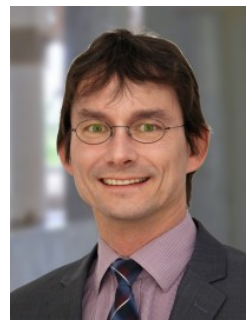
Interpolation and SAT-Based Model Checking Revisited: Adoption to Software Verification

Philipp Wendler

Joint work with Dirk Beyer and Nian-Ze Lee

LMU Munich, Germany



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CPA

SoSy-Lab Software Systems

Leveraging Knowledge from Related Fields

- Software verification is hard
- Successfully adopted algorithms from hardware
  - BMC [9, 12]
  - k-Induction [17, 4]
  - IC3/PDR [11, 3]

Interpolation and SAT-Based Model Checking

- K. L. McMillan, CAV 2003 [18]
- Interpolation-based model checking (IMC)
  - Finite-state transition systems (circuit)
  - Fixed points with interpolants from unsatisfiable BMC

Interpolation and SAT-Based Model Checking

- K. L. McMillan, CAV 2003 [18]
- Interpolation-based model checking (IMC)
  - Finite-state transition systems (circuit)
  - Fixed points with interpolants from unsatisfiable BMC
- Inspired many interpolation-based approaches
- State of the art for hardware

Interpolation and SAT-Based Model Checking

- K. L. McMillan, CAV 2003 [18]
- Interpolation-based model checking (IMC)
  - Finite-state transition systems (circuit)
  - Fixed points with interpolants from unsatisfiable BMC
- Inspired many interpolation-based approaches
- State of the art for hardware
- IMC itself has not been used to verify programs

Research Questions

- Efficient adoption of IMC to verify programs
- Performance of IMC against state of the art

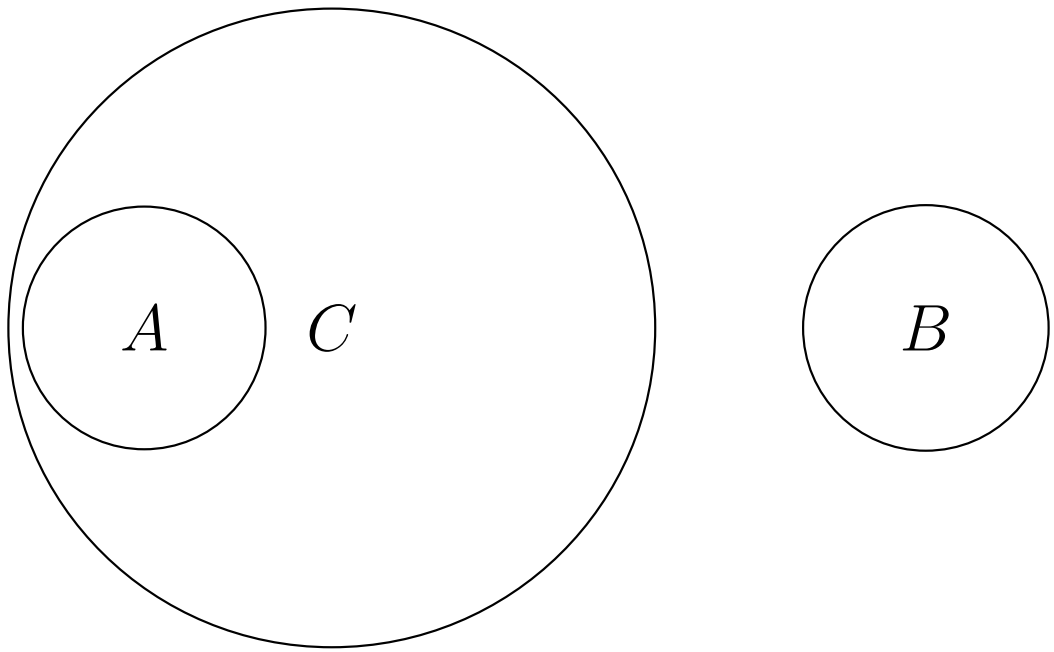
Research Questions

- Efficient adoption of IMC to verify programs
- Performance of IMC against state of the art

Spoiler:
Competitive among polished software-verification methods

Craig Interpolation

- $A(X,Y) \wedge B(Y,Z)$ UNSAT: interpolant $C(Y)$
  - $A(X,Y) \rightarrow C(Y)$
  - $C(Y) \wedge B(Y,Z)$ UNSAT



Interpolation-Based Model Checking

- State-transition system: $I(s), T(s, s'), P(s)$

Interpolation-Based Model Checking

- State-transition system: $I(s), T(s, s'), P(s)$
- $I(s_0)T(s_0, s_1)\ T(s_1, s_2) \ldots T(s_{k-1}, s_k)\neg P(s_k)$

Interpolation-Based Model Checking

- State-transition system: $I(s), T(s, s'), P(s)$
- $\underbrace{I(s_0)T(s_0, s_1)}_{A(s_0,s_1)} \underbrace{T(s_1, s_2) \ldots T(s_{k-1}, s_k)\neg P(s_k)}_{B(s_1,s_2,\ldots,s_k)}$
- Interpolant $C_1(s_1)$: 1-step overapproximation

Interpolation-Based Model Checking

- State-transition system: $I(s), T(s, s'), P(s)$
- $\underbrace{I(s_0)T(s_0, s_1)}_{A(s_0,s_1)} \underbrace{T(s_1, s_2) \dots T(s_{k-1}, s_k)\neg P(s_k)}_{B(s_1,s_2,\dots,s_k)}$
- Interpolant $C_1(s_1)$: 1-step overapproximation
- $\underbrace{C_1(s_0)T(s_0, s_1)}_{A'(s_0,s_1)} \underbrace{T(s_1, s_2) \dots T(s_{k-1}, s_k)\neg P(s_k)}_{B'(s_1,s_2,\dots,s_k)}$
  - Interpolant $C_2(s_1)$: 2-step overapproximation
  - Repeat until $\bigvee C_i$ becomes a fixed point
  - Increment k if query becomes satisfiable

Towards an Efficient Adoption

- System under verification $\rightarrow I(s), T(s, s'), P(s)$
  - Sequential circuit: monolithic loop
  - Program: arbitrary control flow

