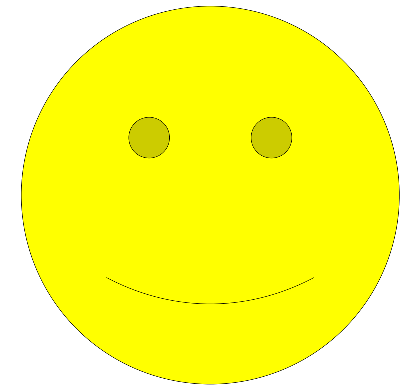
The goal is to find an answer to the question:

Does the specification hold?

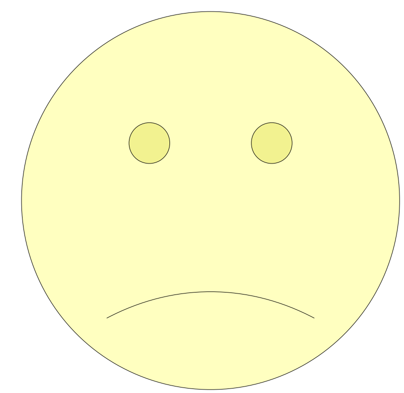
Software Model Checking

```
int a, b, c;
a := 0;
b := a;
c := a;
if(a == 0) {
    a := 1;
}
if(a == -1) {
   assert(0);
}
```

TRUE ?



FALSE ?



Does the specification hold?

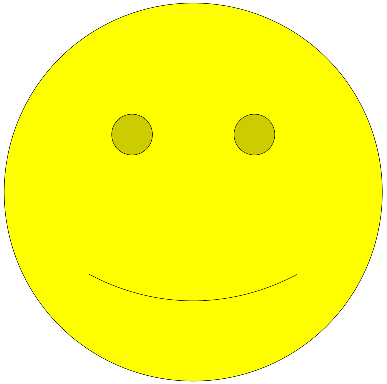
State of the Art in 2013

- Explicit-state model checking (SPIN, ...)
- Symbolic-state model checking (SLAM, BLAST, SATABS, …)
- Data-flow analysis (Astree, ...)

- Space between these extremes were largely unexplored

- Contribution: Explore this!
  - Explicit-value domain with abstraction
  - CEGAR for both combined explicit+predicate

Explicit-State Software Model Checking



int a, b, c;

a := 0;

b := a;

c := a;

if(a == 0) {
    a := 1;
}

if(a == -1) {
 ~~assert(0);~~
}

{a → T, b → T, c → T}

{**a → 0**, b → T, c → T}

{a → 0, **b → 0**, c → T}

{a → 0, b → 0, **c → 0**}

{**a → 1**, b → 0, c → 0}

{**a → 1**, b → 0, c → 0}

SAFE !

Does the specification hold?

Status Before

- Very efficient successor computation
- Independent of expensive solver techniques

- Imprecise when joining

- State-space explosion
  especially when not joining

Explicit-State Software Model Checking

Existing approach: simple value assignments

- ? Abstraction
- ? Counterexample-Guided Abstraction Refinement
- ? Interpolation

All known in the predicate domain for years

Explicit-State Software Model Checking

New approach: **integrate CEGAR and Interpolation**

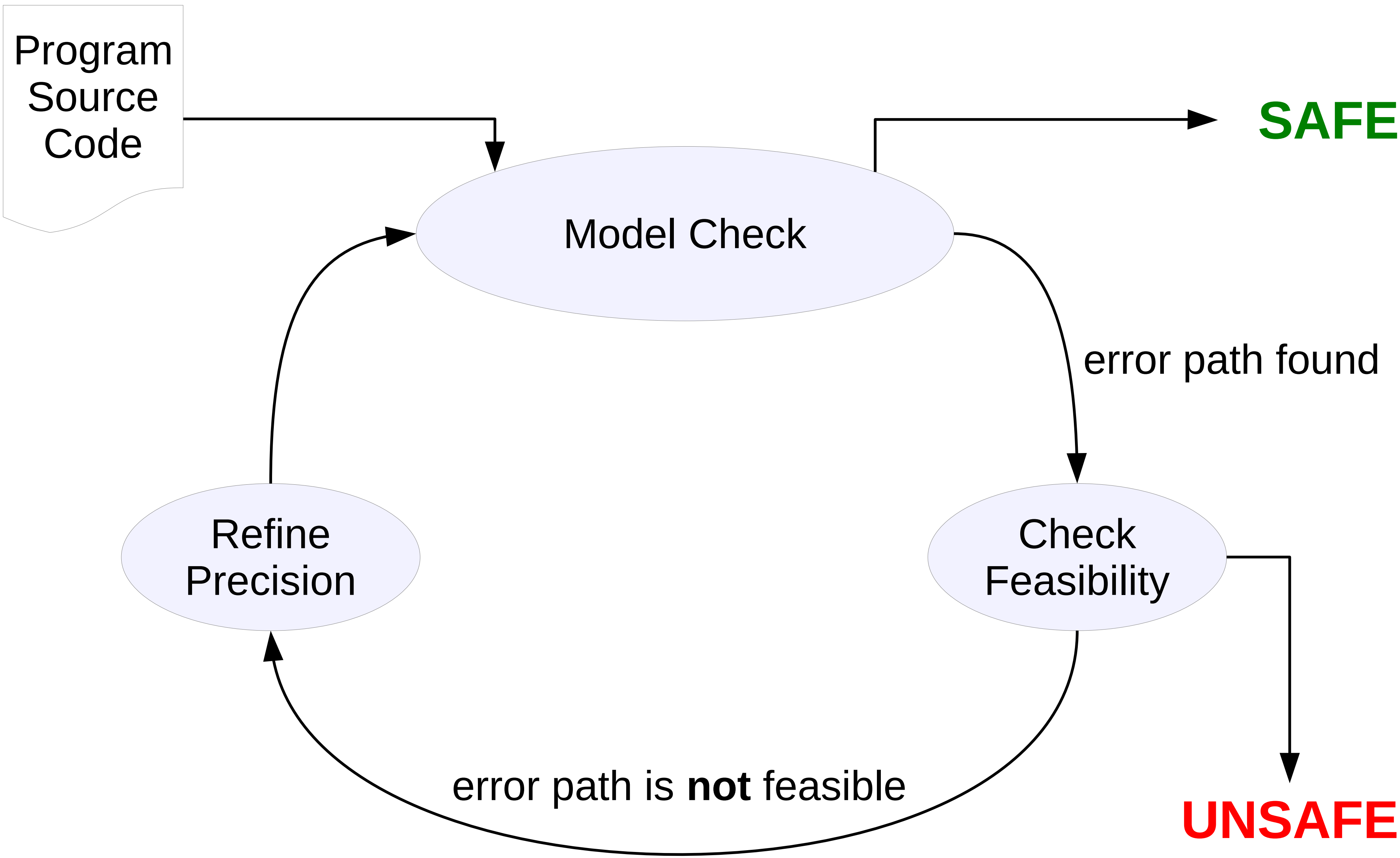
- Abstraction
- Counterexample-Guided Abstraction Refinement
- Interpolation

✔ *Explicit-State Software Model Checking*

based on ***CEGAR***

and ***Interpolation***

CEGAR Loop



Program Source Code
SAFE
Model Check
error path found
Refine Precision
Check Feasibility
error path is **not** feasible
UNSAFE

Abstraction

```
int a, b, c;
a := 0;
b := a;
c := a;
if(a == 0) {
  a := 1;
}
if(a == -1) {
  assert(0);
}
```

{ }

{ }

{ }

{ }

{ }

{ }

if the abstraction is too coarse,
spurious counterexamples will be reported

Counterexamples

counterexample as

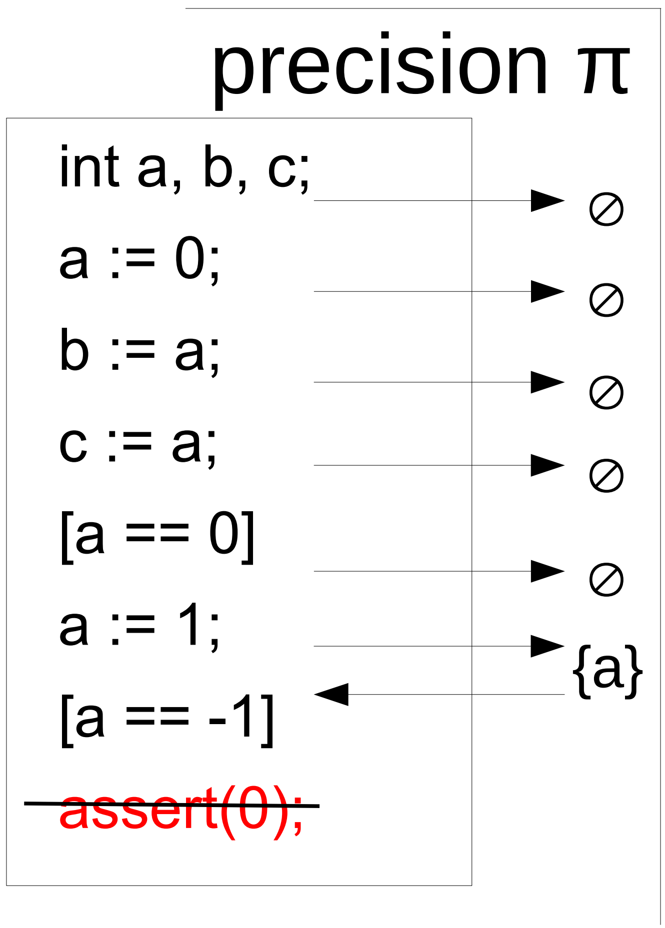
constraint sequence

```
int a, b, c;
a := 0;
b := a;
c := a;
[a == 0]
a := 1;
[a == -1]
assert(0);
```

