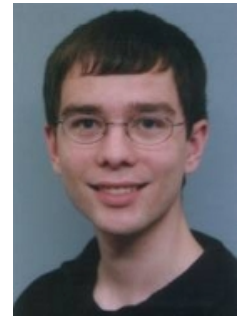
A Unifying View on SMT-Based Software Model Checking

Dirk Beyer, Matthias Dangl, and Philipp Wendler

LMU Munich, Germany





LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CPA

SoSy-Lab Software Systems

Based on:

Dirk Beyer, Matthias Dangl, Philipp Wendler:

A Unifying View on SMT-Based Software Verification

Journal of Automated Reasoning (2018).
`https://doi.org/10.1007/s10817-017-9432-6`
preprint: online on CPAchecker website under "Documentation"

SMT-based Software Model Checking

- Predicate Abstraction
  (Blast, CPAchecker, Slam, ...)
- Impact
  (CPAchecker, Impact, Wolverine, ...)
- Bounded Model Checking
  (Cbmc, CPAchecker, Esbmc, ...)
- k-Induction
  (CPAchecker, Esbmc, 2LS, ...)
- Property-Directed Reachability (PDR, also known as IC3)
  (Seahorn, VVT, ...)
- ...

SMT-based Software Model Checking

- Predicate Abstraction
  (Blast, CPAchecker, Slam, ...)
- Impact
  (CPAchecker, Impact, Wolverine, ...)
- Bounded Model Checking
  (Cbmc, CPAchecker, Esbmc, ...)
- k-Induction
  (CPAchecker, Esbmc, 2LS, ...)

Our Goals

- Provide a unifying conceptual framework for SMT-based algorithms
- Perform an extensive comparative evaluation
- Confirm intuitions about strengths
- Determine potential of extensions and combinations

Approach

- Understand, and, if necessary, re-formulate the algorithms
- Implement all algorithms in one tool (CPAchecker)
- Run the algorithms on a large set of benchmarks
- Measure efficiency and effectiveness

Experimental Validity: All Algorithms in one Tool

Compare algorithms, not tools:

- Share same front-end code
- Share same utilities
- Share same SMT-solver integration
- Share algorithm-independent optimizations

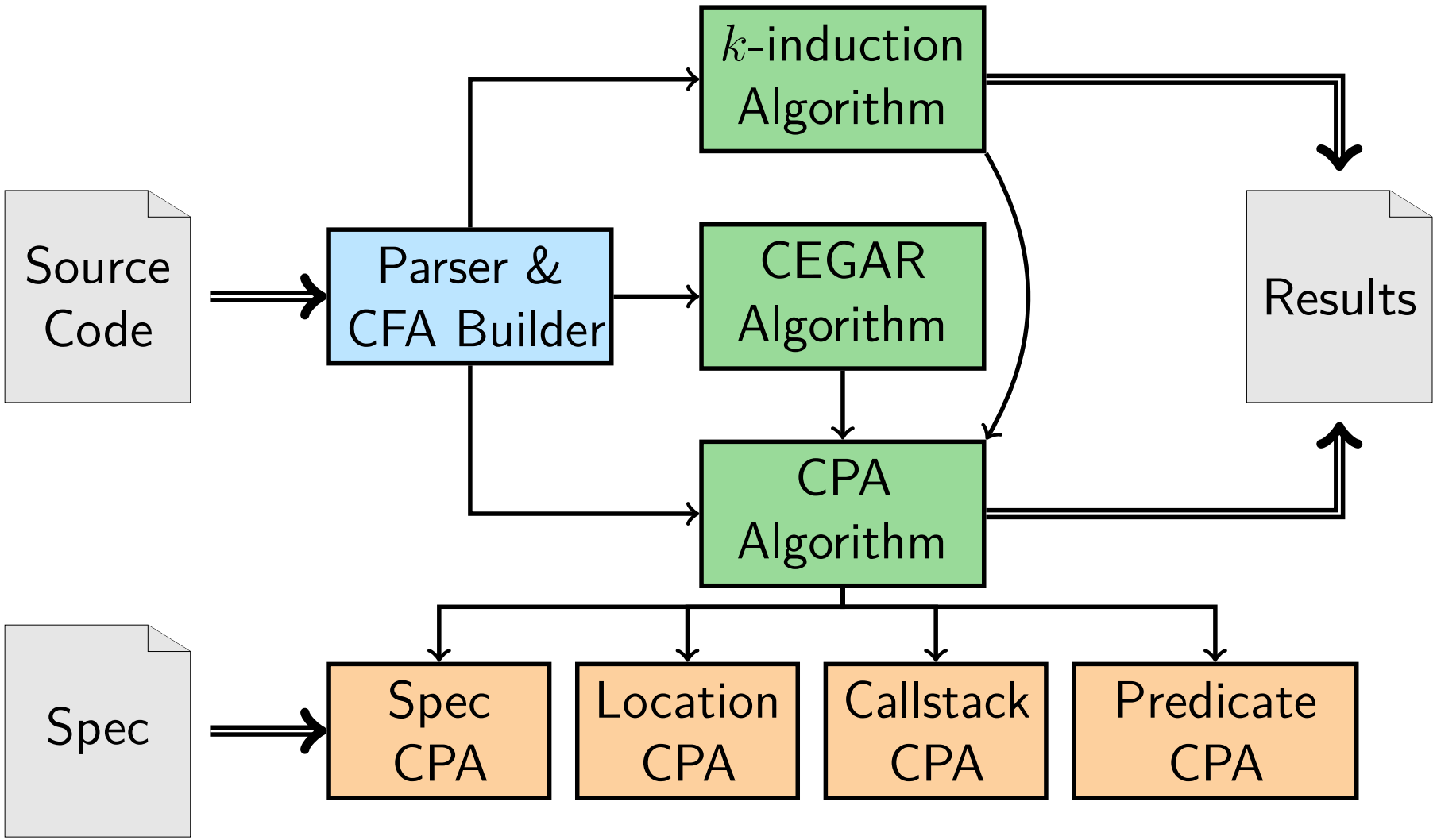
$\rightarrow$ Differences in performance must be caused by algorithms

Configurable Program Analysis

Configurable Program Analysis (CPA):

- Beyer, Henzinger, Théoduloz: [CAV'07]
- One single unifying algorithm for all algorithms based on state-space exploration
- Configurable components: Abstract domain, abstract-successor computation, path sensitivity, ...
- Separation of concerns:
  - Reusable CPA for tracking the program counter
  - Reusable CPA for tracking the callstack
  - Reusable CPA representing the specification
  - Reusable CPA for combining multiple CPAs

CPAchecker


k-induction Algorithm
Source Code
Parser & CFA Builder
CEGAR Algorithm
Results
CPA Algorithm
Spec
Spec CPA
Location CPA
Callstack CPA
Predicate CPA

Predicate Abstraction

- Predicate Abstraction
  - Graf, Saïdi: [CAV'97]
  - Abstract-Interpretation technique
  - Abstract domain constructed from a set of predicates π
  - Use CEGAR to add predicates to π (refinement)
  - Derive new predicates using Craig interpolation
  - Good for finding proofs

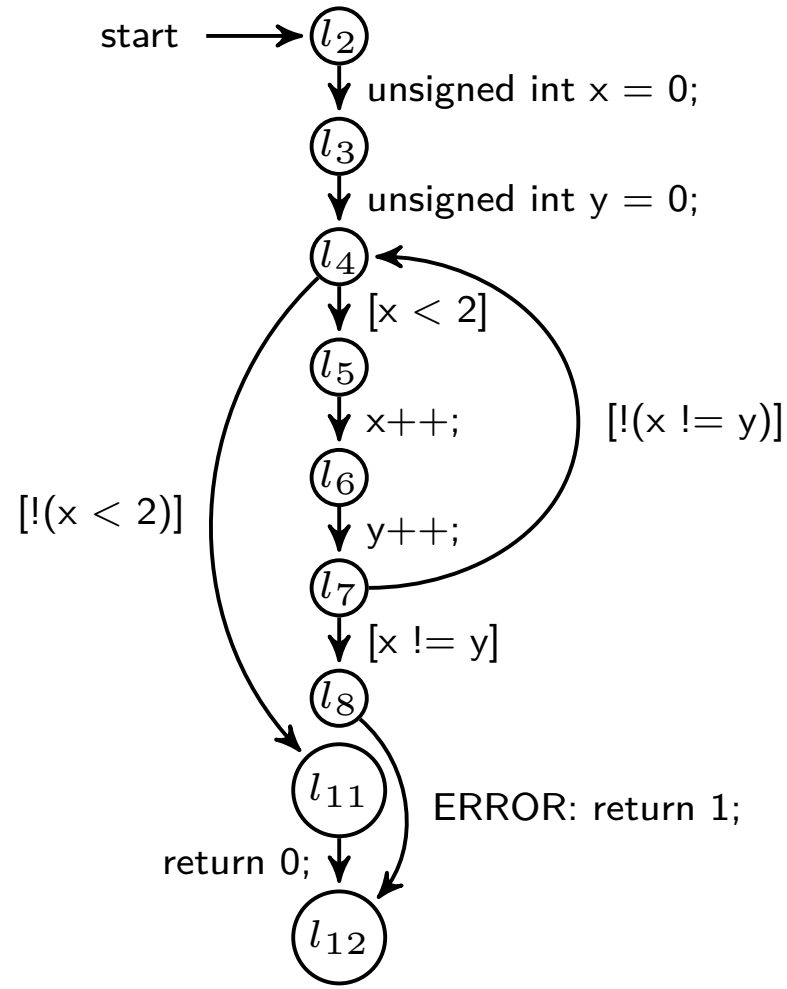
Adjustable-Block Encoding

Adjustable-Block Encoding:

- Abstraction computation is expensive
- Abstraction is not necessary after every transition
- Track precise path formula between abstraction states
- Reset path formula and compute abstraction formula at abstraction states
- Large-Block Encoding: Abstraction only at loop (and function) heads
- Adjustable-Block Encoding: Introduce block operator "blk" to make it configurable

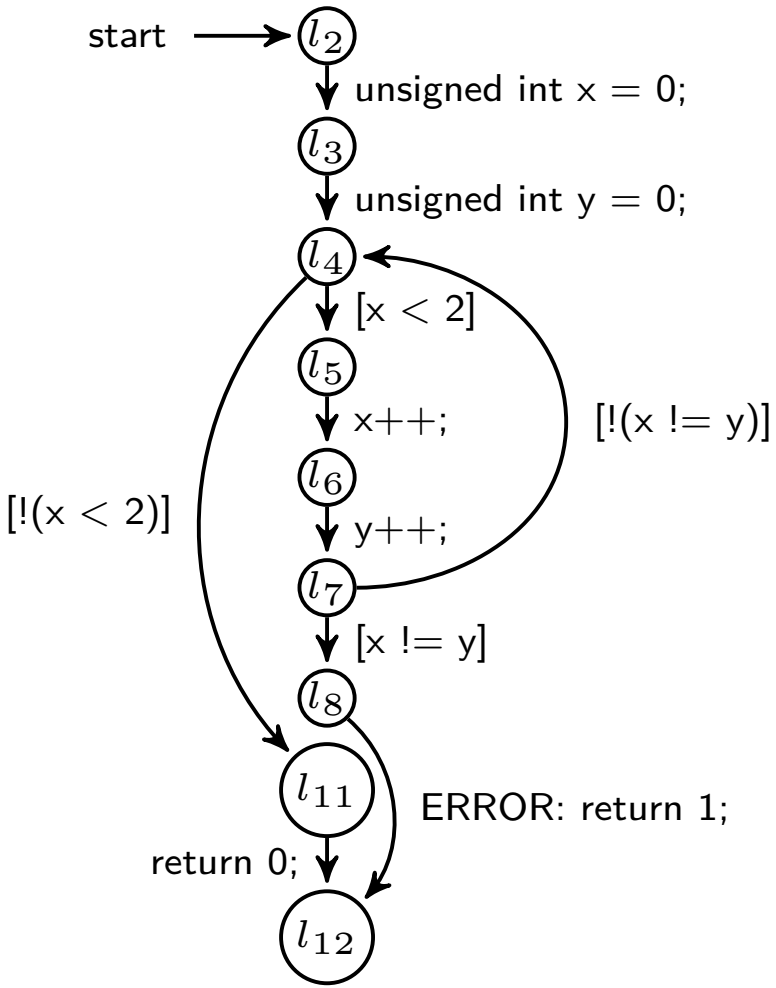
Example Program

```
1  int main() {
2    unsigned int x = 0;
3    unsigned int y = 0;
4    while (x < 2) {
5      x++;
6      y++;
7      if (x != y) {
8        ERROR: return 1;
9      }
10   }
11   return 0;
12 }
```



Predicate Abstraction: Example

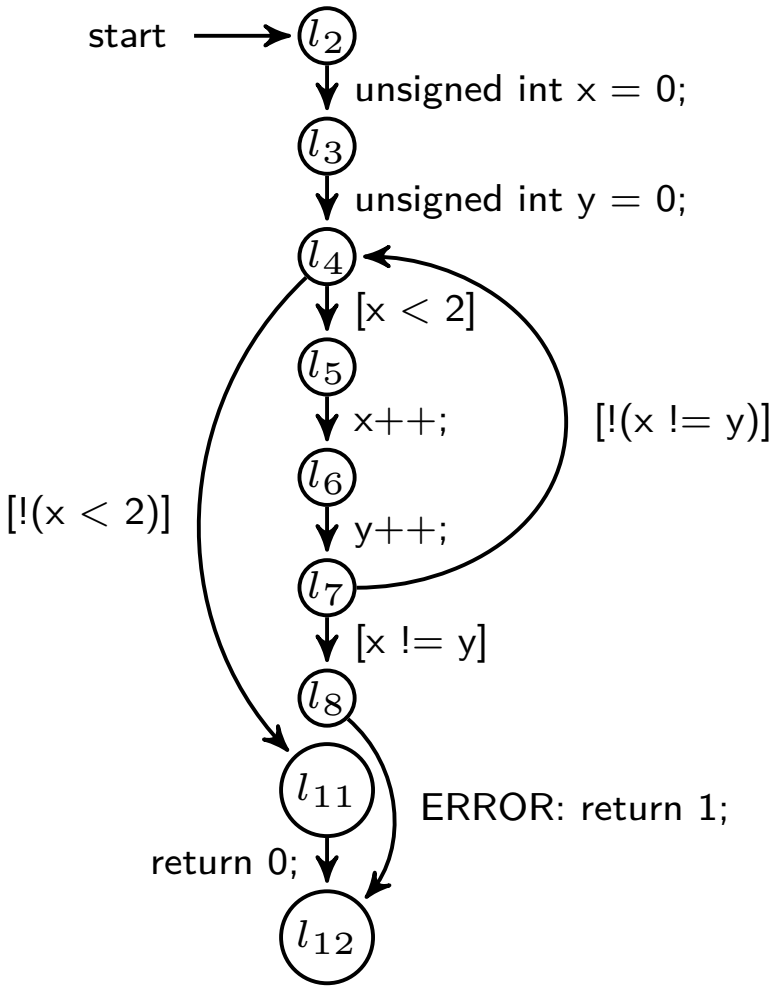
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