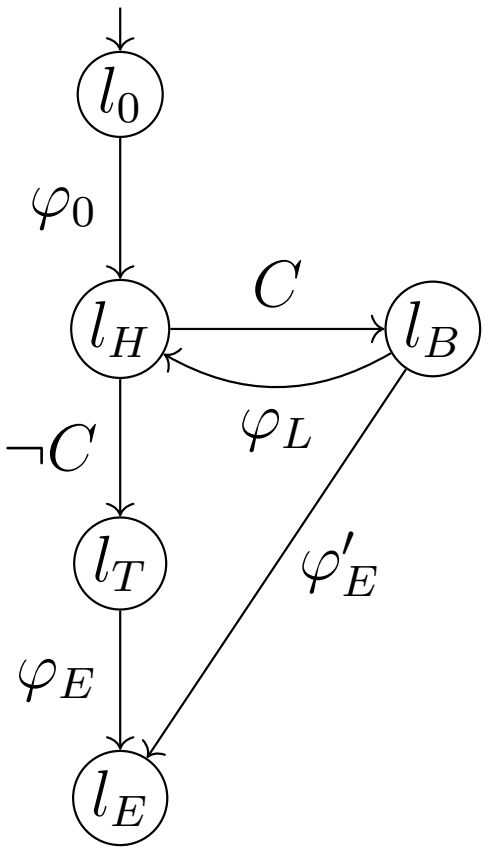
Towards an Efficient Adoption

- System under verification $\rightarrow I(s), T(s, s'), P(s)$
  - Sequential circuit: monolithic loop
  - Program: arbitrary control flow
- Challenge: minimal disruption of program structure

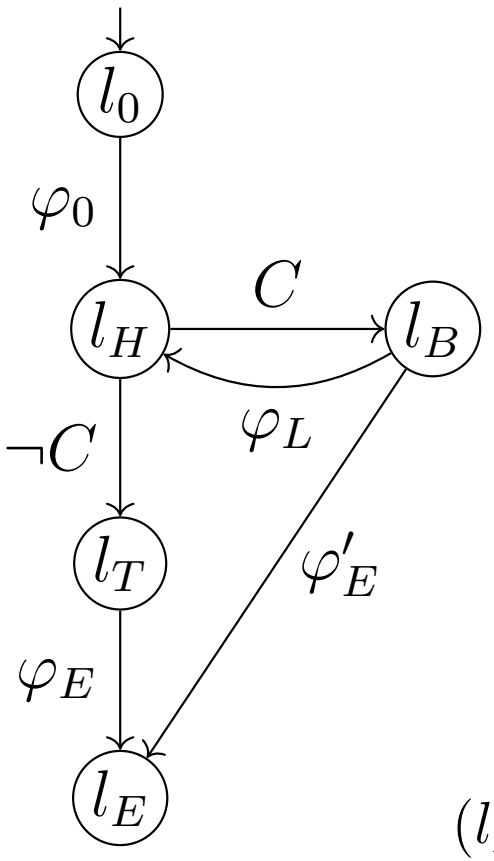
Towards an Efficient Adoption

- System under verification $\rightarrow I(s), T(s, s'), P(s)$
  - Sequential circuit: monolithic loop
  - Program: arbitrary control flow
- Challenge: minimal disruption of program structure
- Idea: Use large-block encoding (LBE) [2, 7] to summarize control-flow automaton (CFA)
  - Loop-free blocks replaced by single transitions

Single-Loop CFA summarized by LBE

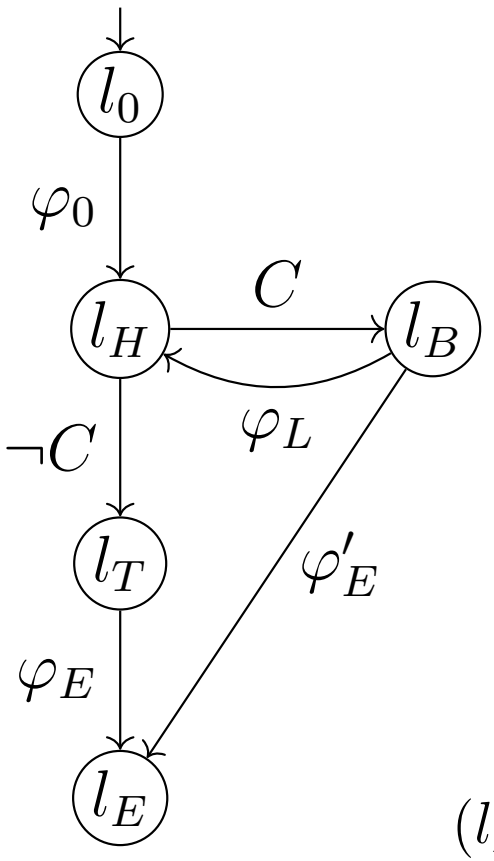


Single-Loop CFA summarized by LBE



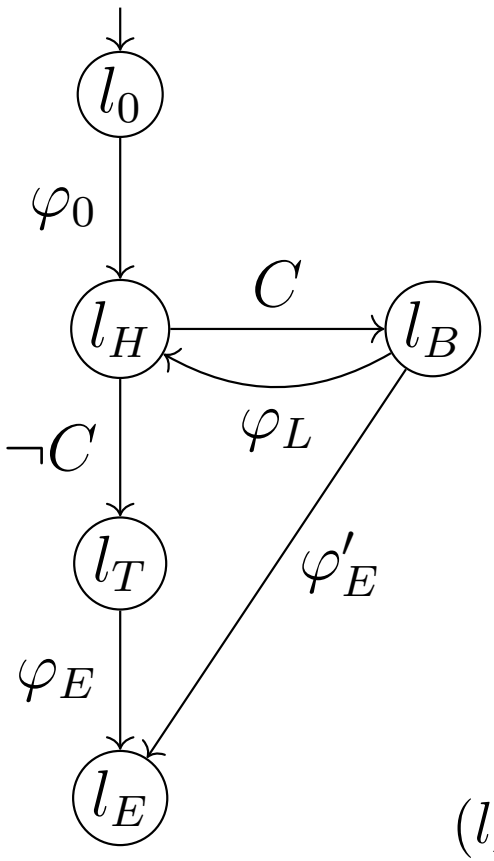
$$(l_0, true)$$
$$\downarrow$$
$$(l_H, \varphi_0|_{i=0})$$
$$\downarrow$$
$$(l_H, (C \wedge \varphi_L)|_{i=1})$$
$$\downarrow$$
$$(l_H, (C \wedge \varphi_L)|_{i=2})$$
$$\vdots$$
$$(l_H, (C \wedge \varphi_L)|_{i=k})$$
$$\downarrow$$
$$(l_E, ((\neg C \wedge \varphi_E) \vee (C \wedge \varphi'_E))|_{i=k})$$

Single-Loop CFA summarized by LBE



$$
\begin{aligned}
&(l_0, \mathit{true}) \\
&\downarrow \\
&(l_H, \varphi_0|_{i=0}) && : I(s_0) \\
&\downarrow \\
&(l_H, (C \wedge \varphi_L)|_{i=1}) && : T(s_0, s_1) \\
&\downarrow \\
&(l_H, (C \wedge \varphi_L)|_{i=2}) && : T(s_1, s_2) \\
&\vdots \\
&(l_H, (C \wedge \varphi_L)|_{i=k}) && : T(s_{k-1}, s_k) \\
&\downarrow \\
&(l_E, ((\neg C \wedge \varphi_E) \vee (C \wedge \varphi'_E))|_{i=k}) && : \neg P(s_k)
\end{aligned}
$$