We extract **variable identifiers** from spurious counterexamples in order to avoid repeated explorations of the same spurious counterexamples

Therefore, we introduce the notion of a precision

Precision



a set of variable identifiers to track at a program location

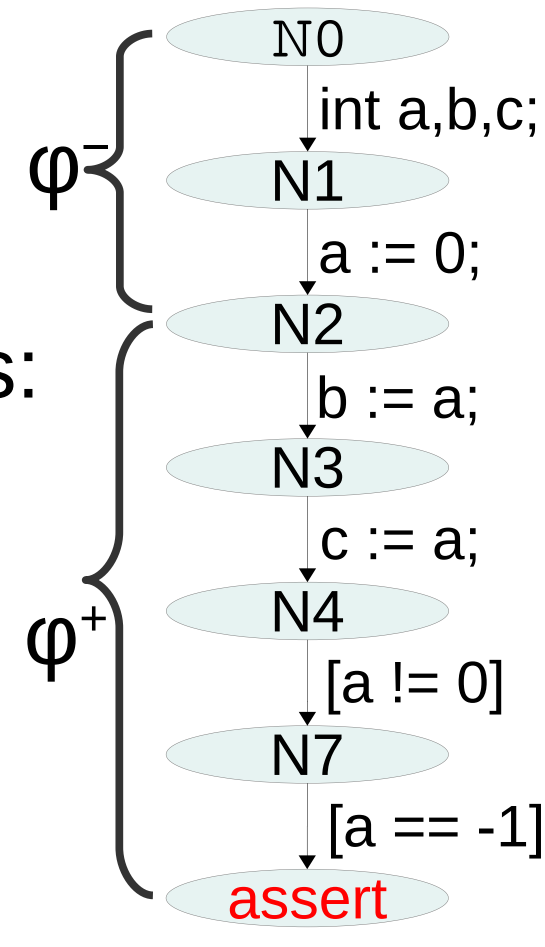
- be precise enough to avoid spurious counterexamples
- be abstract enough to allow an efficient analysis

How to obtain such a parsimonious precisions?

Craig Interpolation

For a pair of ***formulas*** $\boldsymbol{\varphi^-}$ and $\boldsymbol{\varphi^+}$,
such that $\boldsymbol{\varphi^-} \wedge \boldsymbol{\varphi^+}$ is ***unsatisfiable***,
a Craig ***interpolant*** $\boldsymbol{\psi}$ is a ***formula***
that fulfills the following requirements:

1) φ^- implies ψ
2) $\psi \wedge \varphi^+$ is unsatisfiable
3) ψ only contains symbols that
   are common to both φ^- and φ^+



[Abstractions from Proofs, 2004, Henzinger, Jhala, Majumdar, McMillan]

Craig Interpolation

For a pair of ***formulas*** $\boldsymbol{\varphi^-}$ and $\boldsymbol{\varphi^+}$,
such that $\boldsymbol{\varphi^- \wedge \varphi^+}$ is ***unsatisfiable***,
a Craig ***interpolant*** $\boldsymbol{\psi}$ is a ***formula***
that fulfills the following requirements:

1) φ^- implies ψ
2) $\psi \wedge \varphi^+$ is unsatisfiable
3) ψ only contains symbols that are common to both φ^- and φ^+

→ ***apply this to the Explicit-Value Domain***

Our Main Contribution

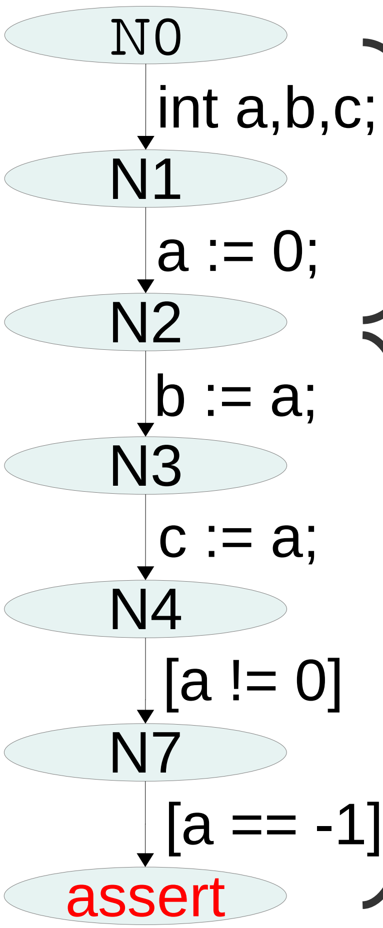
→ apply interpolation to constraint sequences

For a pair of ***constraint sequences*** $\boldsymbol{\gamma}^{-}$ and $\boldsymbol{\gamma}^{+}$,
such that $\boldsymbol{\gamma}^{-} \wedge \boldsymbol{\gamma}^{+}$ is ***contradicting***,
an ***interpolant*** $\boldsymbol{\psi}$ is a ***constraint sequence***
that fulfills the following requirements:

1) γ^{-} implies ψ
2) $\psi \wedge \gamma^{+}$ is contradicting
3) ψ only contains symbols that
are common to both γ^{-} and γ^{+}

→ ***Explicit-Value Interpolation***

Explicit-Value Interpolation



- ✔ path is infeasible, i.e., $\gamma^- \wedge \gamma^+$ is contradicting

γ^- : a := 0;

γ^+ : b := a;
c := a;
[a != 0]
[a == -1]