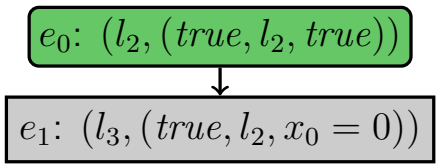


e_0: $(l_2, (true, l_2, true))$

Predicate Abstraction: Example

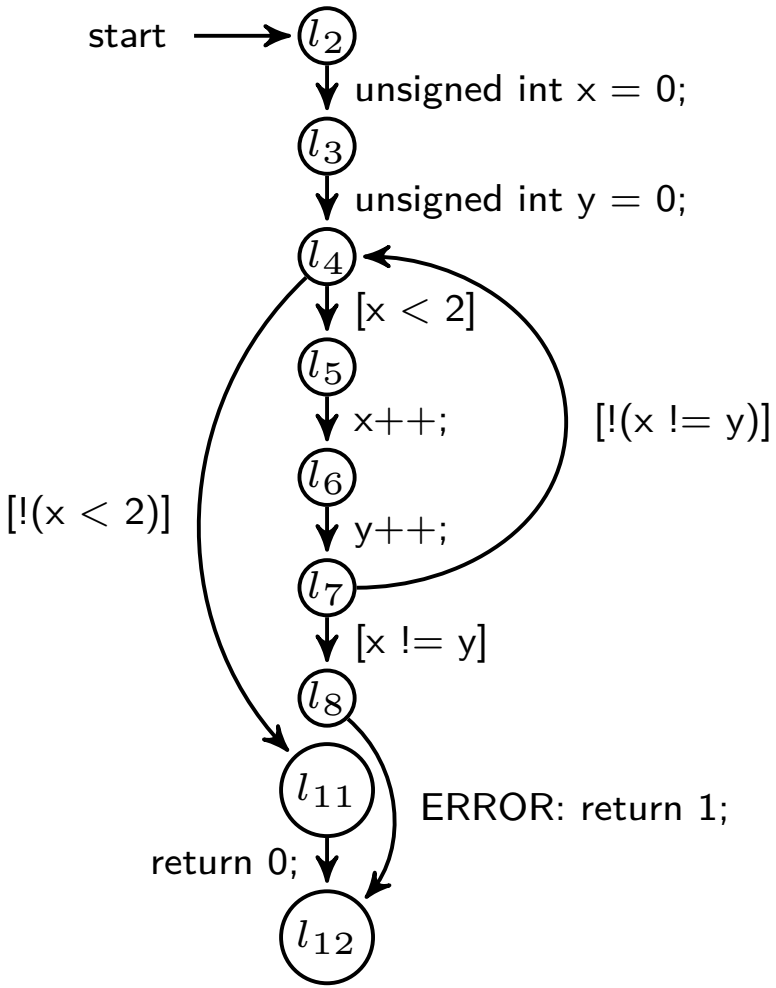
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

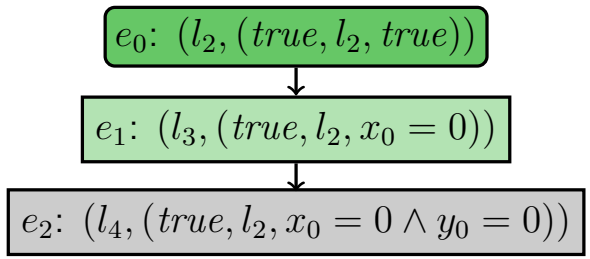




Predicate Abstraction: Example

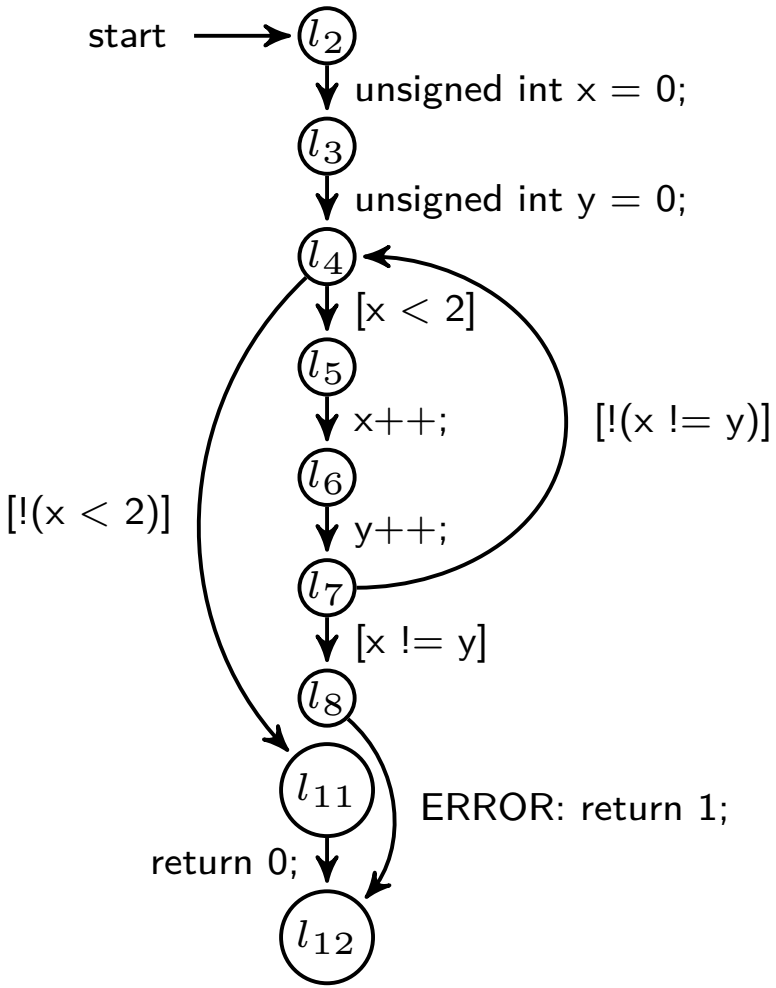
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

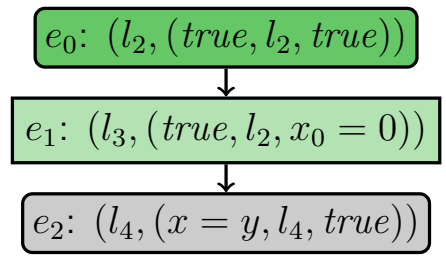




Predicate Abstraction: Example

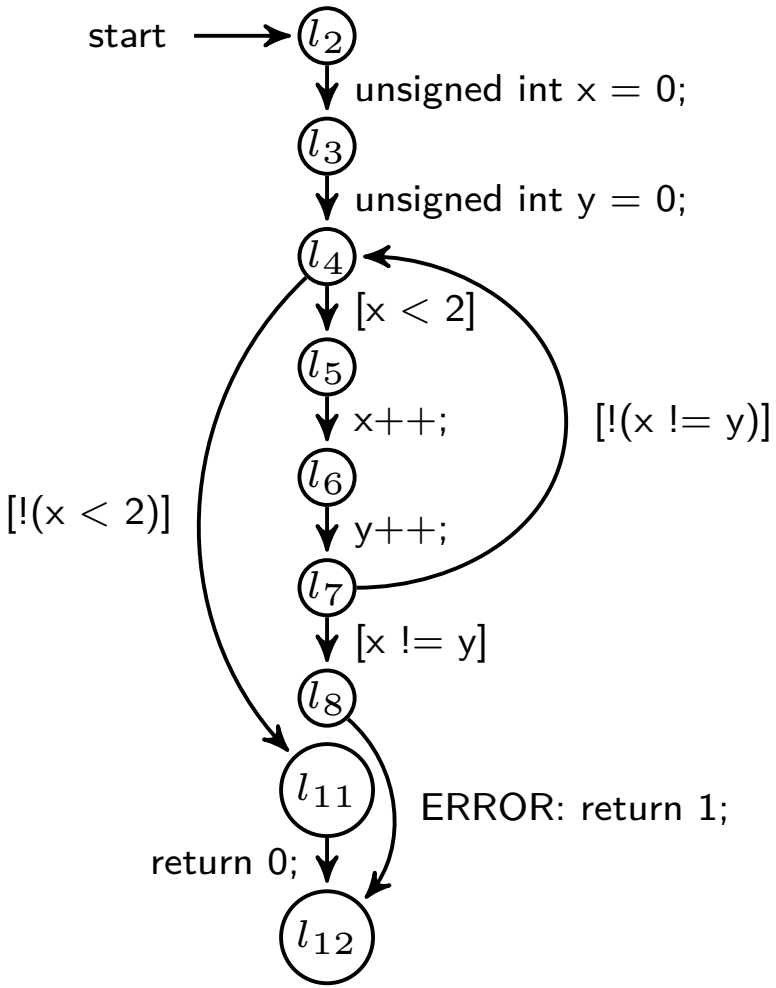
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

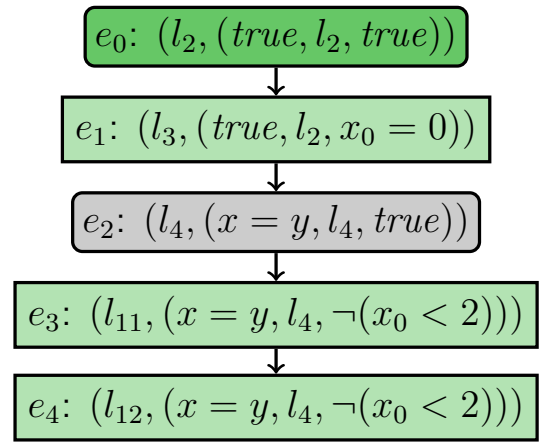




Predicate Abstraction: Example

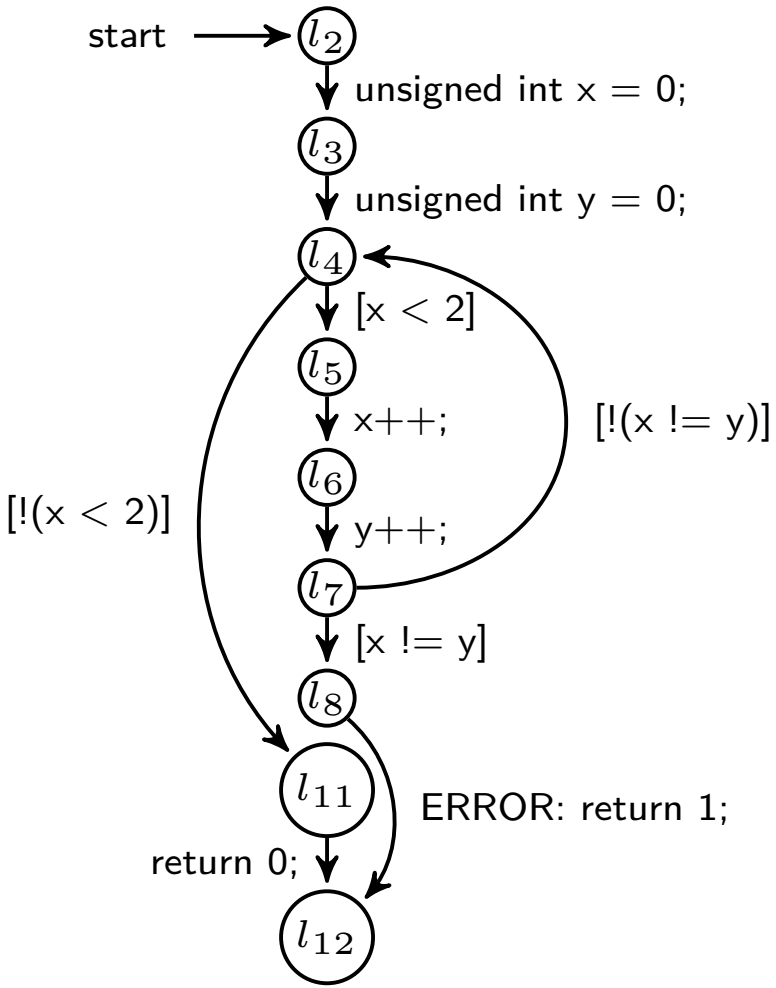
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