Single-Loop CFA summarized by LBE



$$
\begin{aligned}
&(l_0, \mathit{true}) \\
&\downarrow \\
&(l_H, \varphi_0|_{i=0}) && : I(s_0) \\
&\downarrow \\
&(l_H, (C \wedge \varphi_L)|_{i=1}) && : T(s_0, s_1) \\
&\downarrow \\
&(l_H, (C \wedge \varphi_L)|_{i=2}) && : T(s_1, s_2) \\
&\vdots \\
&(l_H, (C \wedge \varphi_L)|_{i=k}) && : T(s_{k-1}, s_k) \\
&\downarrow \\
&(l_E, ((\neg C \wedge \varphi_E) \vee (C \wedge \varphi'_E))|_{i=k}) && : \neg P(s_k)
\end{aligned}
$$

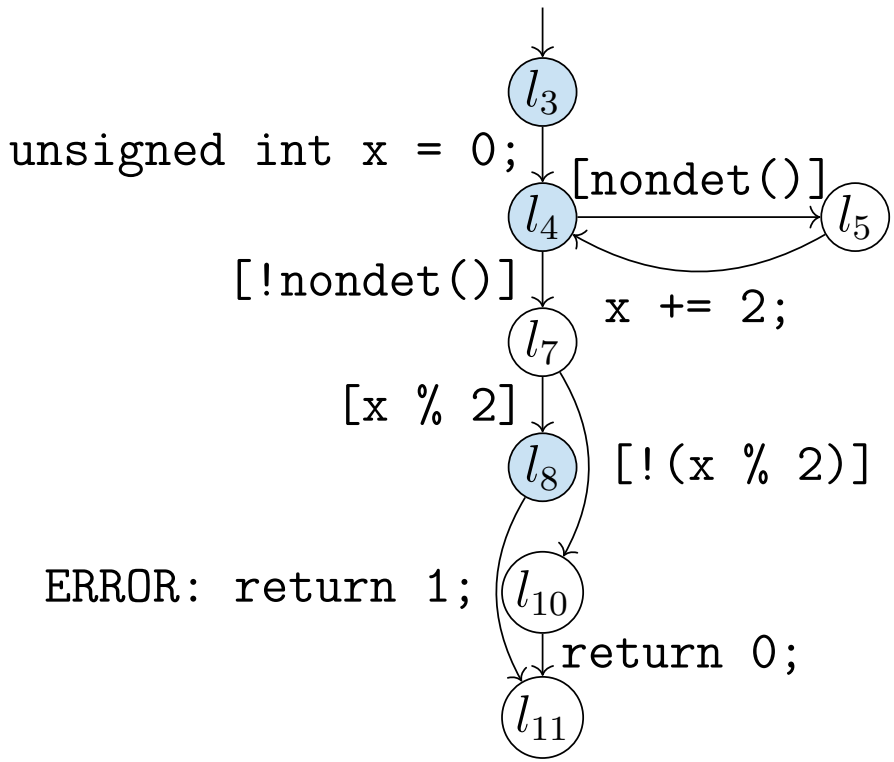
Solution for multi-loop programs:
standard transformation to single loop

Interpolation-based model checking for Software

- Expressed on top of existing unifying framework for SMT-based software verification [5]
  - Provides LBE, CFA unrolling, formula collection
  - Easy comparison with other approaches (k-induction, IMPACT, ...)
  - Details in paper [8]
    `https://www.sosy-lab.org/research/cpa-imc/`
- Implemented in CPACHECKER [6]

Reachability Analysis of Errors

```
1  extern int nondet();
2  int main(void) {
3    unsigned int x = 0;
4    while (nondet()) {
5      x += 2;
6    }
7    if (x % 2) {
8      ERROR: return 1;
9    }
10   return 0;
11 }
```



Example for One Loop Unrolling

$e\colon (l, \varphi)$

$e_0\colon (l_3, \top)$

↓

$e_1\colon (l_4, x_0 = 0)$

↓

$e_5\colon (l_5, \neg(ret_0 = 0))$

↓

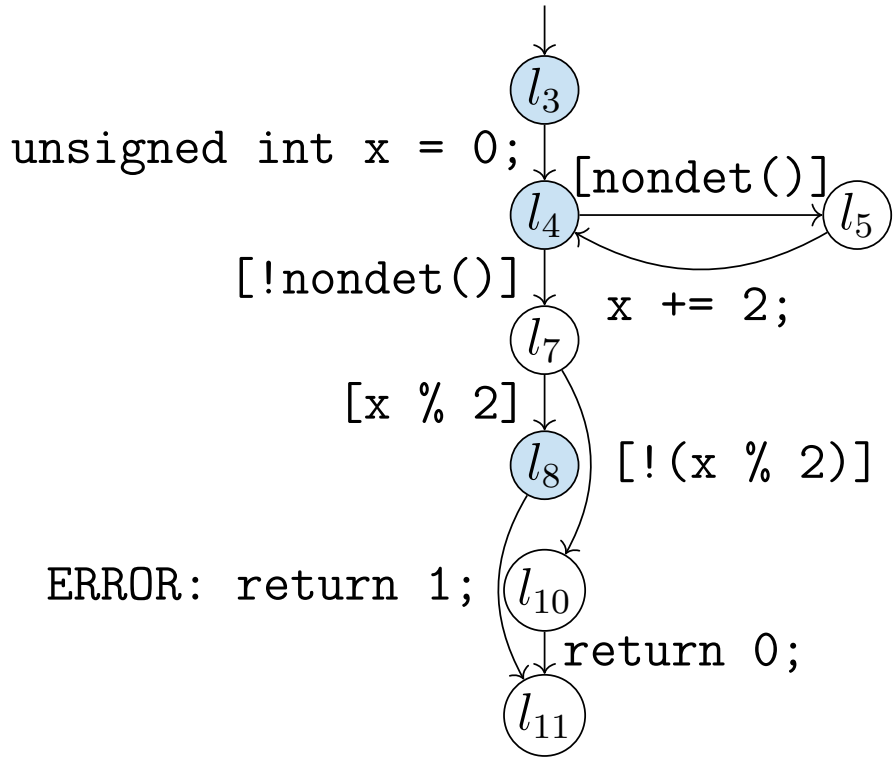
$e_6\colon (l_4, \neg(ret_0 = 0) \wedge x_1 = x_0 + 2)$

↓

$e_7\colon (l_7, ret_1 = 0)$

↓

$e_8\colon (l_8, ret_1 = 0 \wedge \neg(x_1 \% 2 = 0))$



(path to error location l_8 with one loop unrolling)