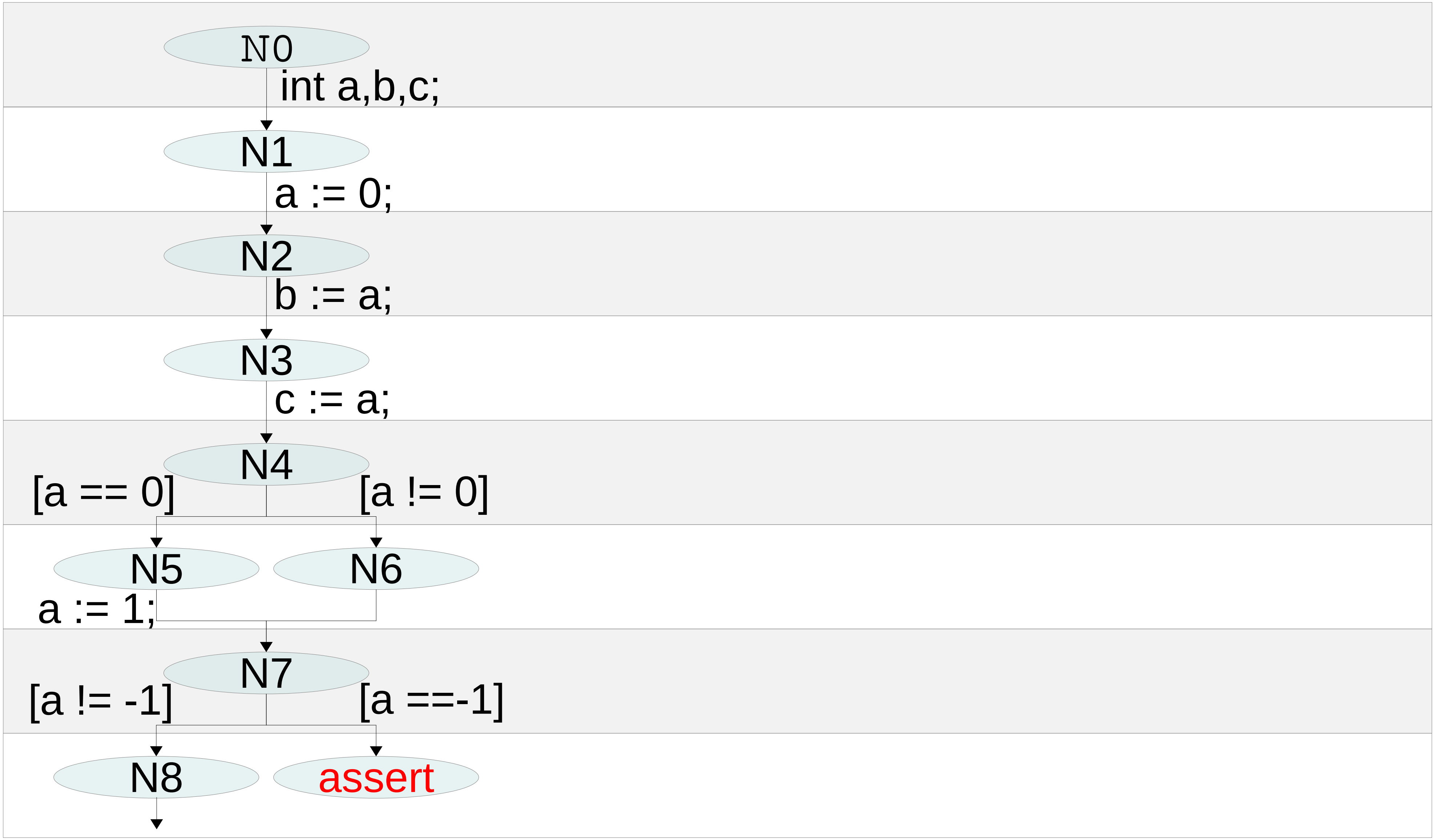
ψ : {a := 0;}

- ✔ γ^- implies ψ
- ✔ $\psi \wedge \gamma^+$ is contradicting
- ✔ common symbols

➢ ***Add “a” to the precision of location N2***

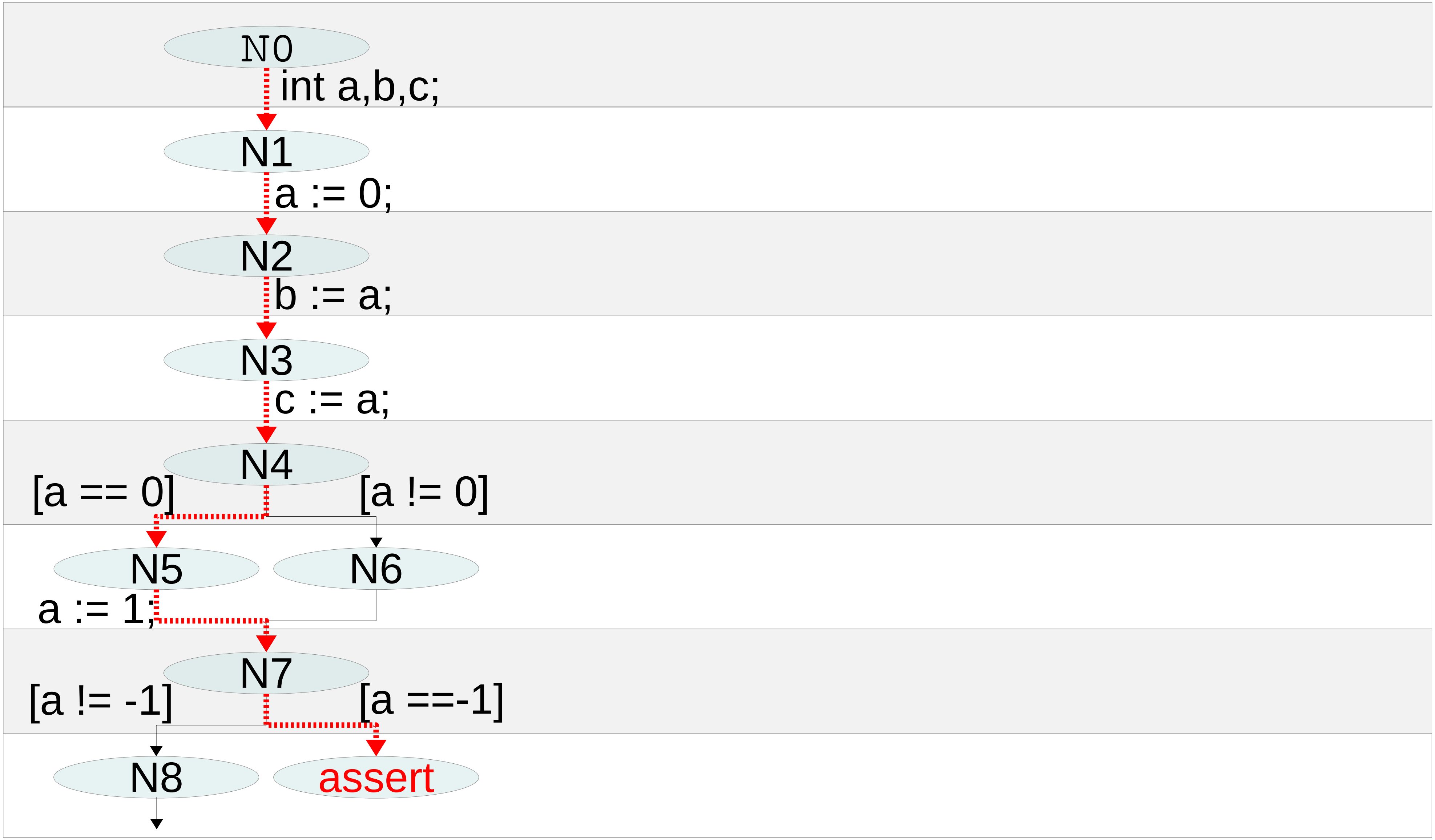
Interpolation-Based Refinement

Control-Flow Automaton

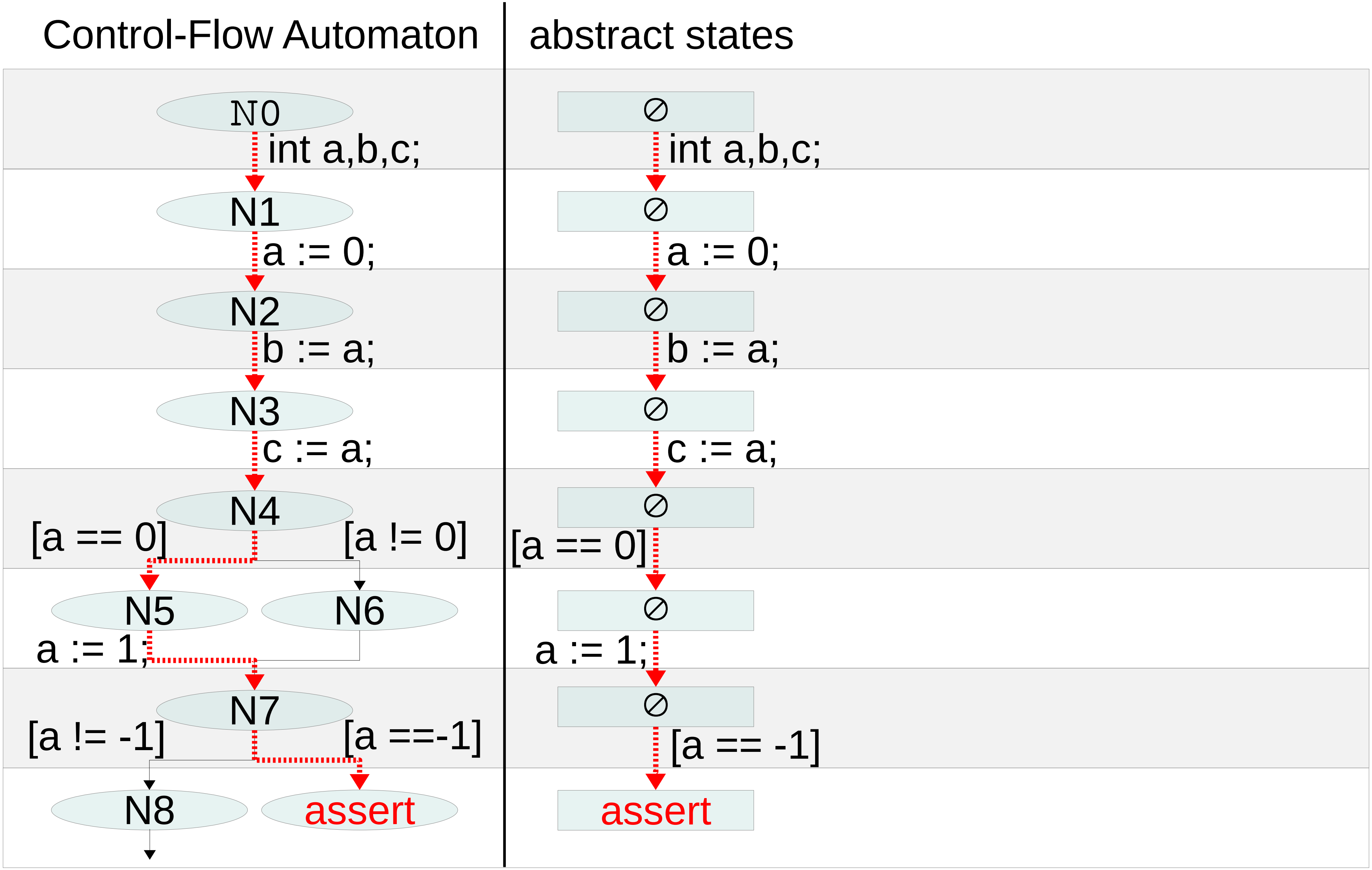


Interpolation-Based Refinement