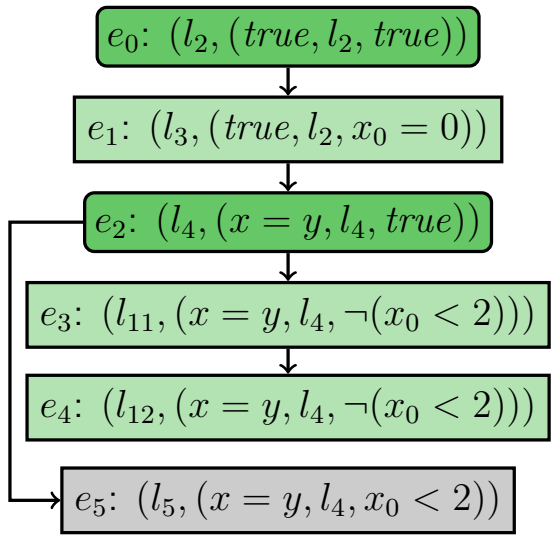




Predicate Abstraction: Example

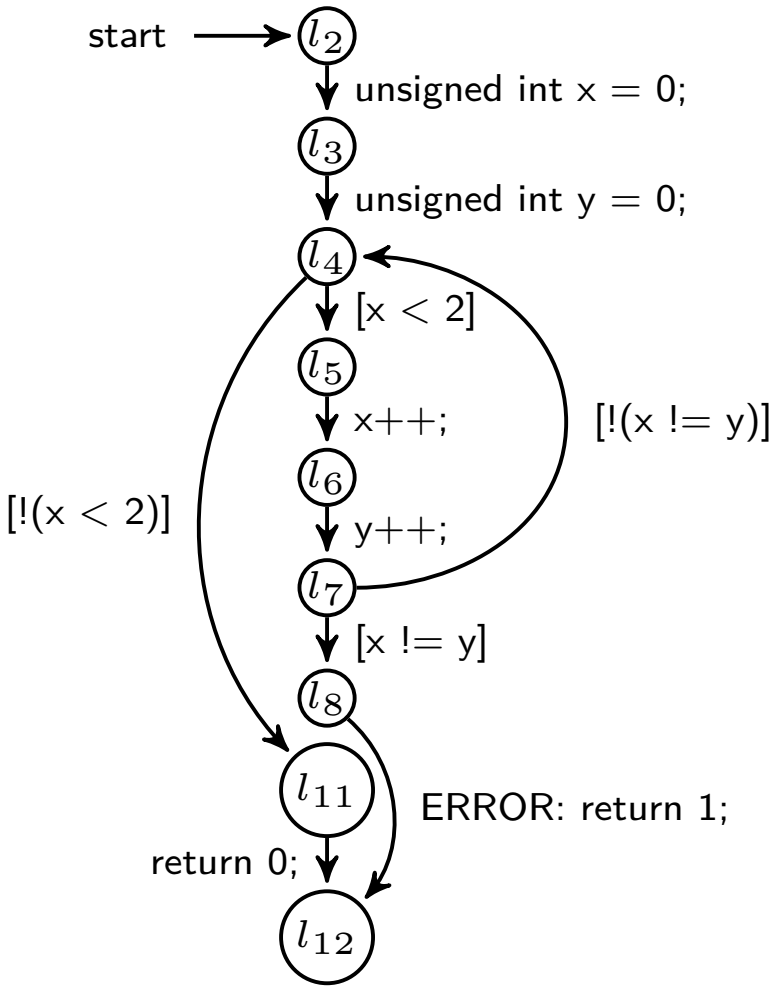
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

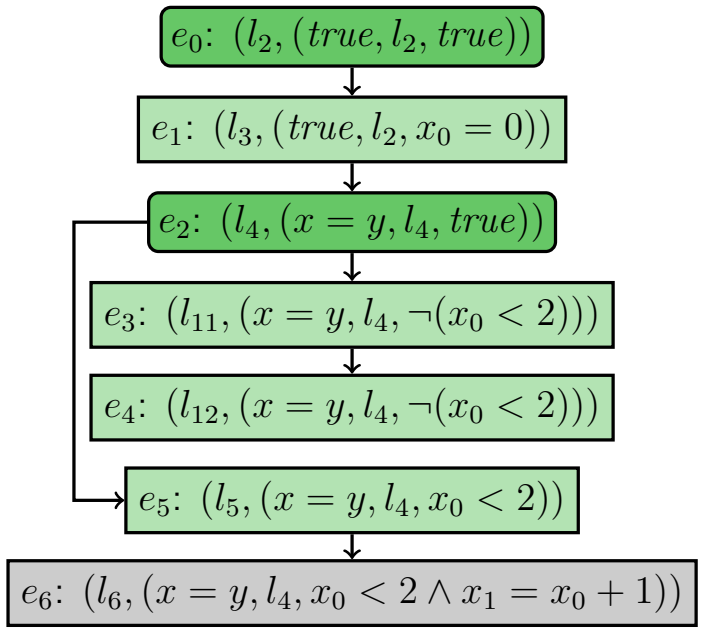




Predicate Abstraction: Example

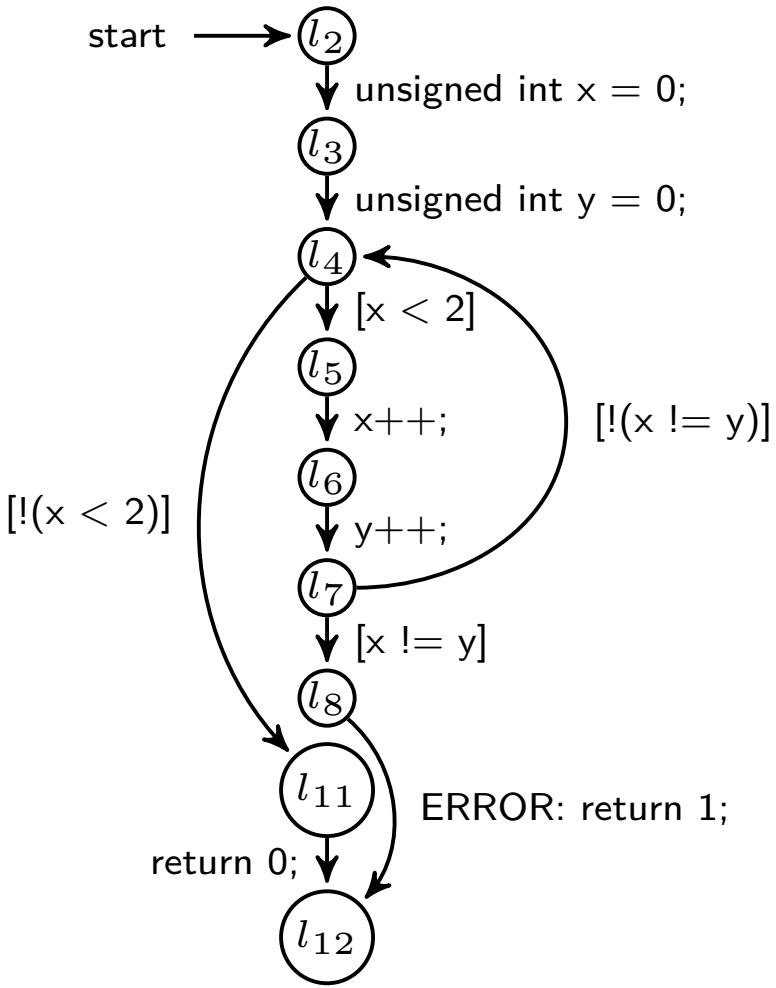
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

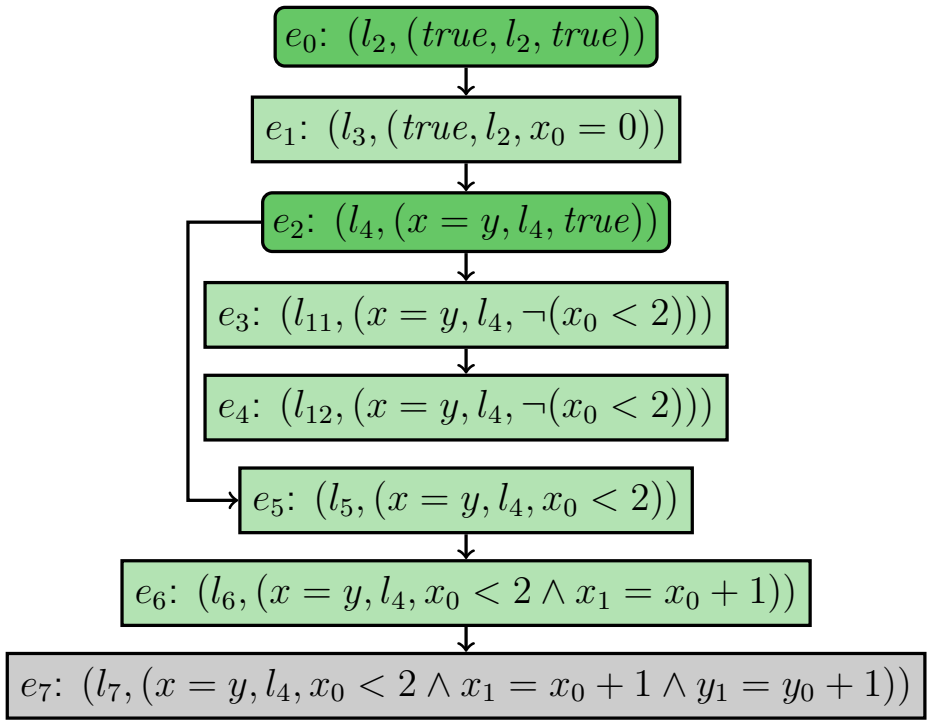




Predicate Abstraction: Example

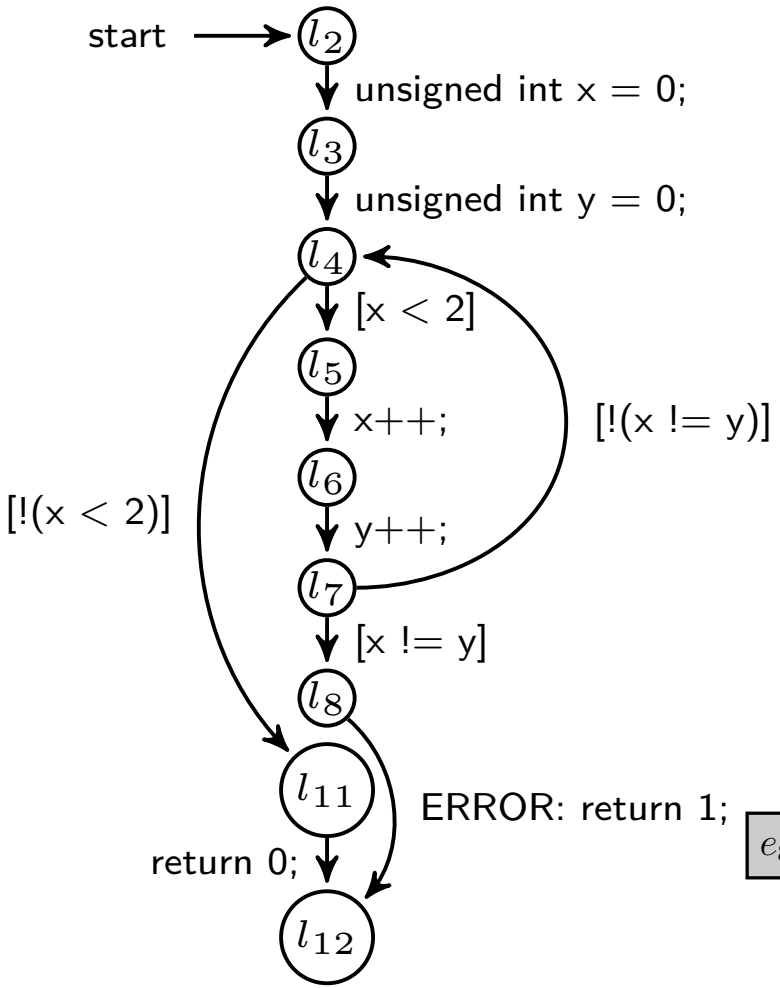
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