Example for One Loop Unrolling

$e\colon (l, \varphi)$

$e_0\colon (l_3, \top)$

↓

$e_1\colon (l_4, x_0 = 0)$

↓

$e_5\colon (l_5, \neg(ret_0 = 0))$

↓

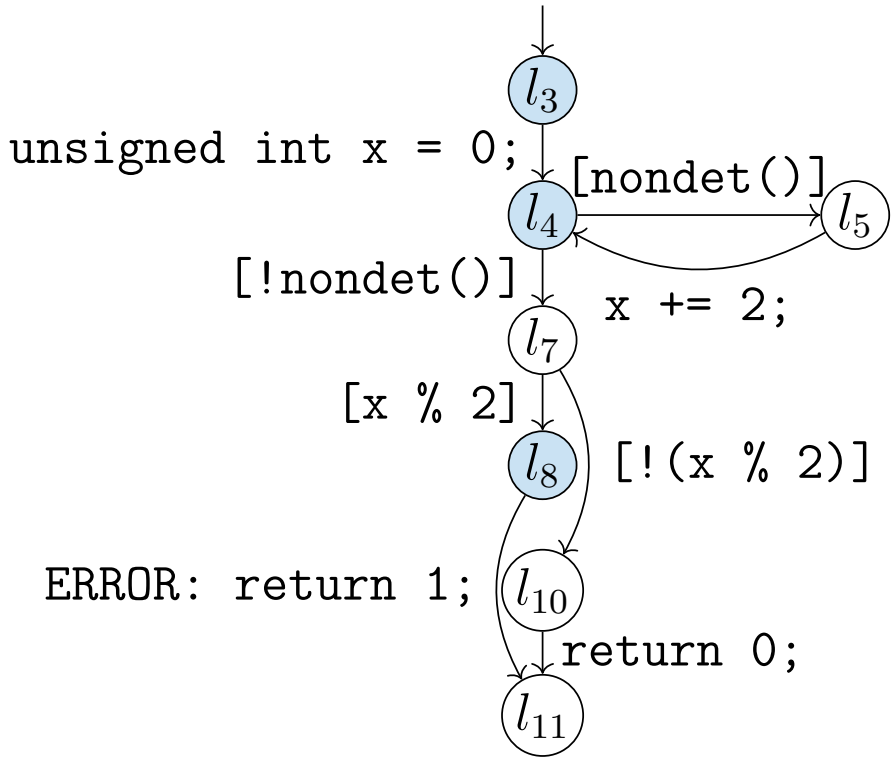
$e_6\colon (l_4, \neg(ret_0 = 0) \wedge x_1 = x_0 + 2)$

↓

$e_7\colon (l_7, ret_1 = 0)$

↓

$e_8\colon (l_8, ret_1 = 0 \wedge \neg(x_1 \% 2 = 0))$



$$\underbrace{x_0 = 0}_{I(s_0) \text{ from } e_1} \wedge \underbrace{\neg(ret_0 = 0) \wedge x_1 = x_0 + 2}_{T(s_0, s_1) \text{ from } e_6} \wedge \underbrace{ret_1 = 0 \wedge \neg(x_1 \% 2 = 0)}_{\neg P(s_1) \text{ from } e_8}$$

$$x_1 \% 2 = 0$$

Example for One Loop Unrolling

$e\colon (l, \varphi)$

$e_0\colon (l_3, \top)$

↓

$e_1\colon (l_4, x_0 = 0)$

↓

$e_5\colon (l_5, \neg(ret_0 = 0))$

↓

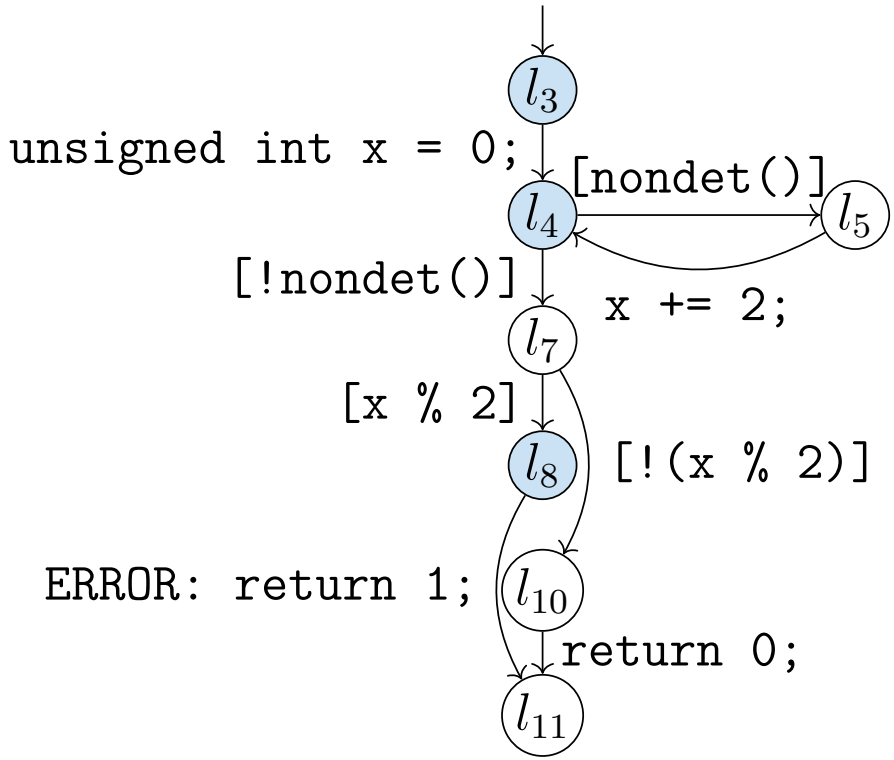
$e_6\colon (l_4, \neg(ret_0 = 0) \wedge x_1 = x_0 + 2)$