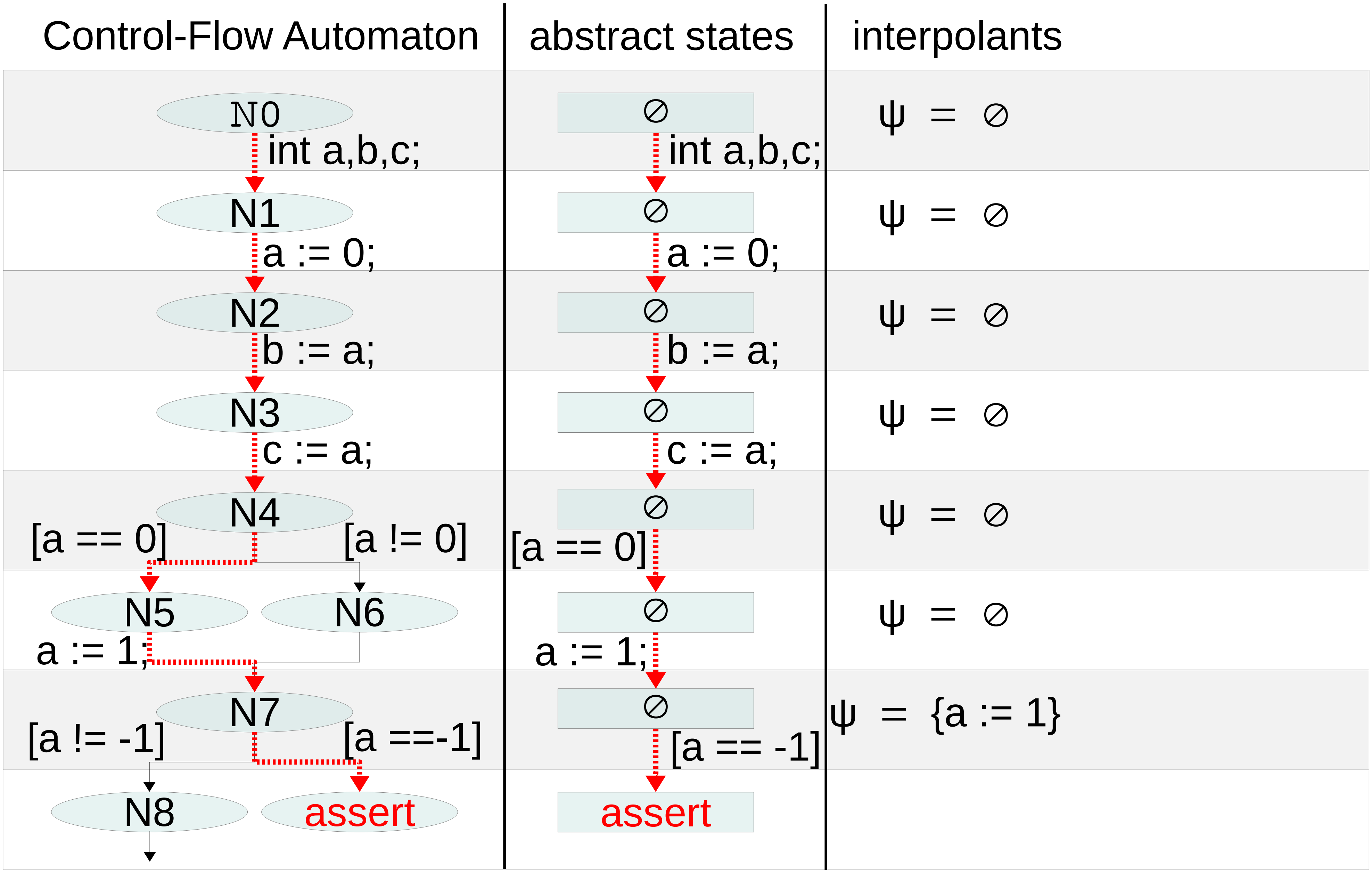
Control-Flow Automaton



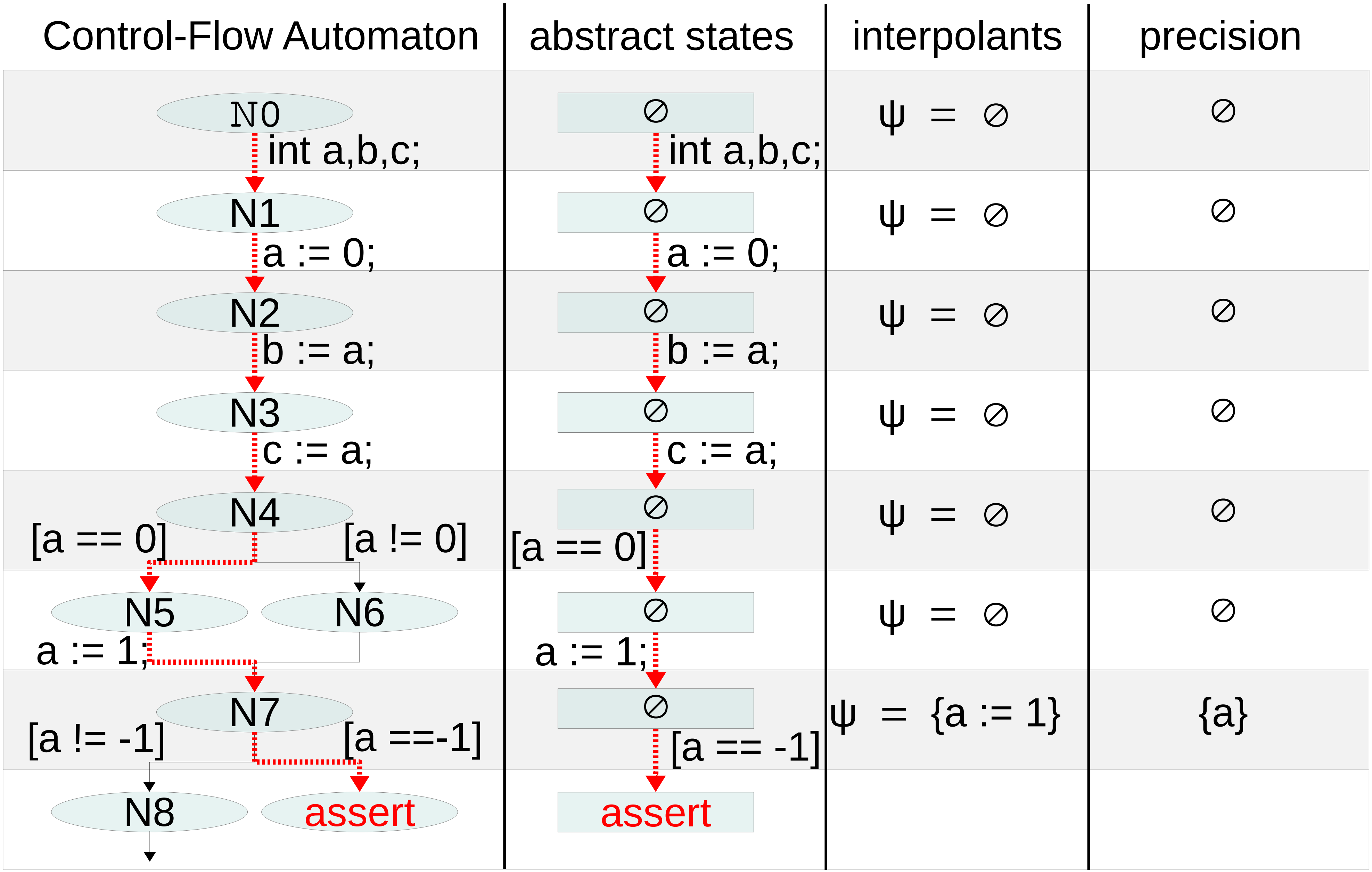
Interpolation-Based Refinement



Interpolation-Based Refinement



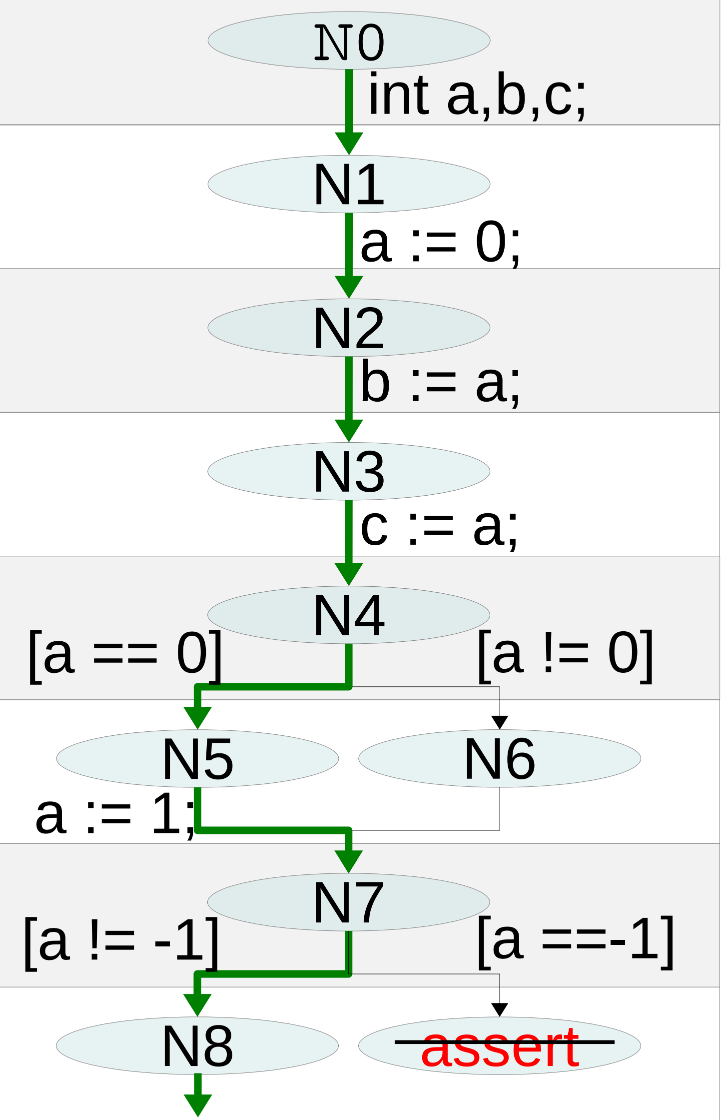
Interpolation-Based Refinement



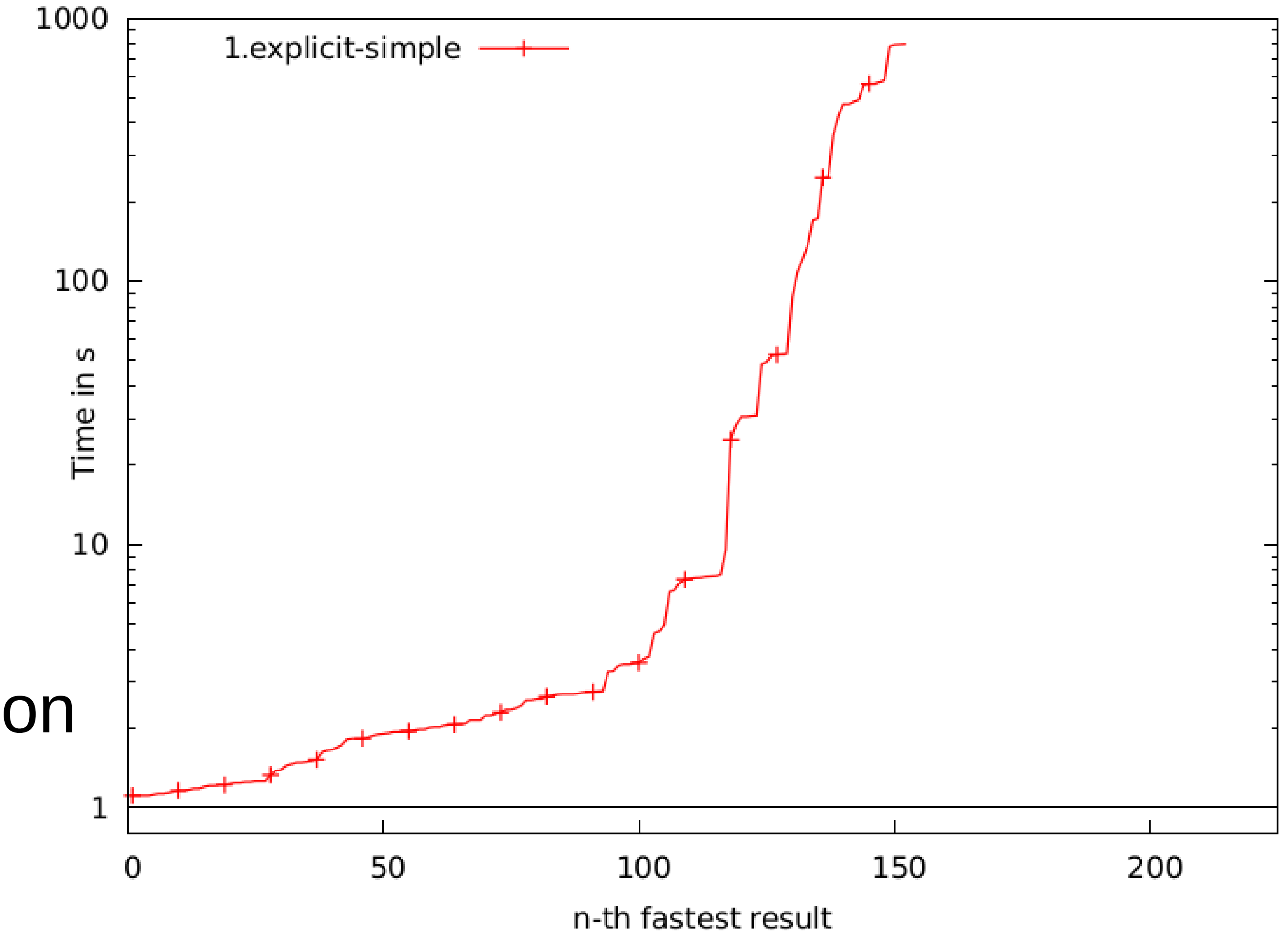
Interpolation-Based Refinement

abstract states	interpolants	precision
$\emptyset$ — int a,b,c;	$\psi = \emptyset$	$\emptyset$
$\emptyset$ — a := 0;	$\psi = \emptyset$	$\emptyset$
$\emptyset$ — b := a;	$\psi = \emptyset$	$\emptyset$
$\emptyset$ — c := a;	$\psi = \emptyset$	$\emptyset$
$\emptyset$ — [a == 0]	$\psi = \emptyset$	$\emptyset$
$\emptyset$ — a := 1;	$\psi = \emptyset$	$\emptyset$
$\emptyset$ — [a == -1]	$\psi = \{a := 1\}$	$\{a\}$
assert		