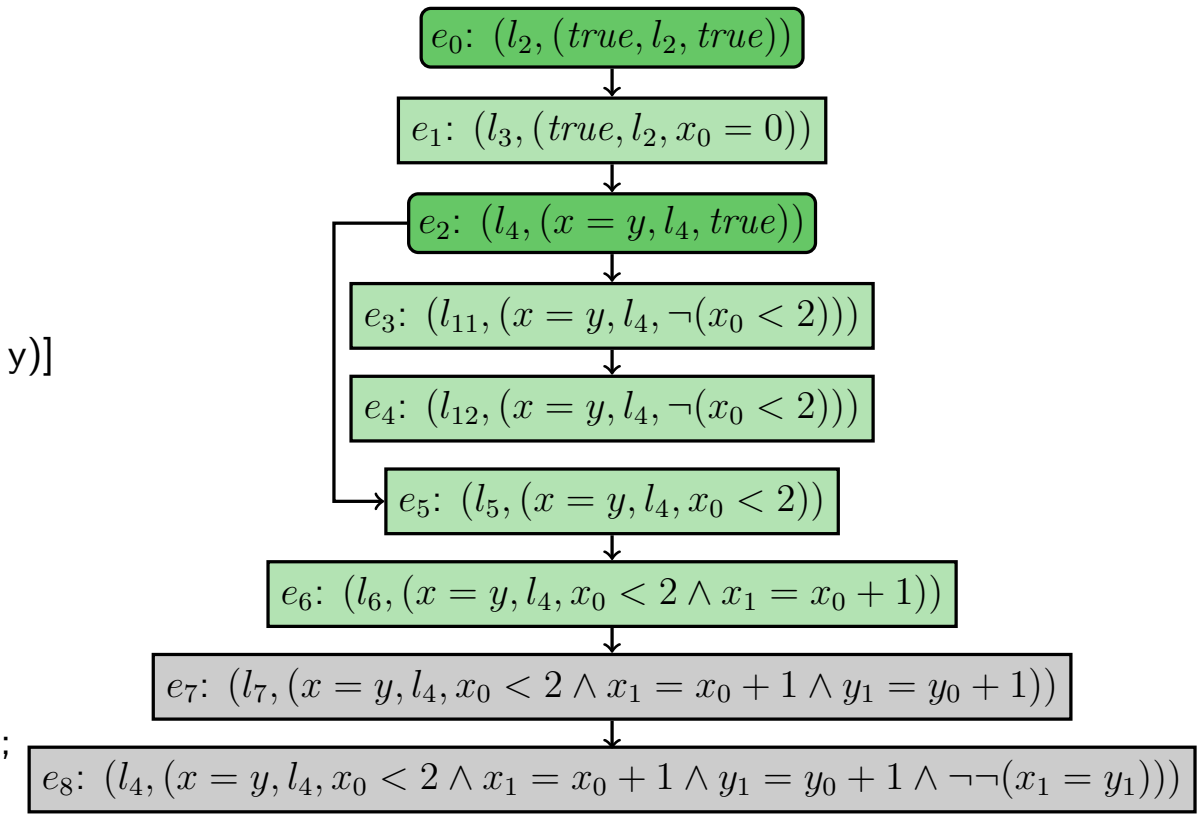




Predicate Abstraction: Example

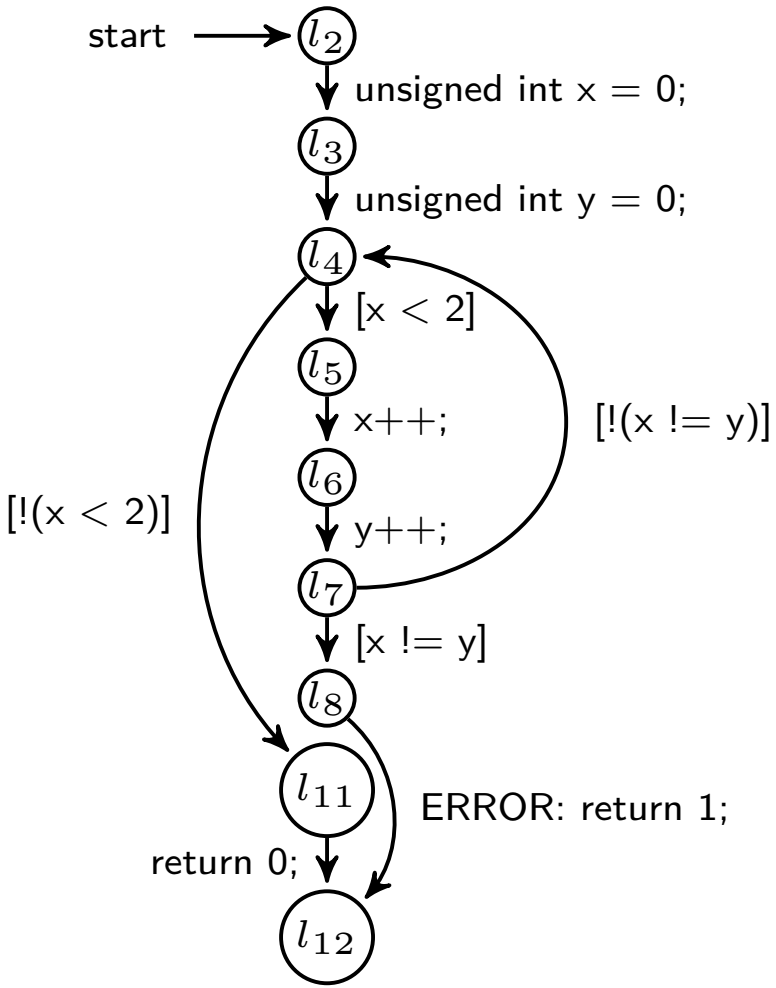
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

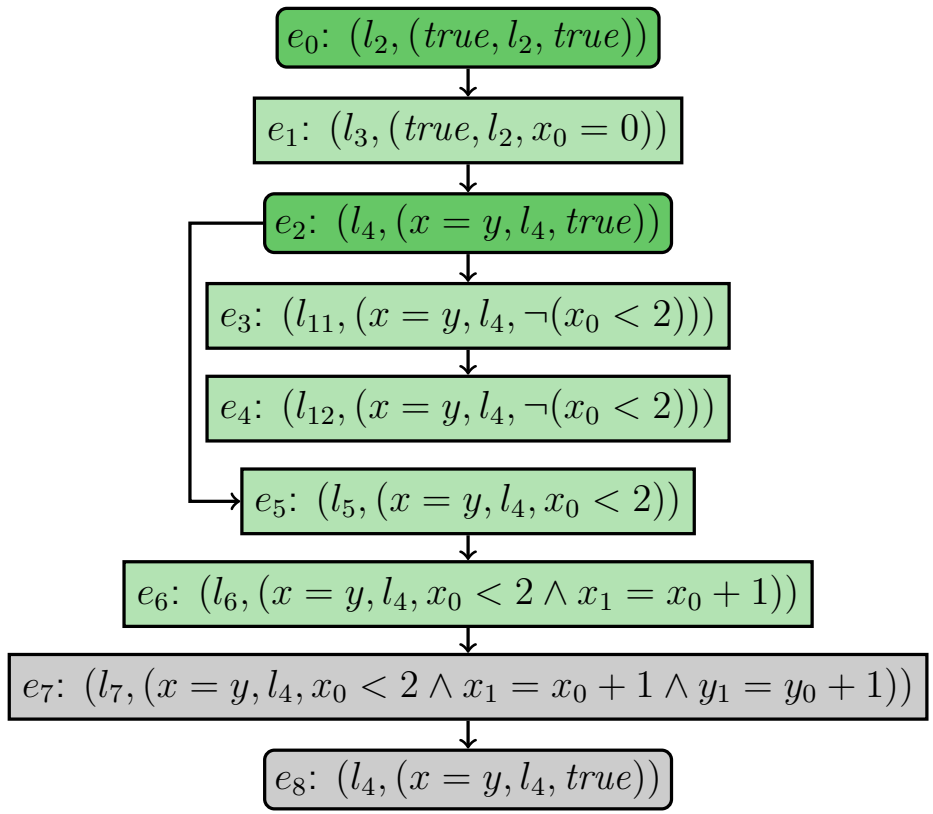




Predicate Abstraction: Example

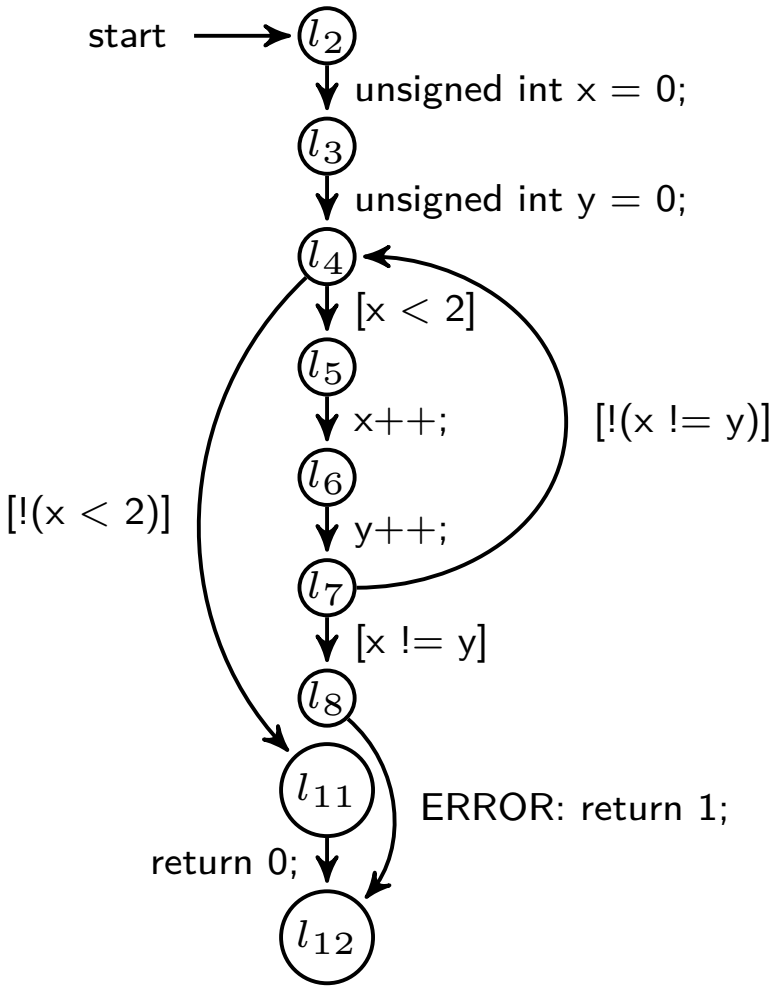
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

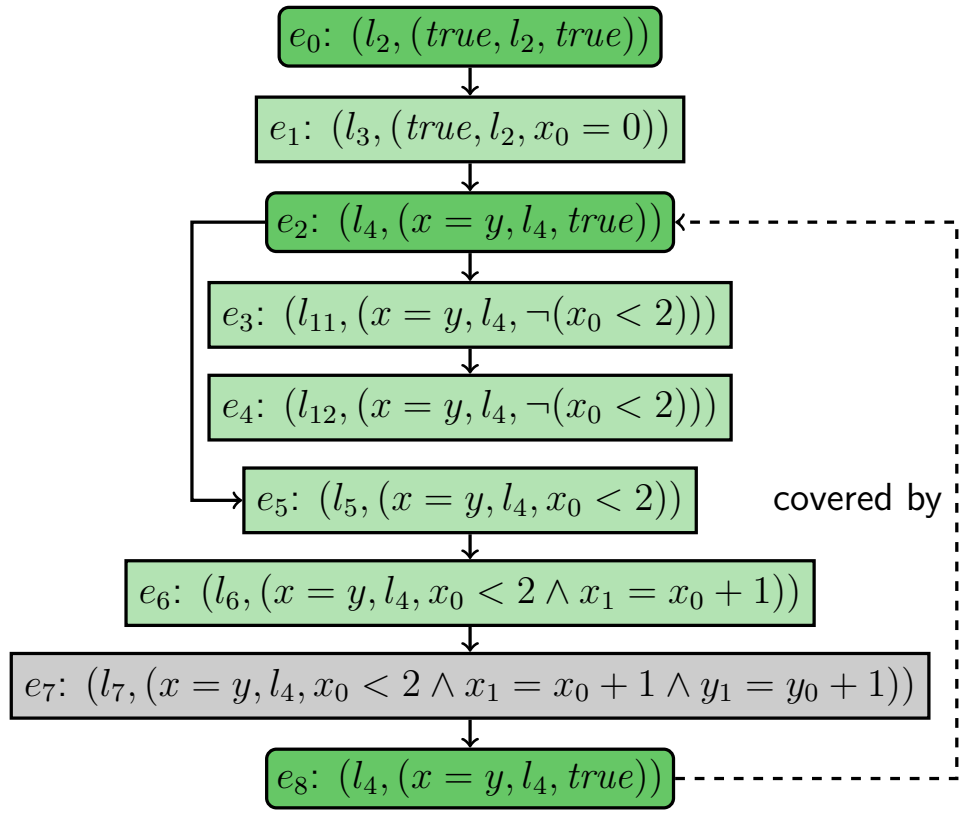




Predicate Abstraction: Example

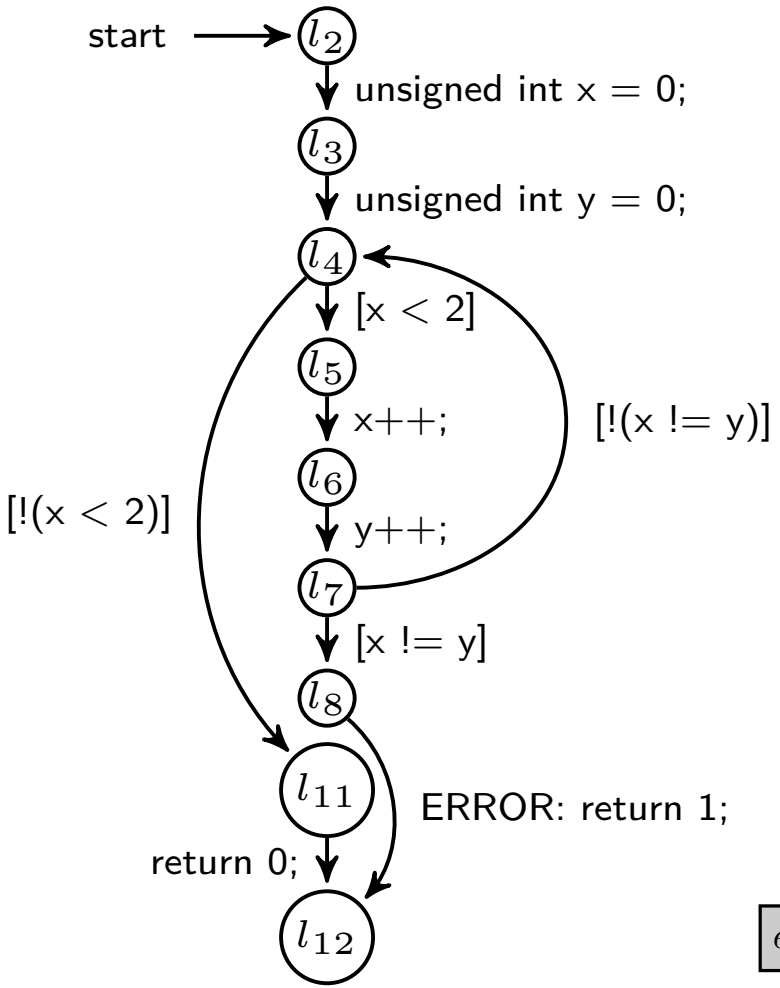
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