↓

$e_7\colon (l_7, ret_1 = 0)$

↓

$e_8\colon (l_8, ret_1 = 0 \wedge \neg(x_1 \% 2 = 0))$



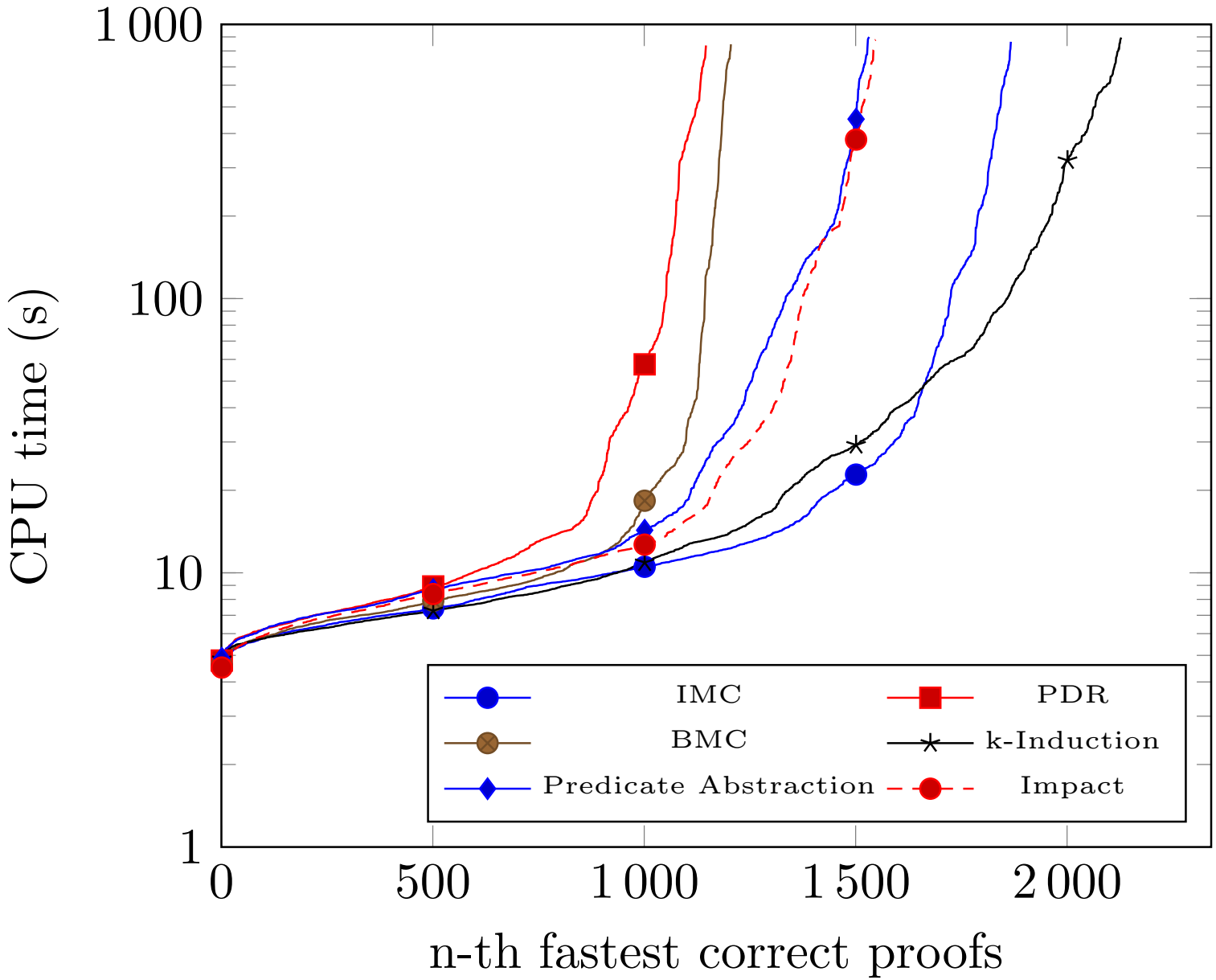
$$\underbrace{x_0 = 0}_{I(s_0) \text{ from } e_1} \wedge \underbrace{\neg(ret_0 = 0) \wedge x_1 = x_0 + 2}_{T(s_0, s_1) \text{ from } e_6} \wedge \underbrace{ret_1 = 0 \wedge \neg(x_1 \% 2 = 0)}_{\neg P(s_1) \text{ from } e_8}$$

$$\underbrace{x_0 \% 2 = 0}_{C_1(s_0)} \wedge \underbrace{\neg(ret_0 = 0) \wedge x_1 = x_0 + 2}_{T(s_0, s_1) \text{ from } e_6} \wedge \underbrace{ret_1 = 0 \wedge \neg(x_1 \% 2 = 0)}_{\neg P(s_1) \text{ from } e_8}$$