error path refuted

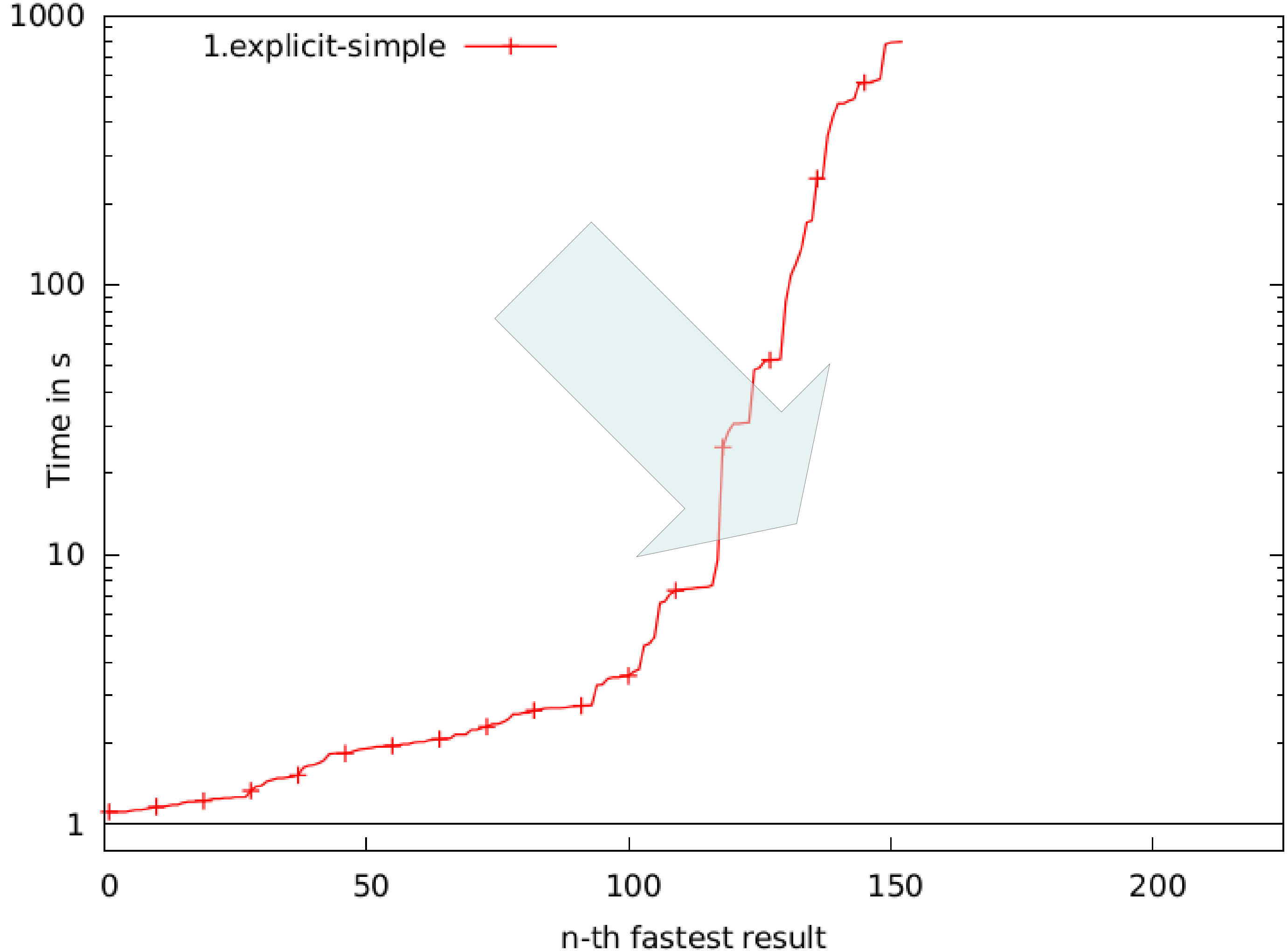


Experimental Evaluation

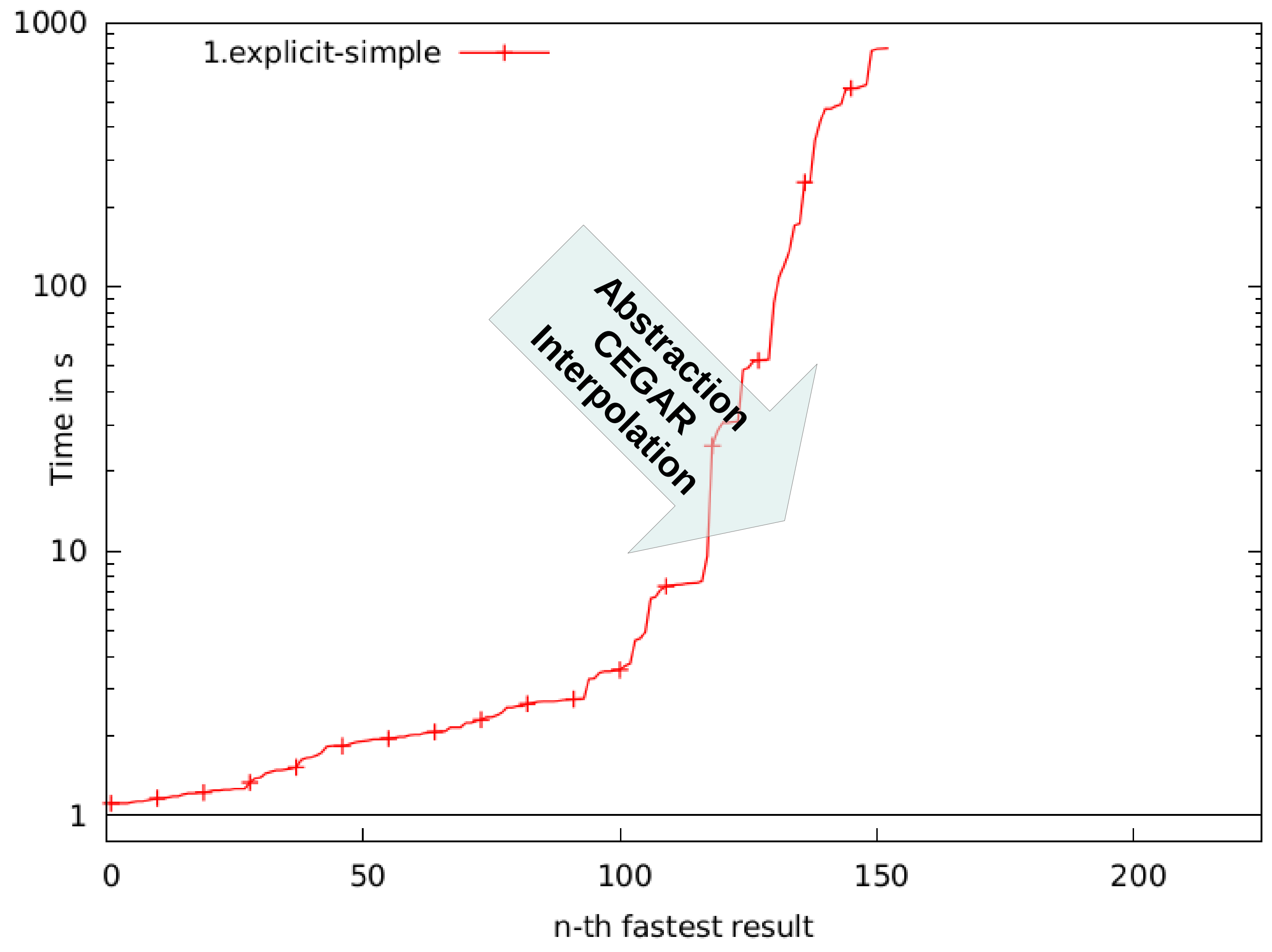


Benchmark from
1st International
Competition on
Software Verification
(SV-Comp'12)