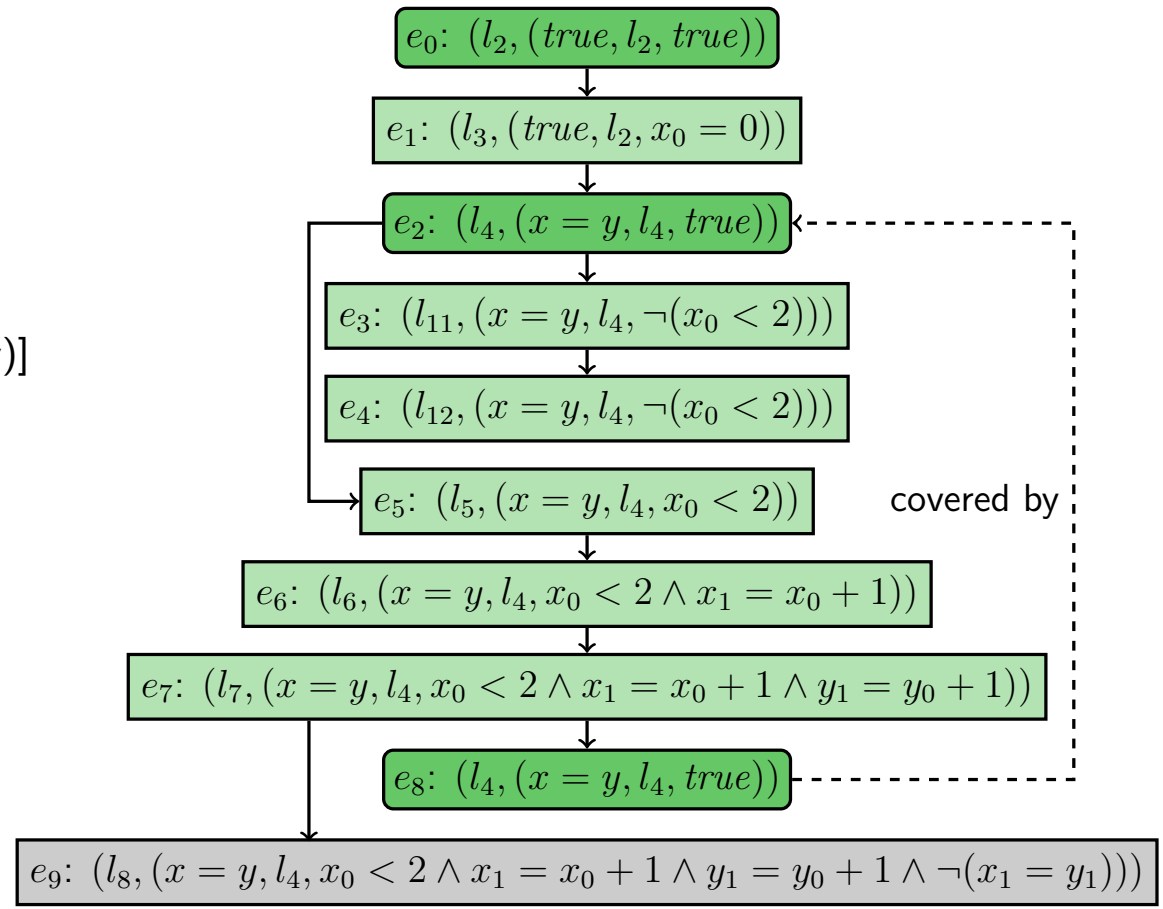




Predicate Abstraction: Example

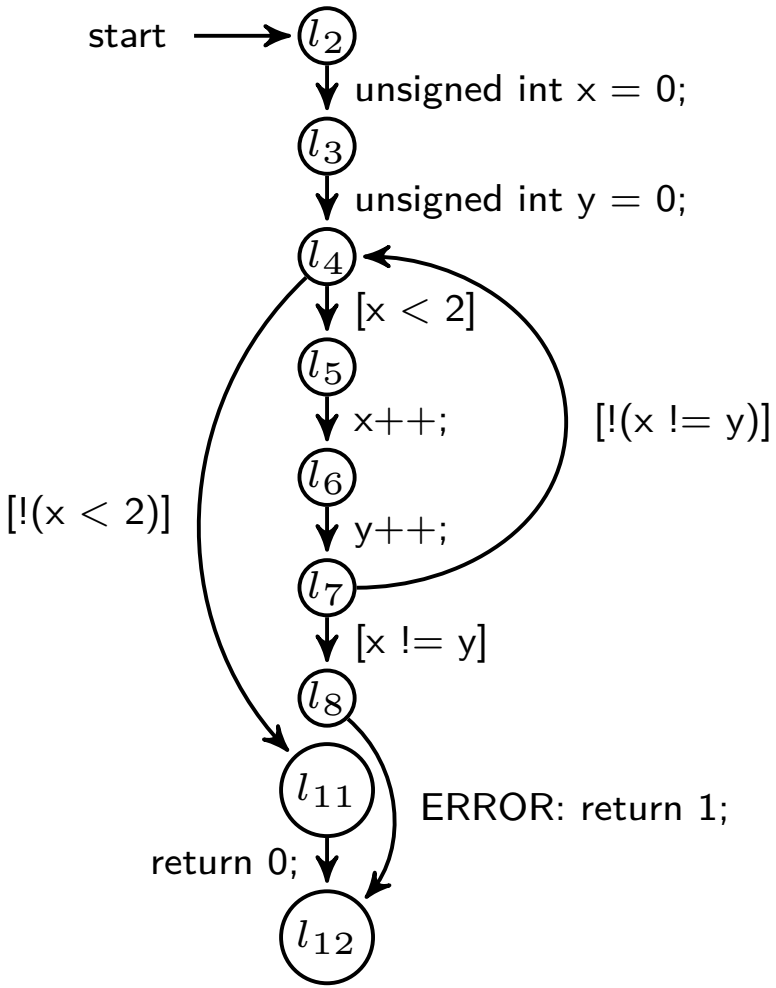
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

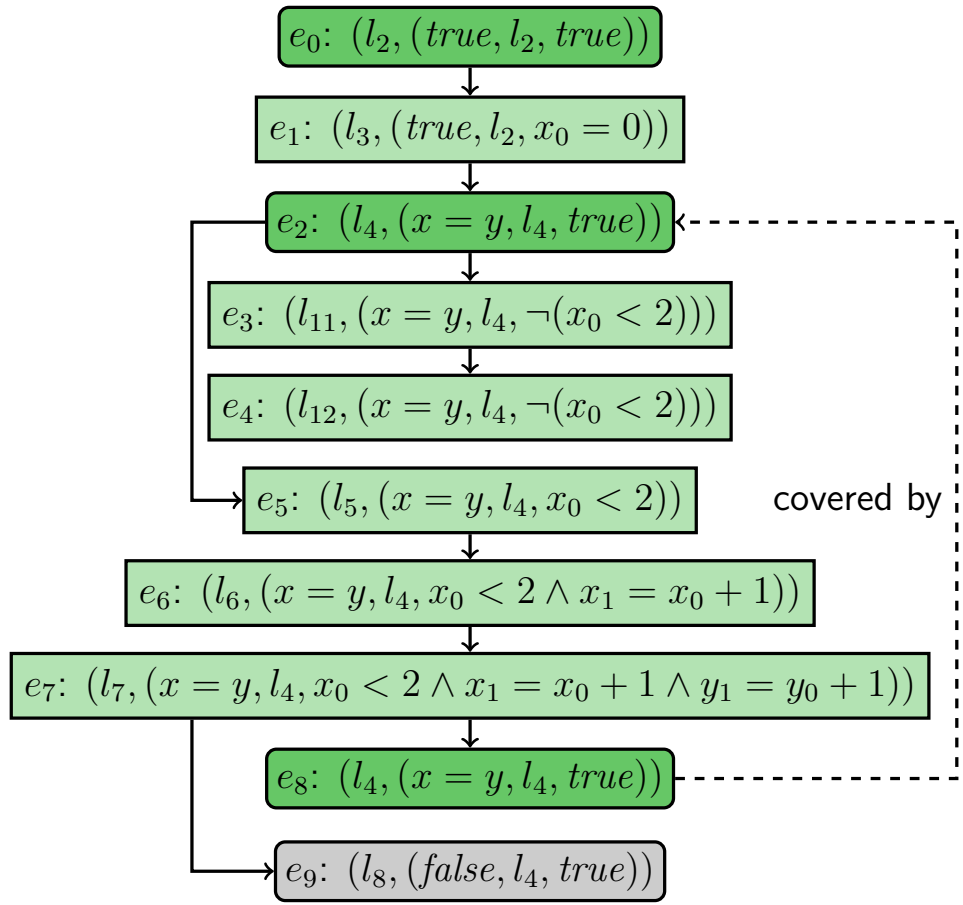




Predicate Abstraction: Example

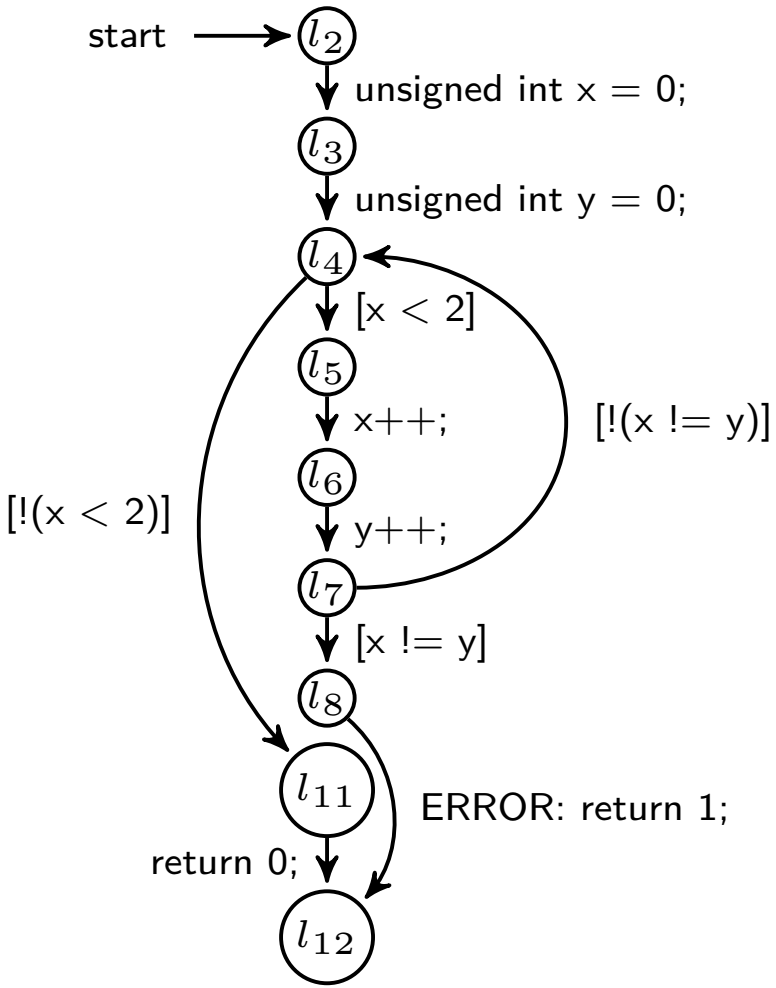
with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