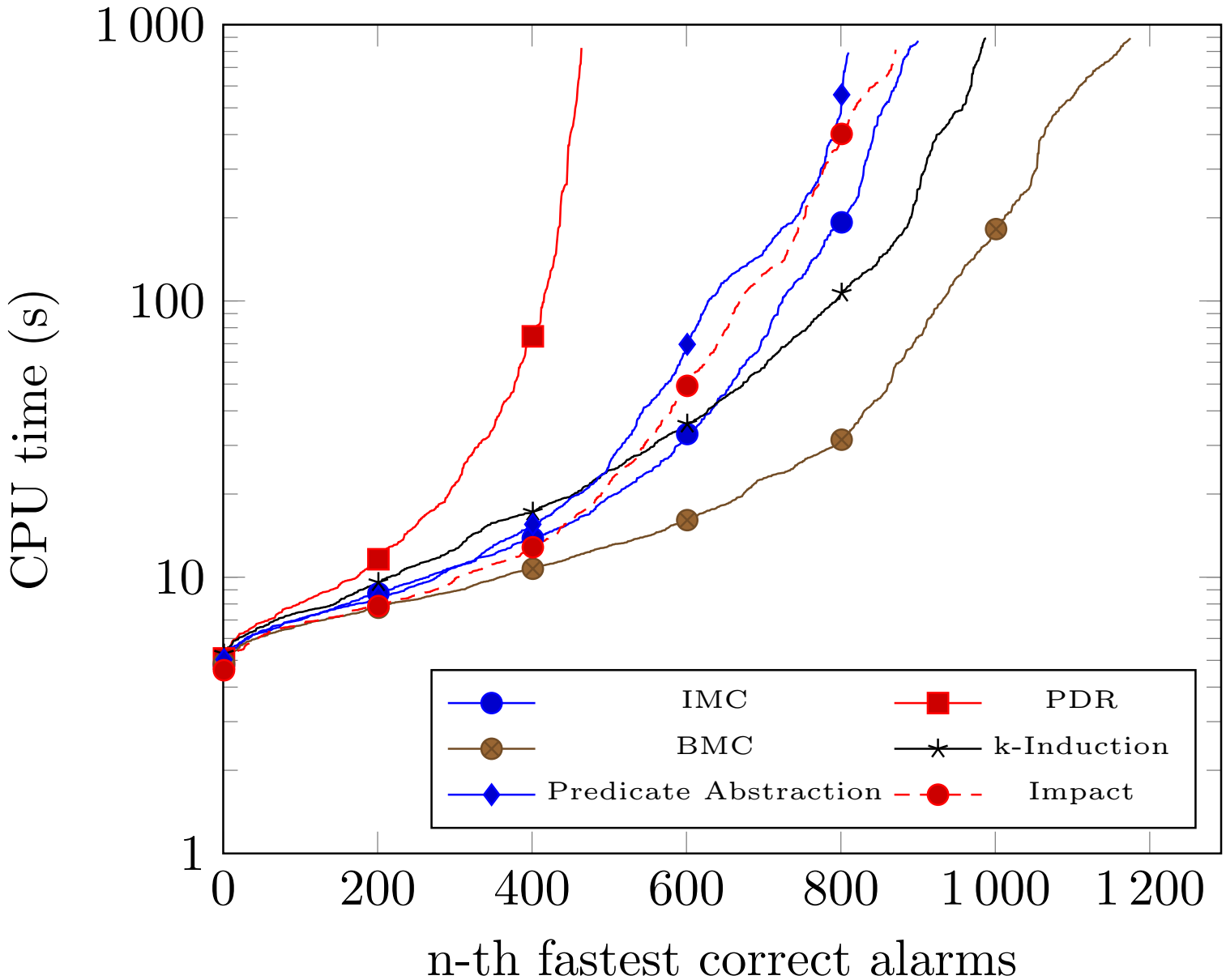
Experimental Setup

- CPAchecker revision 40806
- Interpolants provided by MathSAT5
- Compared algorithms
  - IMC
  - PDR
  - BMC
  - k-Induction
  - Predicate abstraction
  - Impact
- Subset of *ReachSafety* from SV-COMP '22 [1]
  - Safe: 4234 tasks
  - Unsafe: 1793 tasks

Quantile Plot: Safe Tasks



Quantile Plot: Unsafe Tasks



Conclusion

- RQ1: Efficient adoption achieved by large-block encoding
- RQ2: Most efficient and effective interpolation-based algorithm in our evaluation
- Successful adoption of IMC to software
- 19 year old knowledge gap closed

https://www.sosy-lab.org/research/cpa-imc/ [8]

References I

[1] Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_20

[2] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351147

[3] Beyer, D., Dangl, M.: Software verification with PDR: An implementation of the state of the art. In: Proc. TACAS (1). pp. 3–21. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_1

[4] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_42

[5] Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. J. Autom. Reasoning **60**(3), 299–335 (2018). https://doi.org/10.1007/s10817-017-9432-6

[6] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16

References II

[7] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010)

[8] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and sat-based model checking revisited: Adoption to software verification. Tech. rep., LMU Munich (August 2022). https://doi.org/10.48550/arXiv.2208.05046, supplementary webpage

[9] Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003). https://doi.org/10.1016/S0065-2458(03)58003-2

[10] Birgmeier, J., Bradley, A.R., Weissenbacher, G.: Counterexample to induction-guided abstraction-refinement (CTIGAR). In: Proc. CAV. pp. 831–848. LNCS 8559, Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_55

[11] Bradley, A.R.: SAT-based model checking without unrolling. In: Proc. VMCAI. pp. 70–87. LNCS 6538, Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_7

[12] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15

References III

[13] Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957). https://doi.org/10.2307/2963593

[14] Heizmann, M., Hoenicke, J., Podelski, A.: Refinement of trace abstraction. In: Proc. SAS. pp. 69–85. LNCS 5673, Springer (2009). https://doi.org/10.1007/978-3-642-03237-0_7

[15] Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: Proc. POPL. pp. 232–244. ACM (2004). https://doi.org/10.1145/964001.964021

[16] Jhala, R., McMillan, K.L.: Interpolant-based transition relation approximation. In: Proc. CAV. pp. 39–51. LNCS 3576, Springer (2005). https://doi.org/10.1007/11513988_6

[17] Kahsai, T., Tinelli, C.: PKind: A parallel k-induction based model checker. In: Proc. Int. Workshop on Parallel and Distributed Methods in Verification. pp. 55–62. EPTCS 72, EPTCS (2011). https://doi.org/10.4204/EPTCS.72.6

[18] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725, Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_1

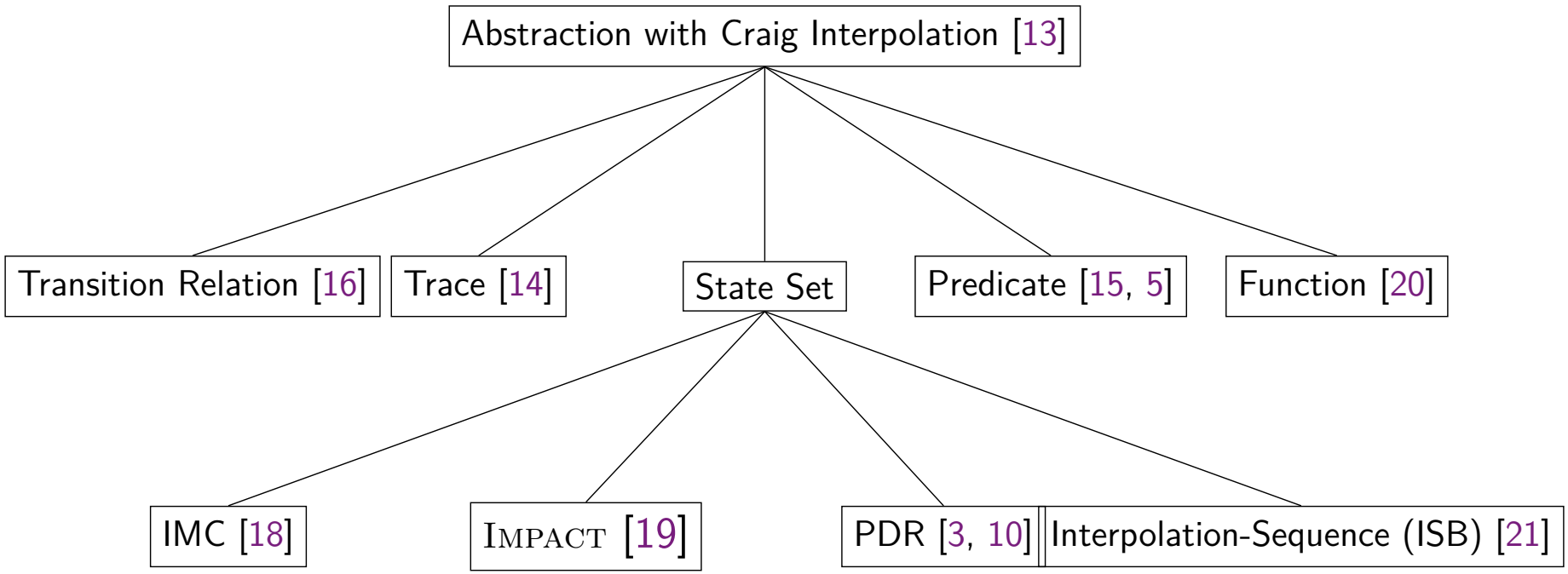
[19] McMillan, K.L.: Lazy abstraction with interpolants. In: Proc. CAV. pp. 123–136. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_14

References IV

[20] Sery, O., Fedyukovich, G., Sharygina, N.: Interpolation-based function summaries in bounded model checking. In: Proc. HVC. pp. 160–175. LNCS 7261, Springer (2011). https://doi.org/10.1007/978-3-642-34188-5_15