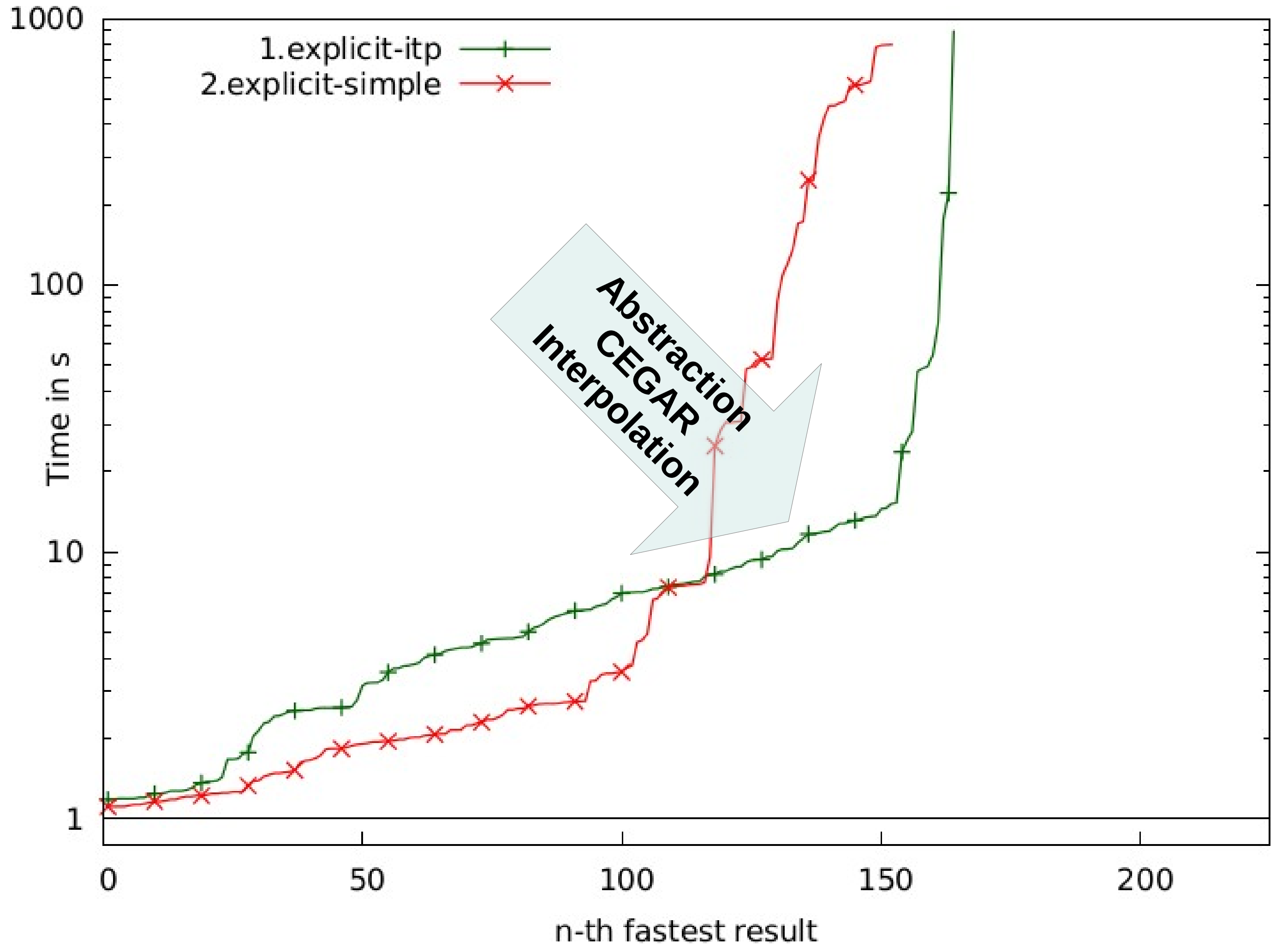
Experimental Evaluation



Experimental Evaluation



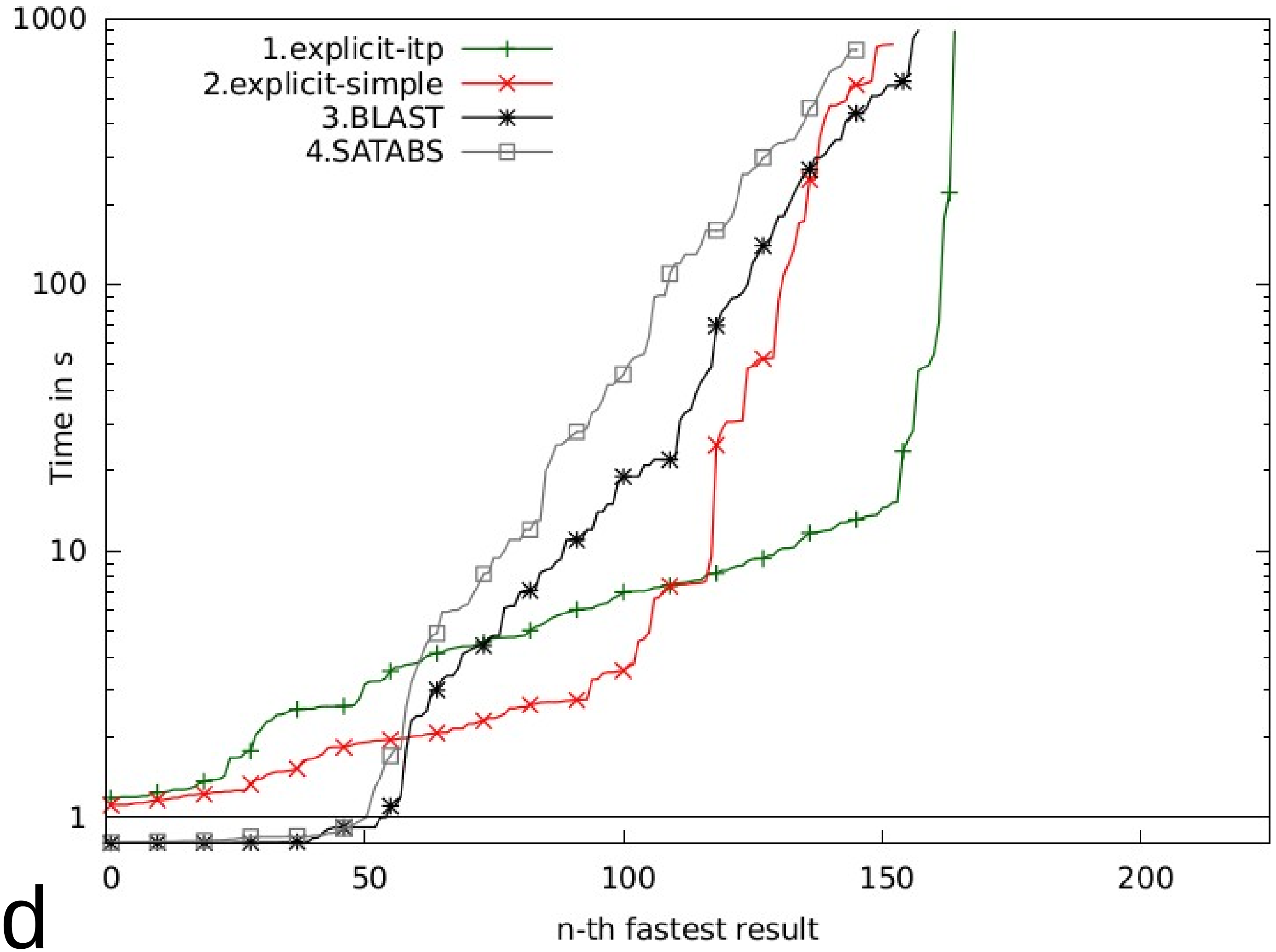
Performance Improvement



- ✔ Faster
- ✔ Better

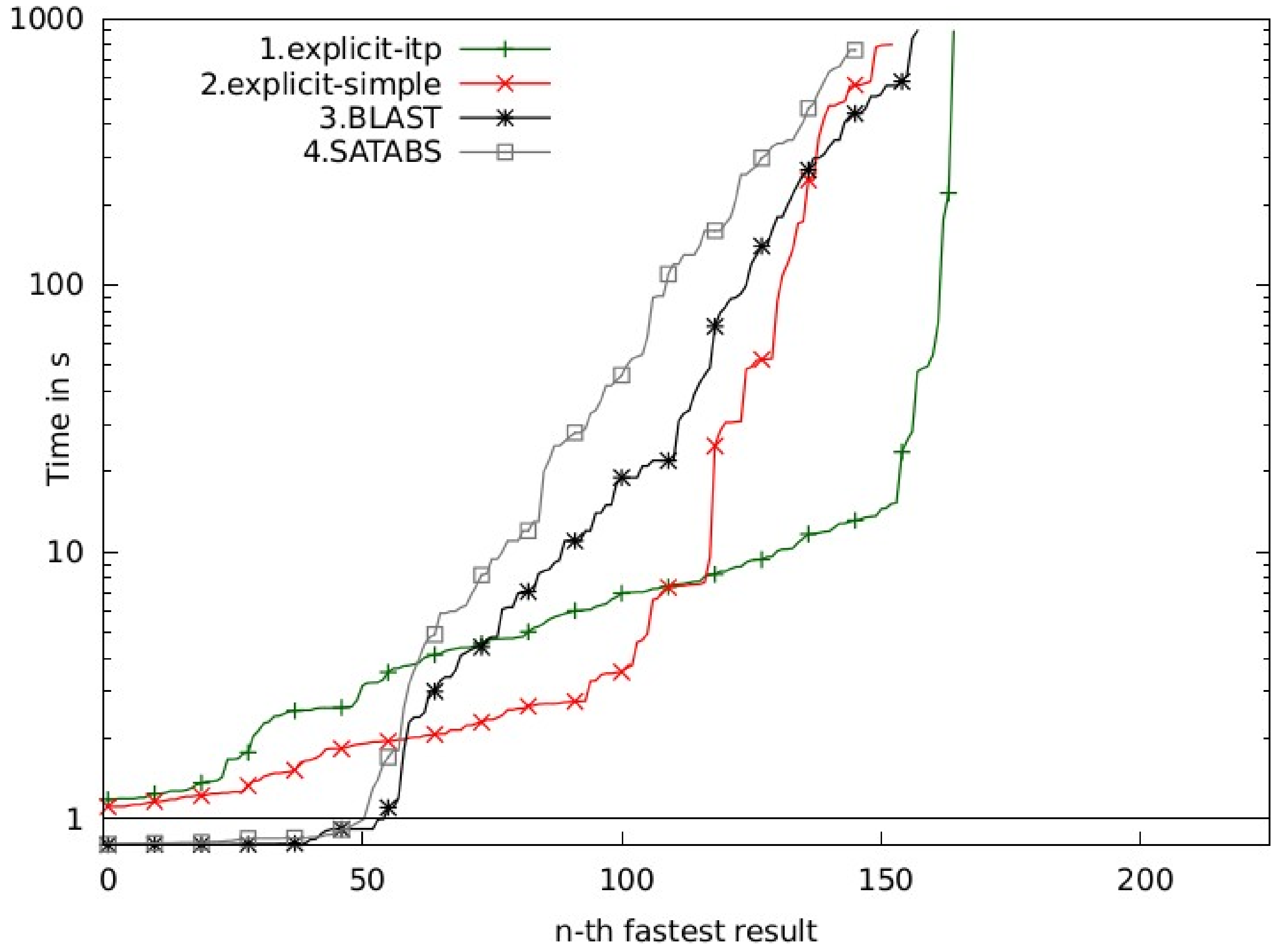
✔ Abstraction ✔ CEGAR ✔ Interpolation

Comparison with Well-Established Tools



Out-performs well-established predicate-based tools like BLAST or SATABS

Comparison with Well-Established Tools



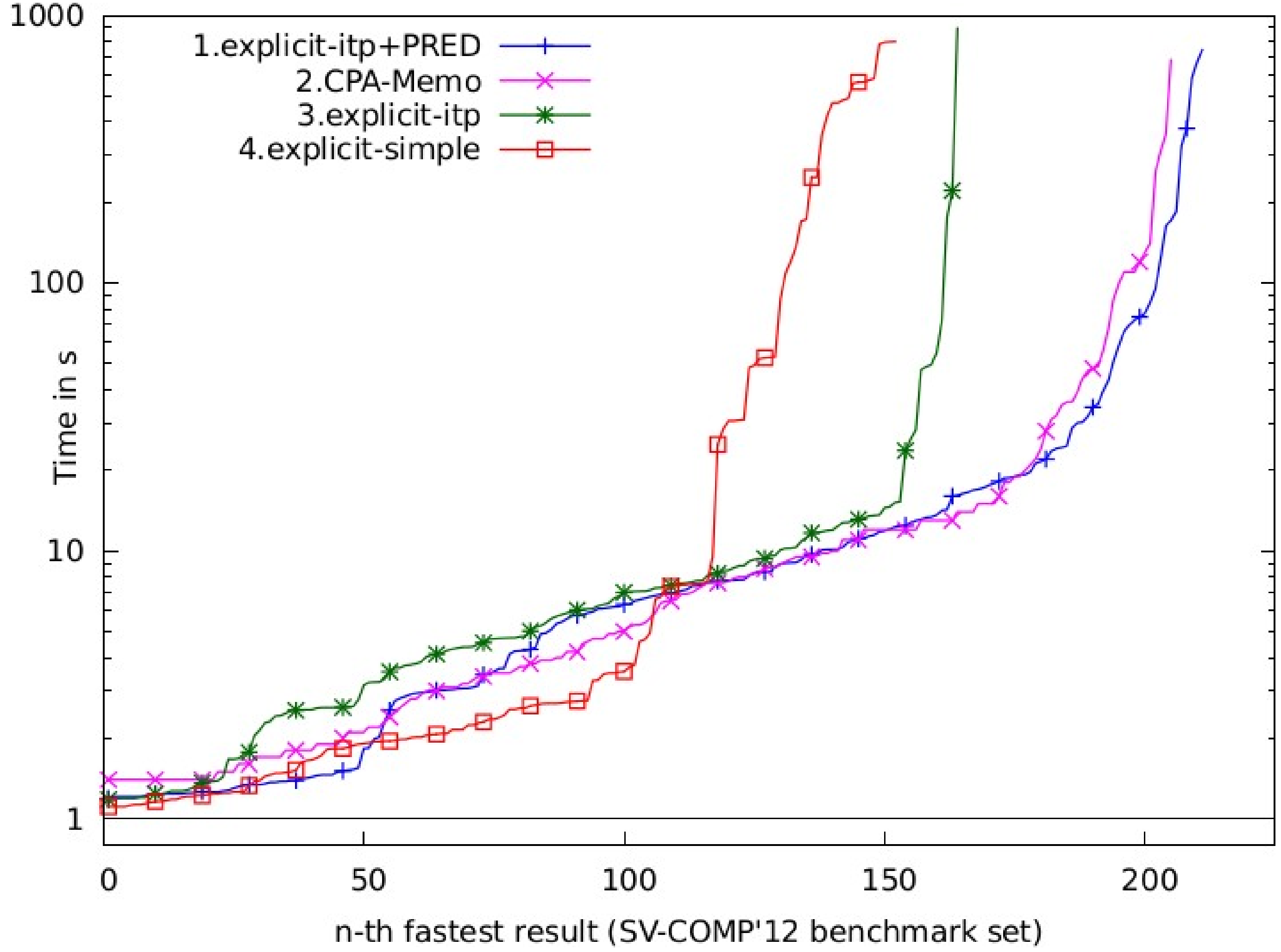
Can we further improve on this?

Have best of both worlds

Add auxiliary predicate analysis:

- Refinement of both domains based on their expressiveness
- Explicit analysis tracks most information efficiently
- Predicate analysis tracks only what is beyond that