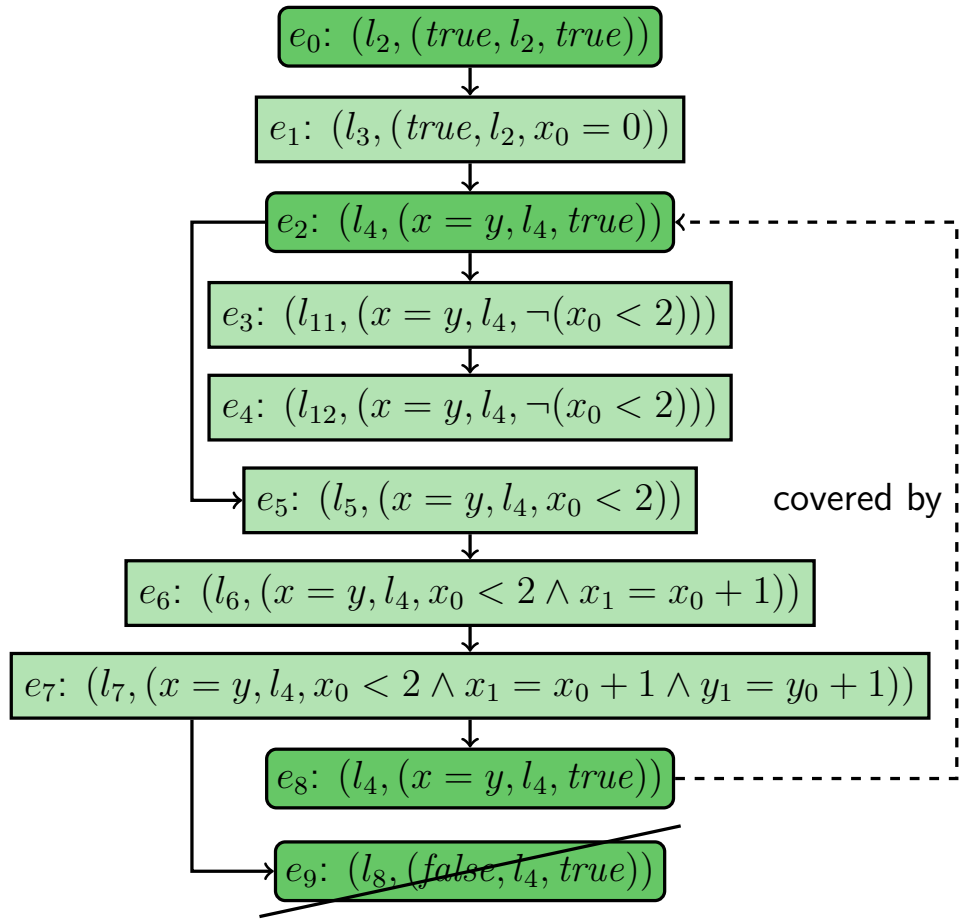




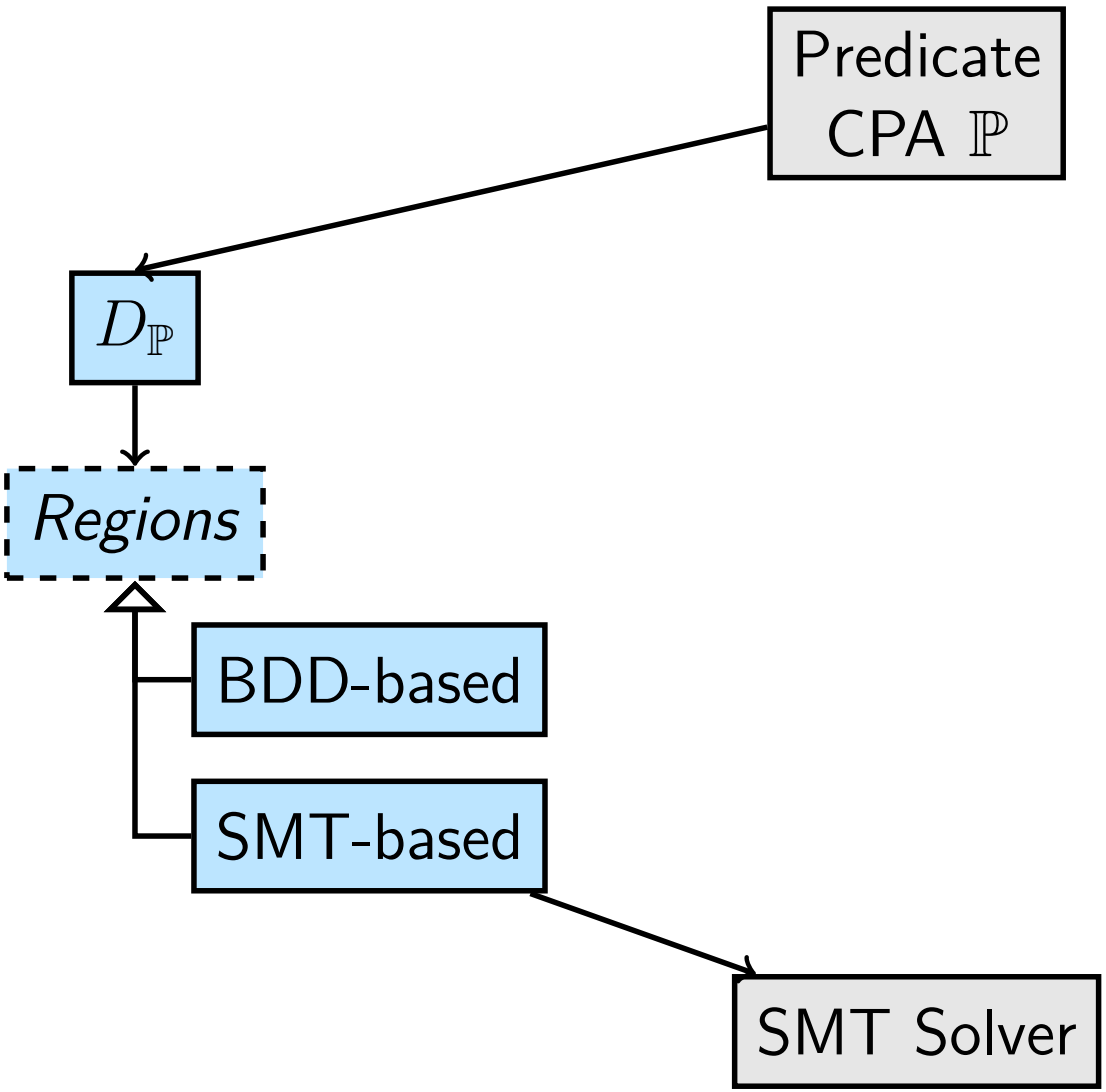
Predicate Abstraction: Example

with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

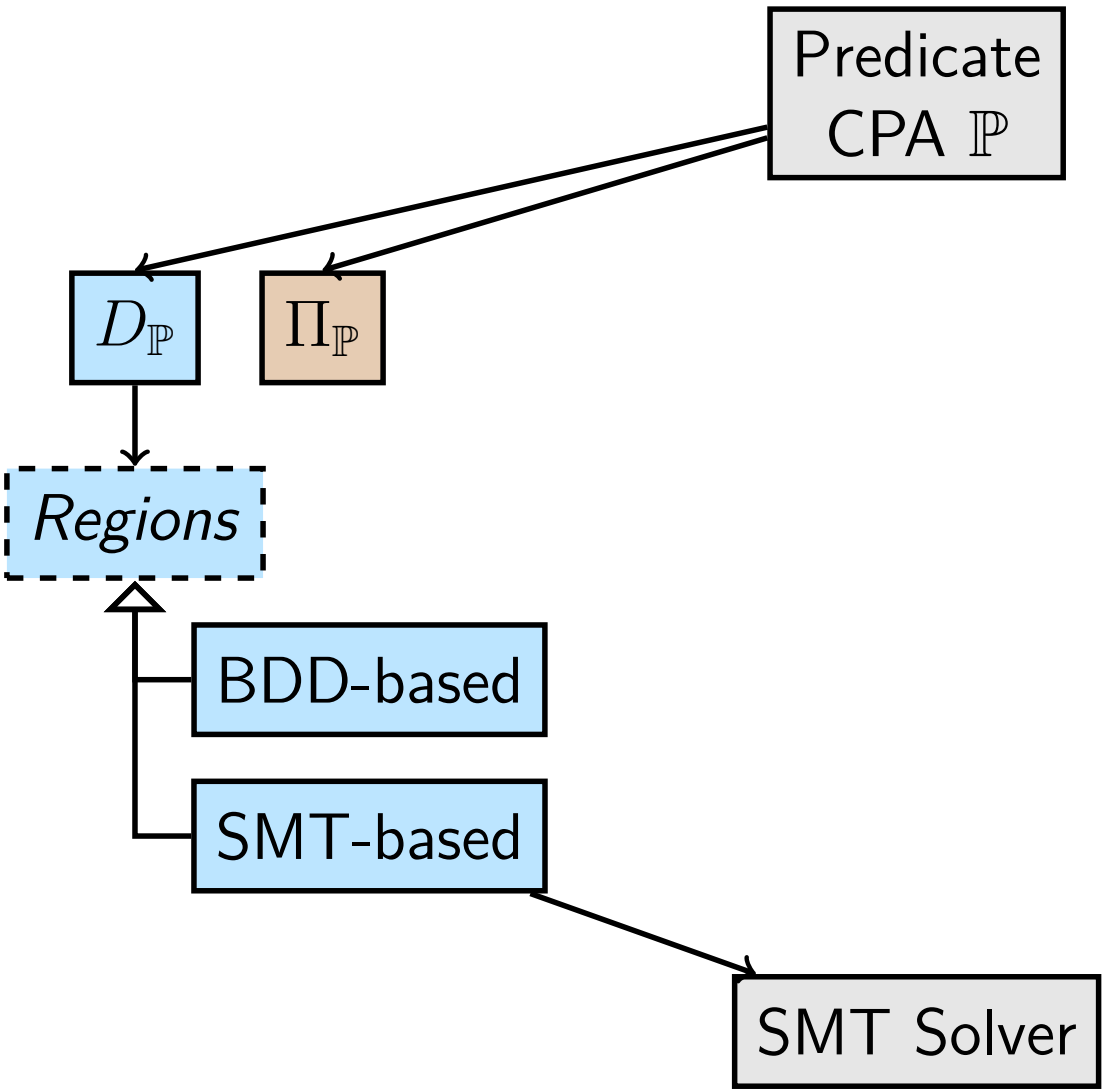




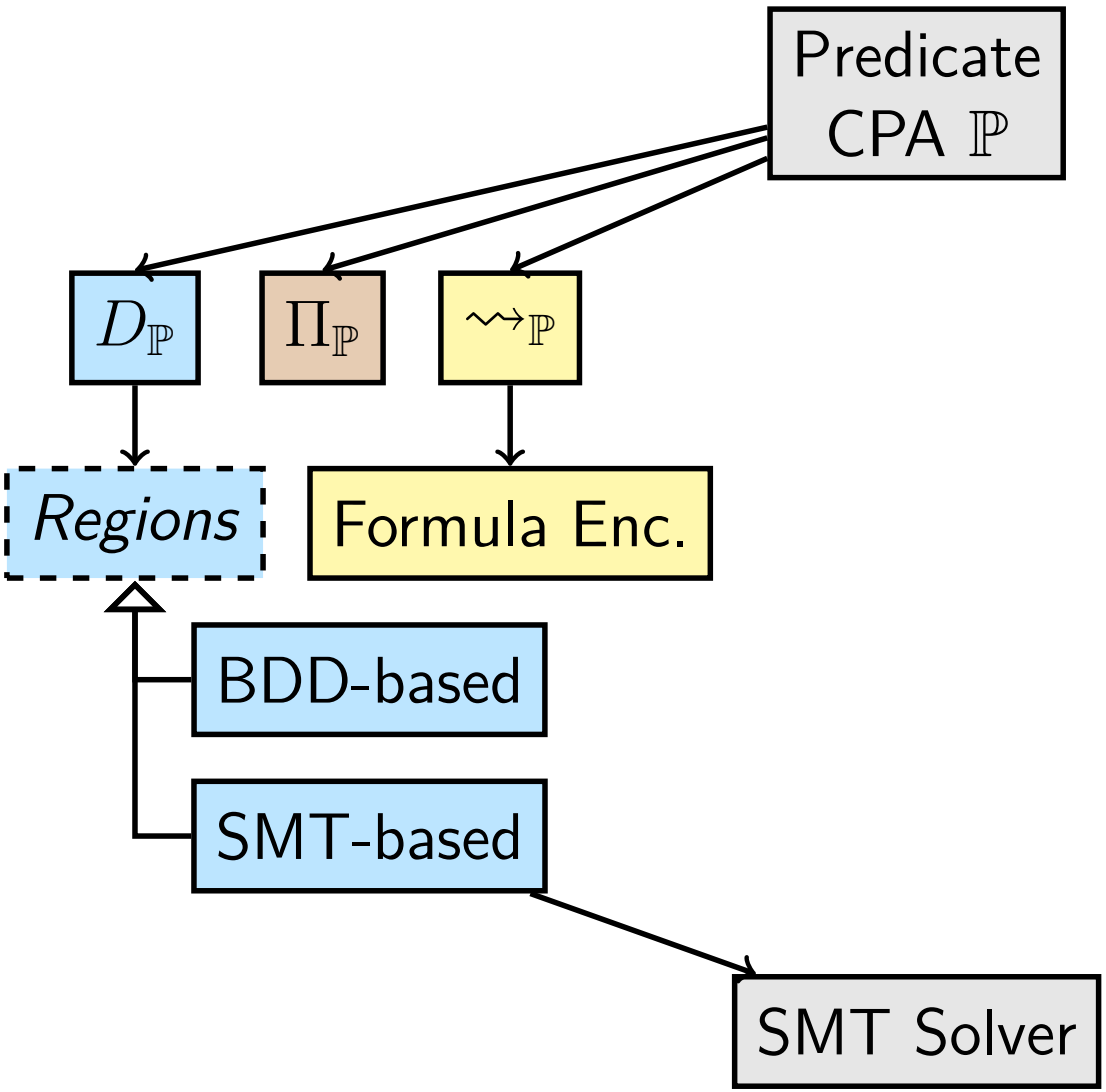
Predicate CPA



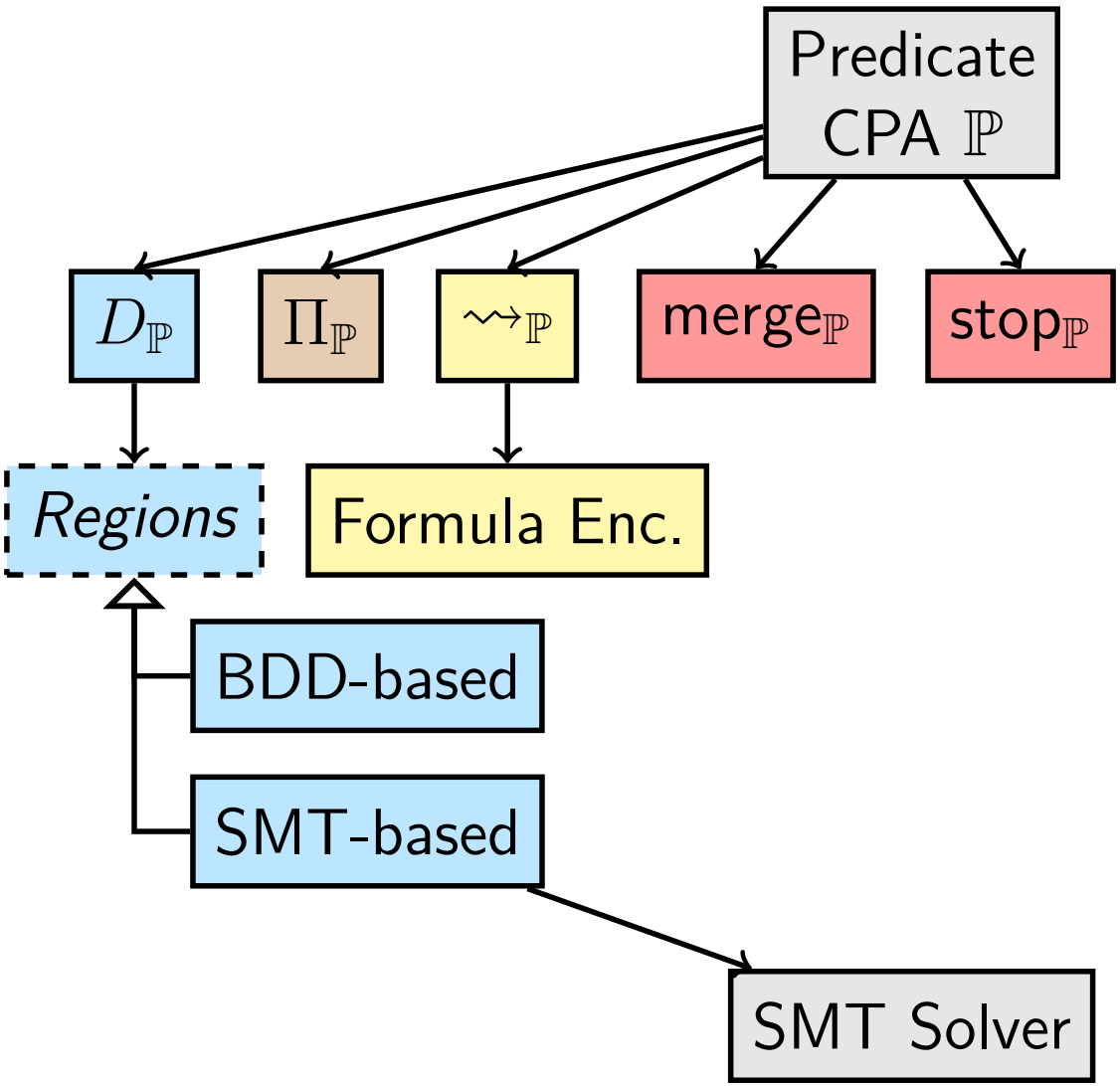
Predicate CPA



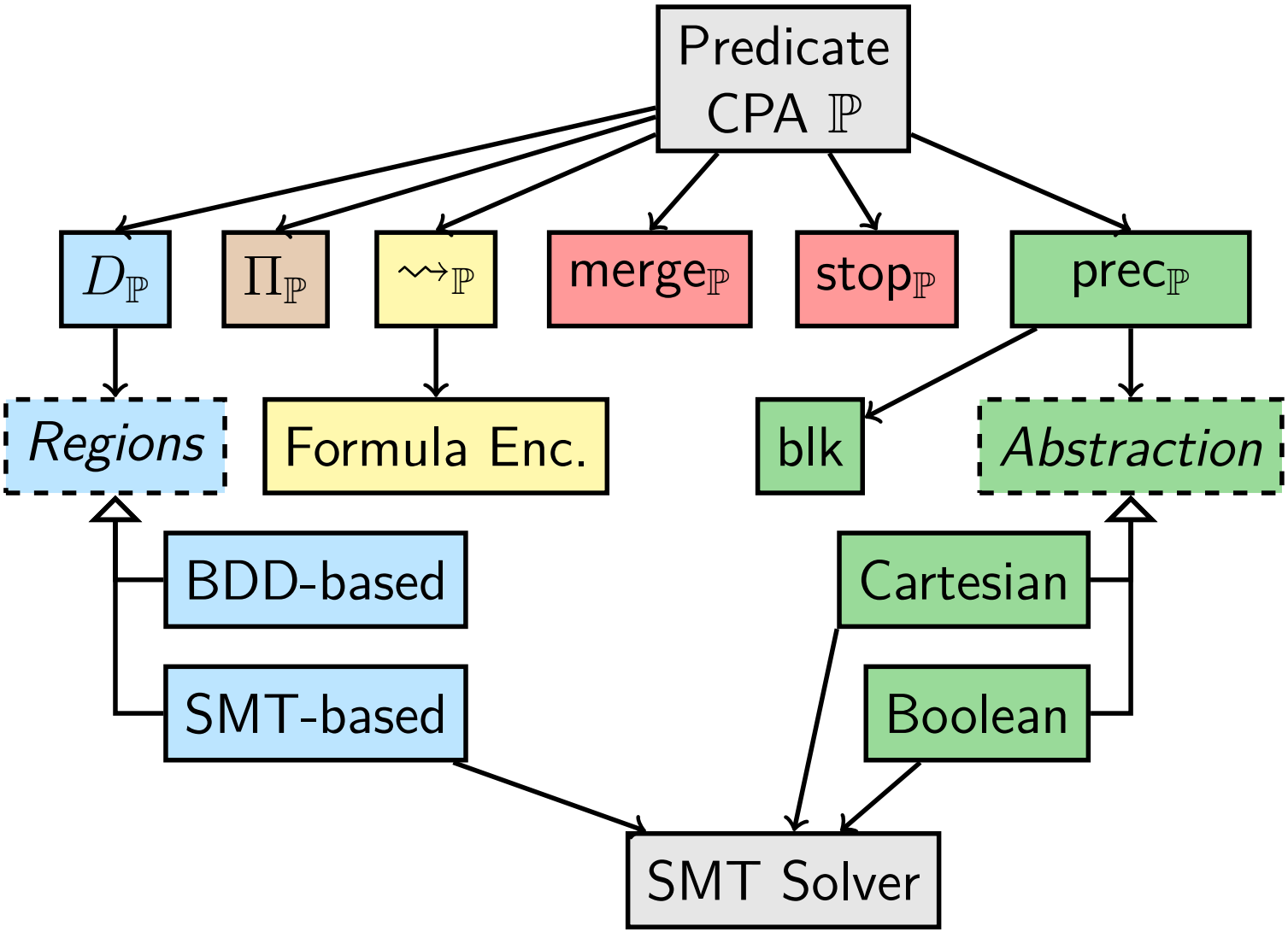
Predicate CPA



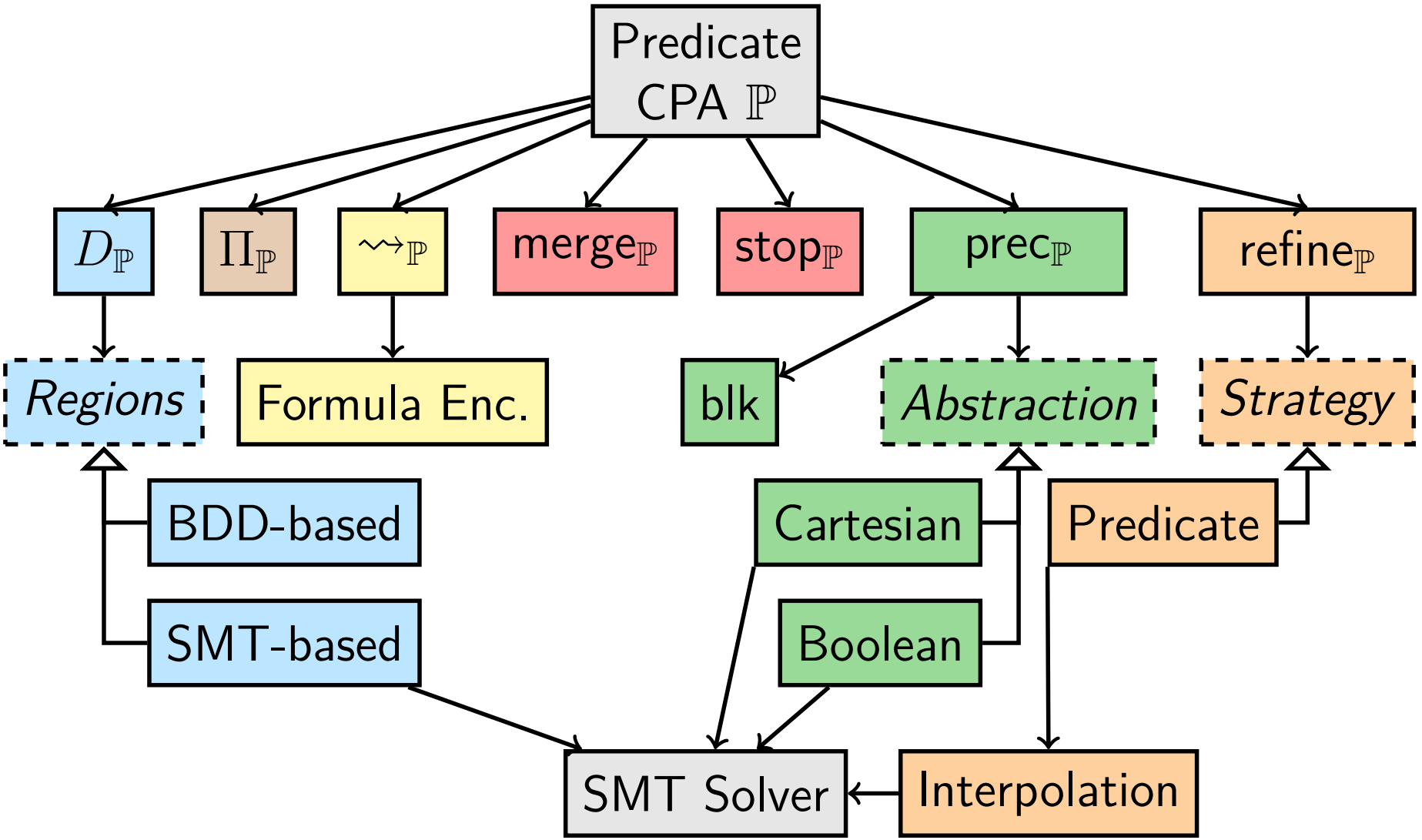
Predicate CPA



Predicate CPA