[21] Vizel, Y., Grumberg, O.: Interpolation-sequence based model checking. In: Proc. FMCAD. pp. 1–8. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351148

Interpolation-Based Verification Approaches



IMC: Main Procedure

Input: $\mathbb{D} = \mathbb{L} \times \mathbb{P} \times \mathbb{LB}$ and k_{max}
Output: **false** if l_E reachable; **true** if fixed point obtained;
unknown otherwise

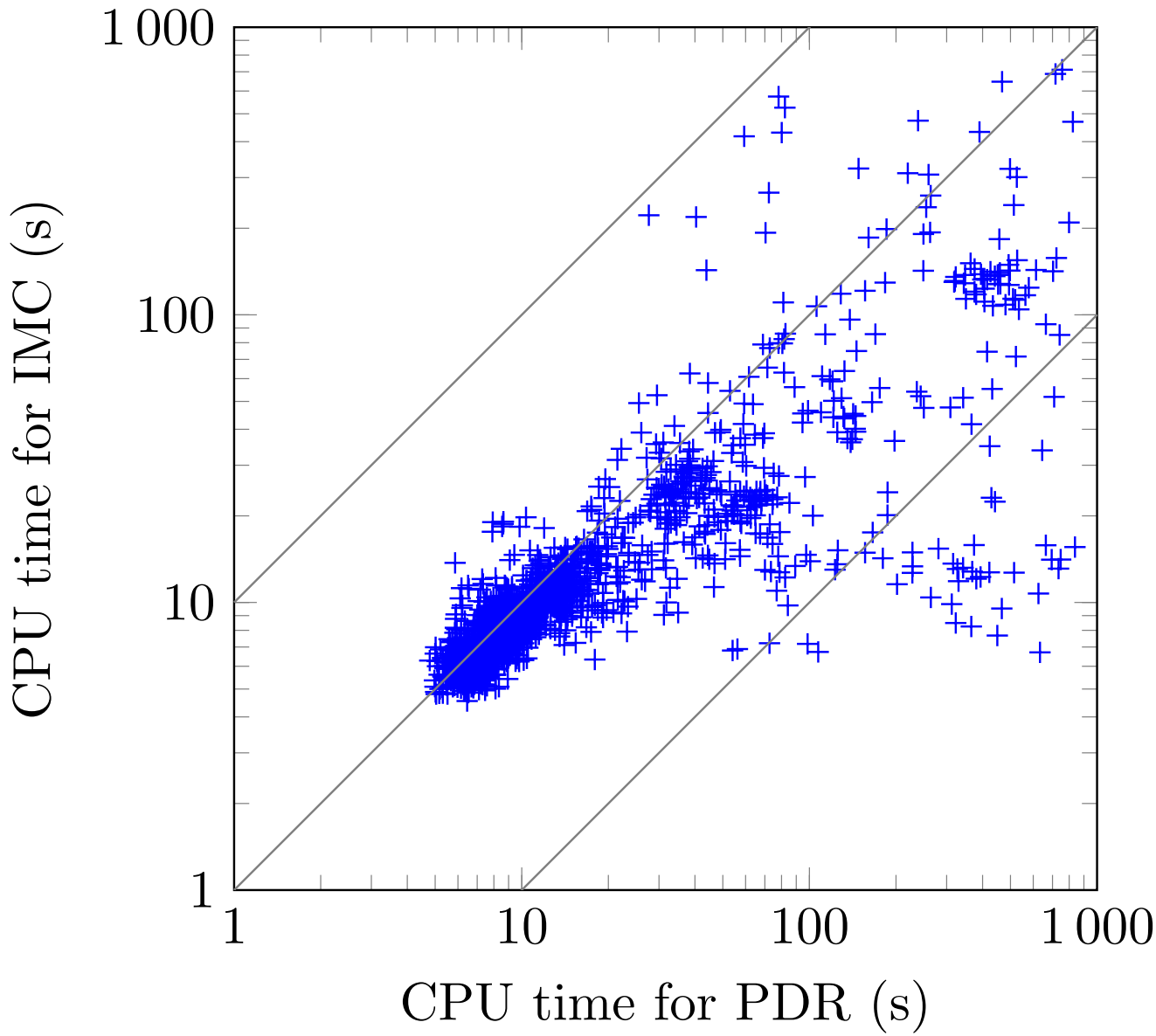
1: $k := 1$
2: $e_0 := (l_0, (true, l_0, true, true), \{l_H \mapsto -1\})$
3: reached := waitlist := $\{e_0\}$
4: **while** $k \leq k_{max}$ **do**
5: (reached, waitlist) := CPA++($\mathbb{D}$, reached, waitlist, k)
6: $(\sigma_p, \sigma_l, \sigma_s)$:= collect_formulas(reached, k)
7: **if** sat($\sigma_p \wedge \sigma_l \wedge \sigma_s$) **then**
8: **return false**
9: **if** $k > 1$ **and** reach_fixed_point(σ_p,σ_l,σ_s) **then**
10: **return true**
11: $k := k + 1$
12: **return unknown**

IMC: reach_fixed_point(σ_p,σ_l,σ_s)