Combined with Predicate Analysis



Out-performs SV-COMP '12 Winner CPA-Memo

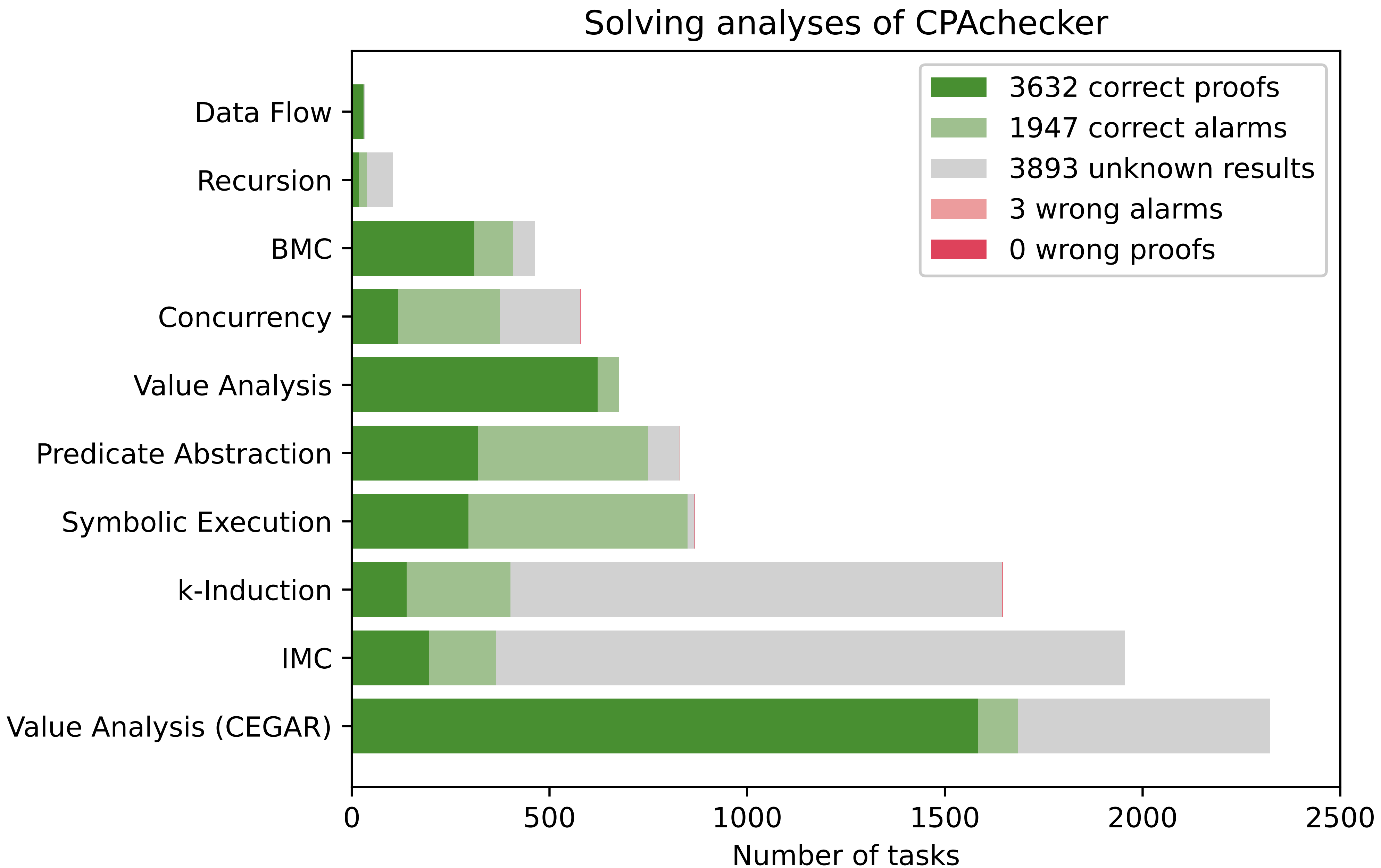
Results of SV-COMP '13

Our tool implementation
CPAchecker-Explicit 1.1.10
participated in SV-COMP '13, and won ...

Silver Medal in category **ControlFlowInteger**
Silver Medal in category **DeviceDrivers64**
Silver Medal in category **SystemC**

Silver Medal in category **Overall**

Usage in CPAchecker 2023



Conclusion

CPAchecker

http://cpachecker.sosy-lab.org

- Defined and implemented
  - Abstraction
  - CEGAR
  - Interpolation

  for the explicit-value domain
- Combination with predicate abstraction
- Compelling results
  - Effective method to reduce reached set
  - Avoid state-space explosion