Predicate CPA

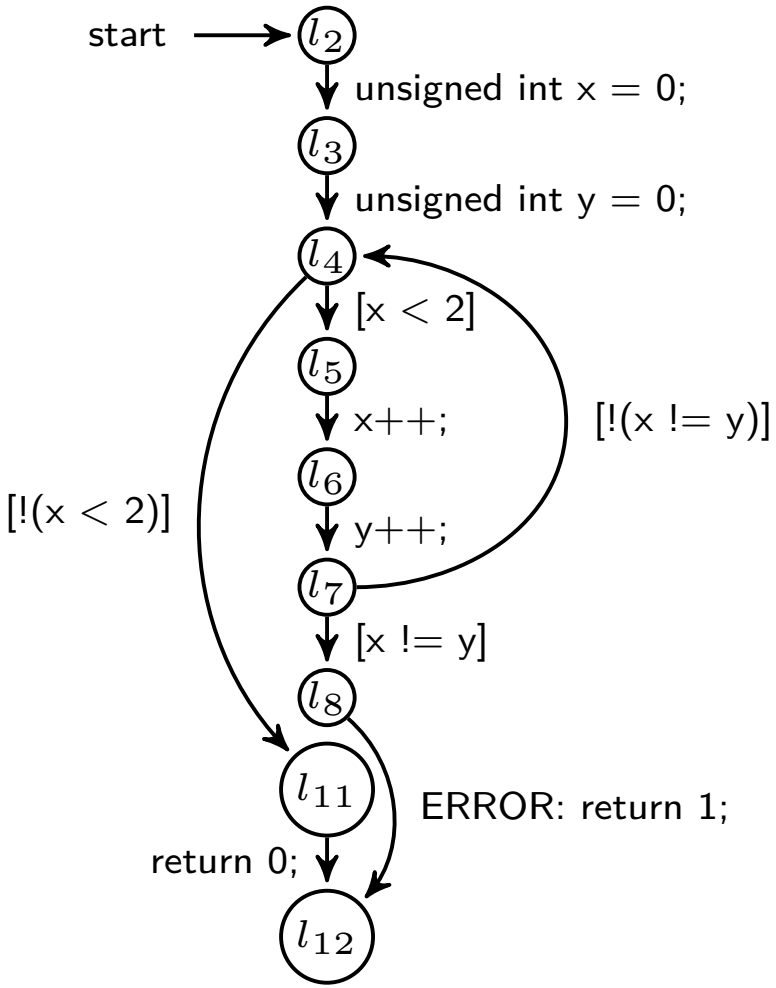


Impact

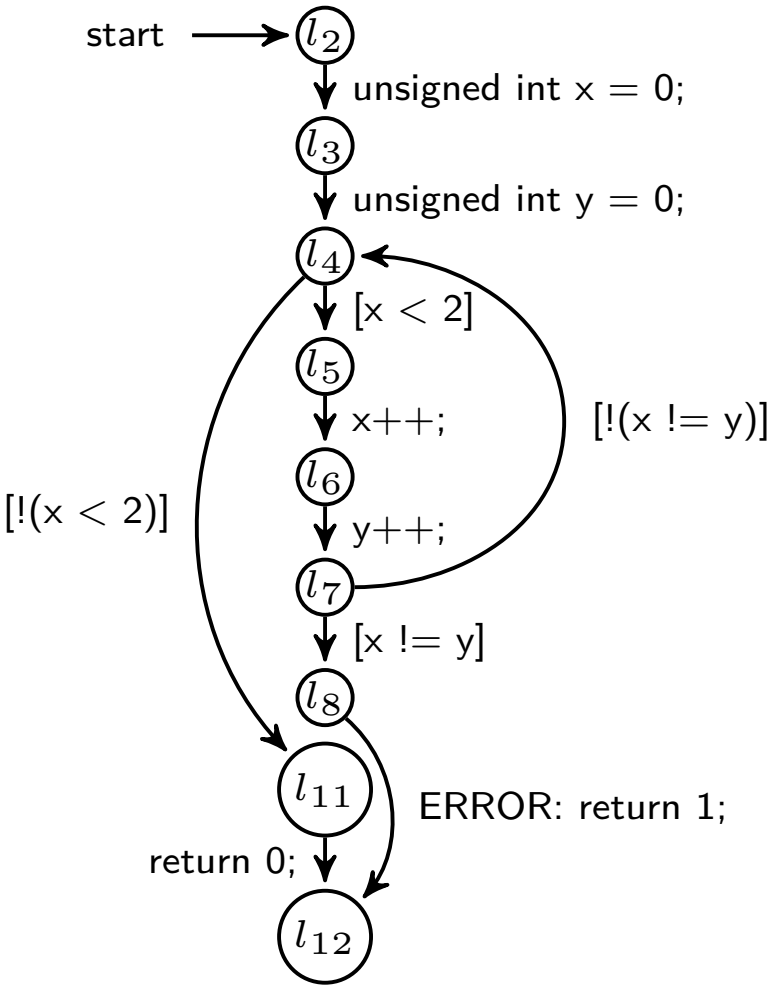
- Impact
  - "Lazy Abstraction with Interpolants"
  - McMillan: [CAV'06]
  - Counter-draft to predicate abstraction
  - Abstraction is derived dynamically/lazily
  - Solution to avoiding expensive abstraction computations
  - Compute fixed point over three operations
    - Expand
    - Refine
    - Cover
  - Quick exploration of the state space
  - Good for finding bugs