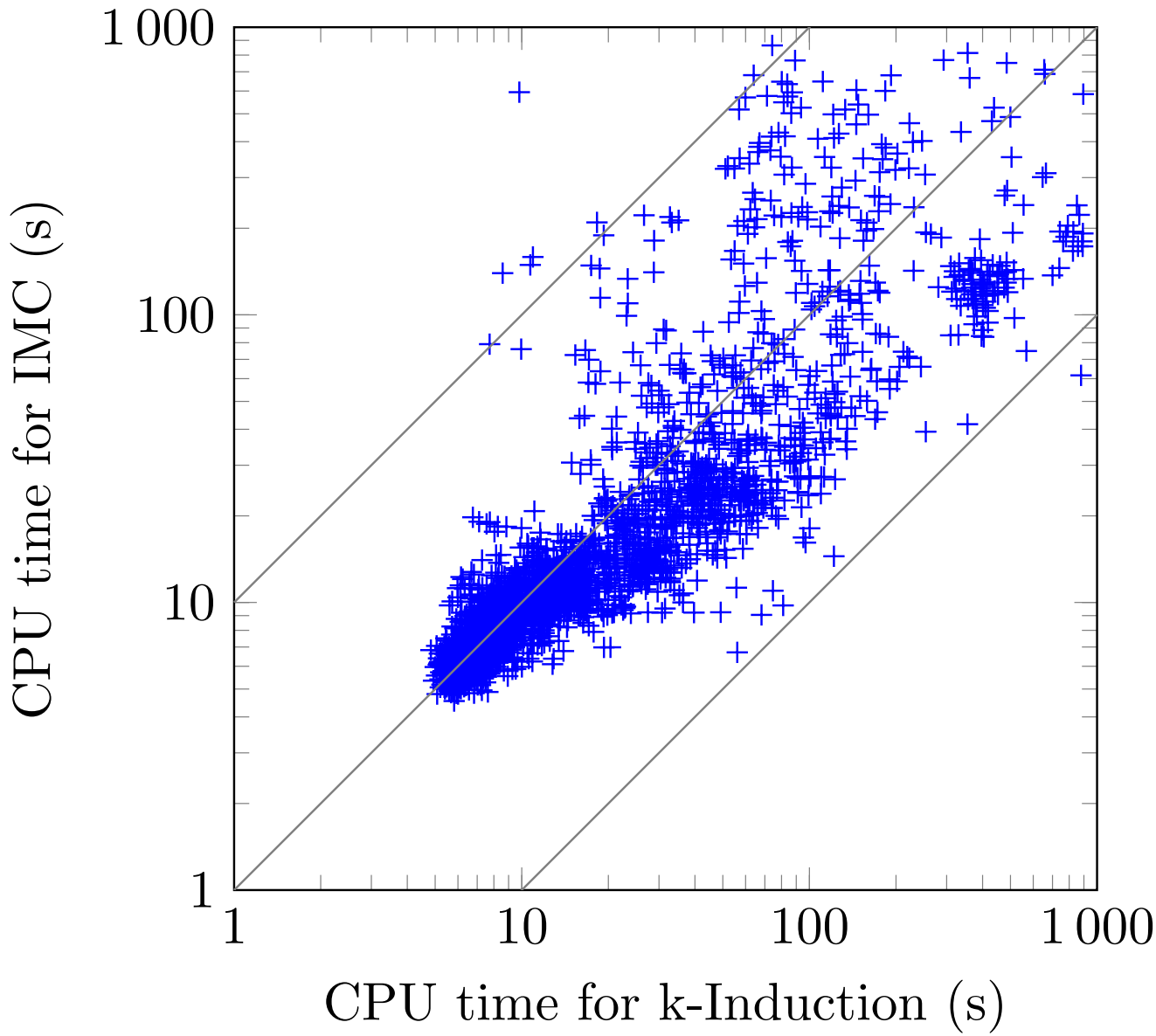
Input: σ_p, σ_l, and σ_s
Output: **true** if fixed point obtained; **false** otherwise

```
1: image := start := σ_p
2: while ¬sat(start ∧ σ_l ∧ σ_s) do
3:     τ := get_interpolant(start ∧ σ_l, σ_s)
4:     τ := shift_variable_index(τ, σ_p)
5:     if ¬sat(τ ∧ ¬image) then
6:         return true
7:     image := image ∨ τ
8:     start := τ
9: return false
```

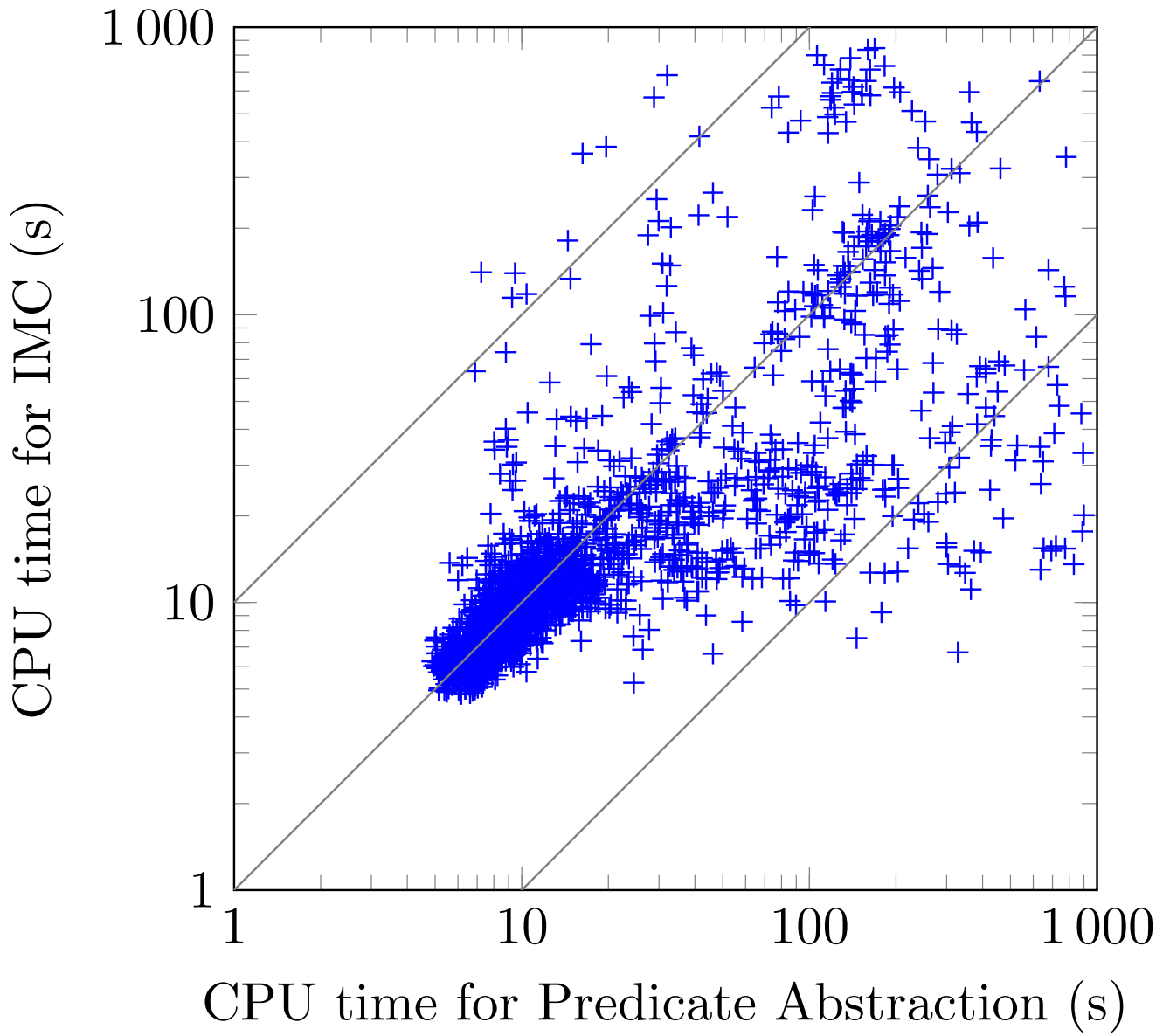
CPU-Time Scatter Plot: vs. PDR



CPU-Time Scatter Plot: vs. k-Induction



CPU-Time Scatter Plot: vs. Predicate Abstraction



CPU-Time Scatter Plot: vs. Impact