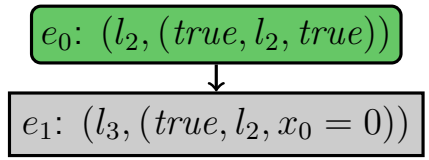
Impact: Example

e_0: $(l_2, (true, l_2, true))$

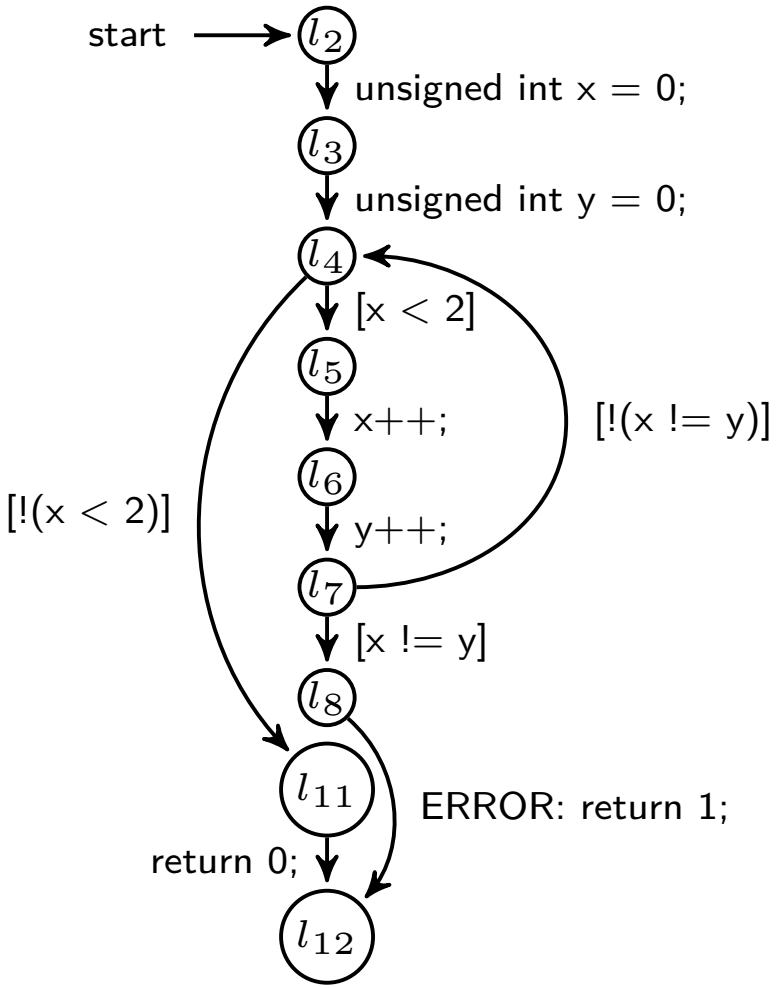


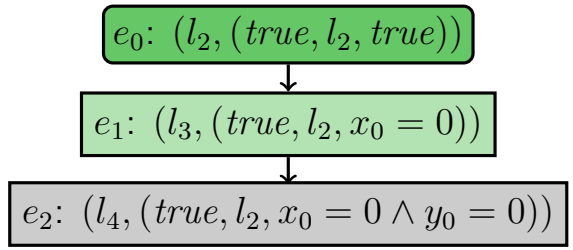
IMPACT: Example



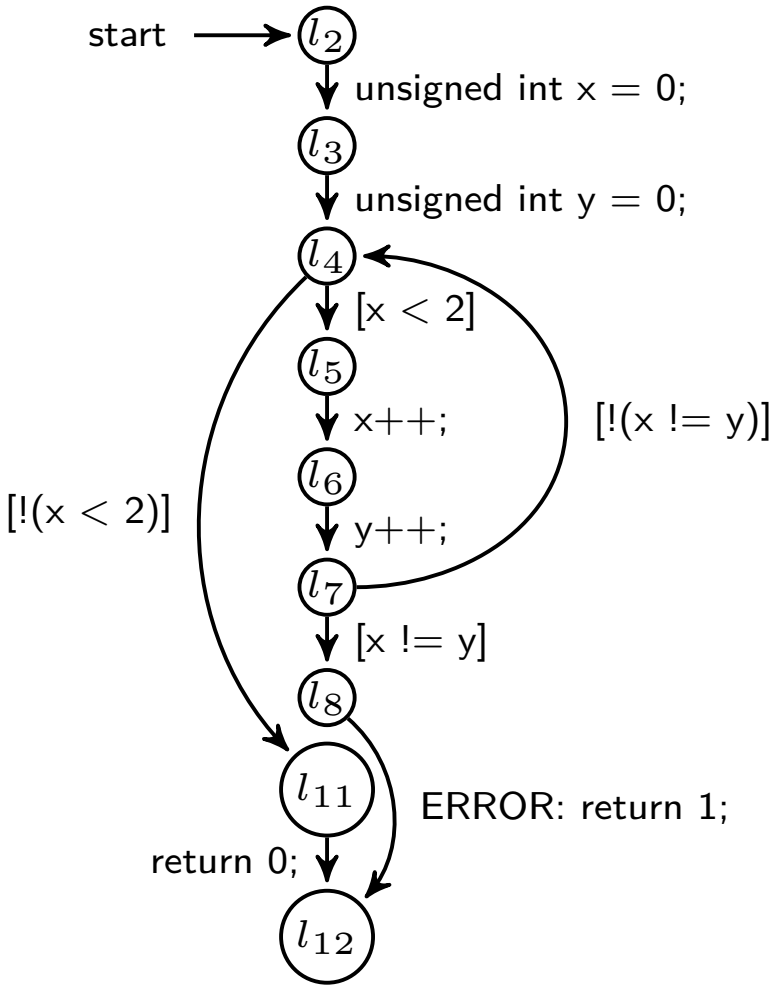


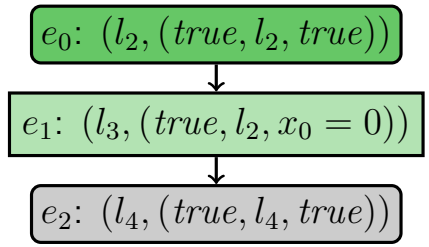
IMPACT: Example



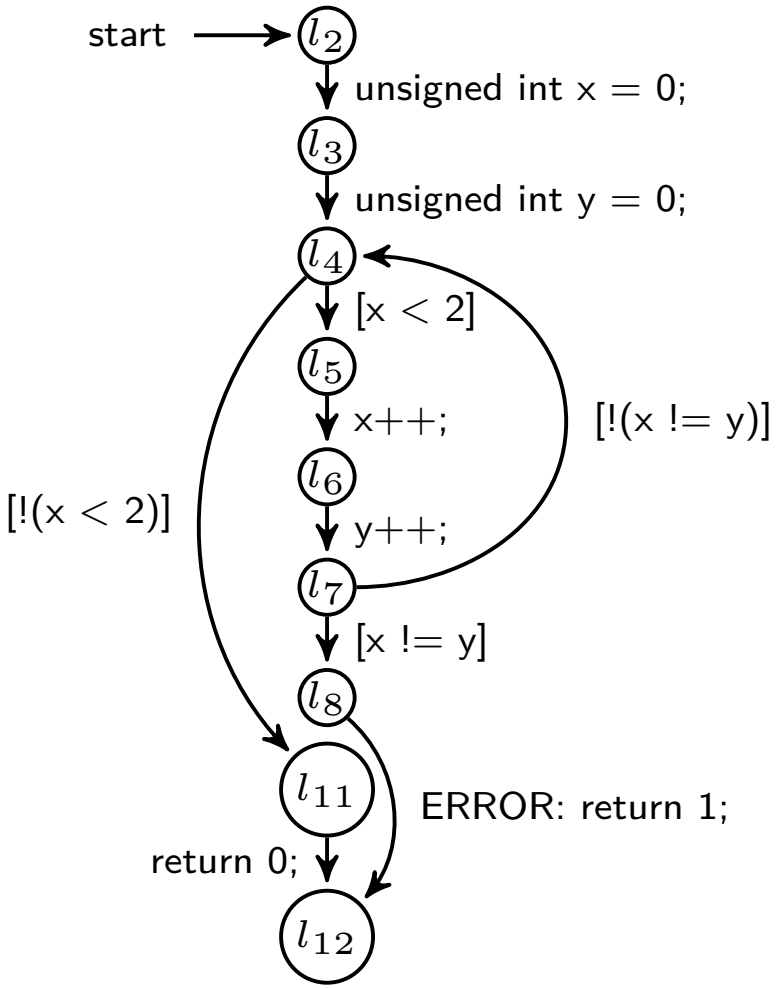


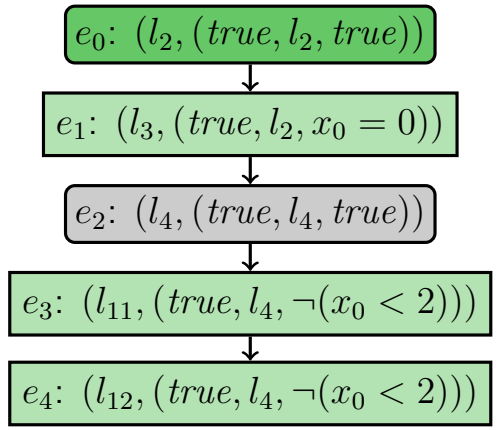
Impact: Example



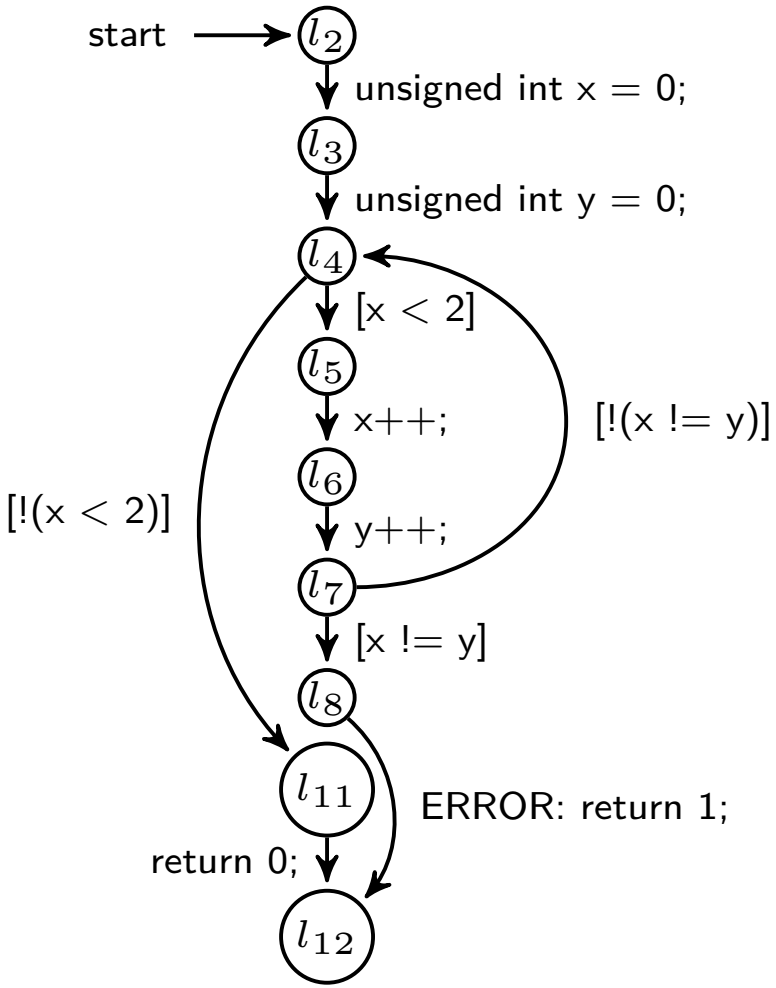


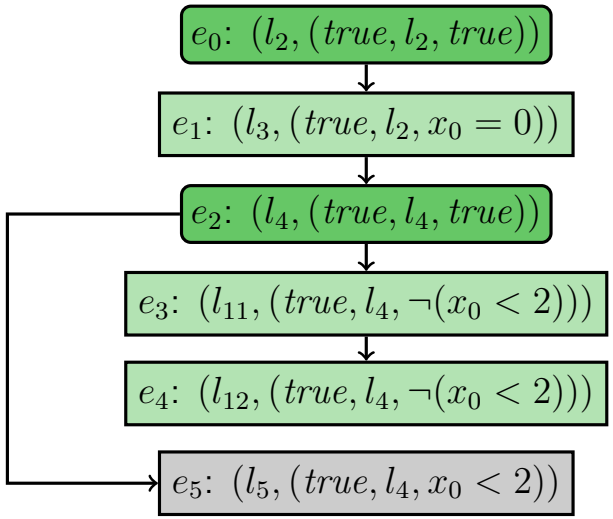
IMPACT: Example



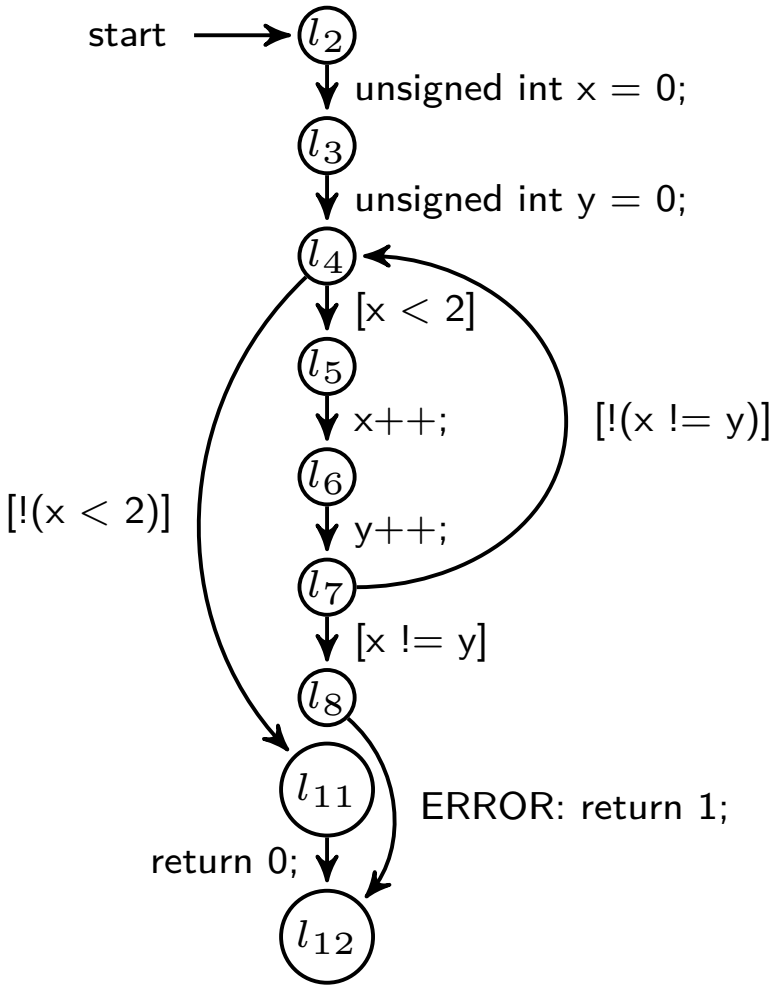


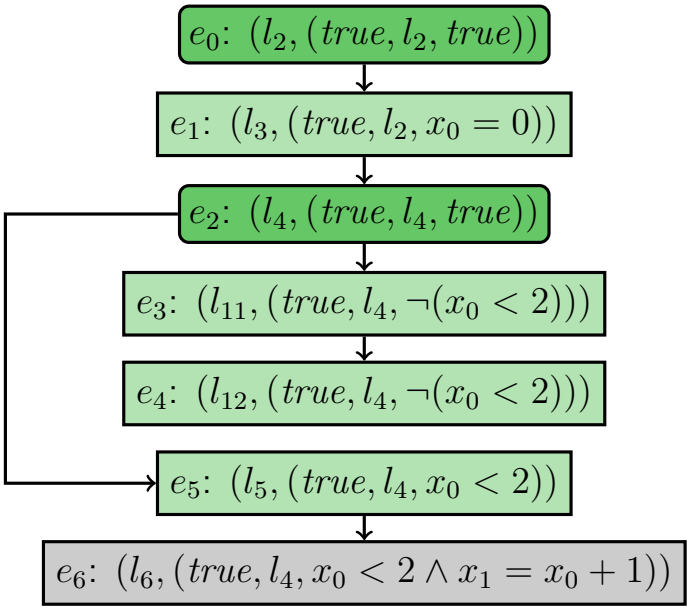
IMPACT: Example



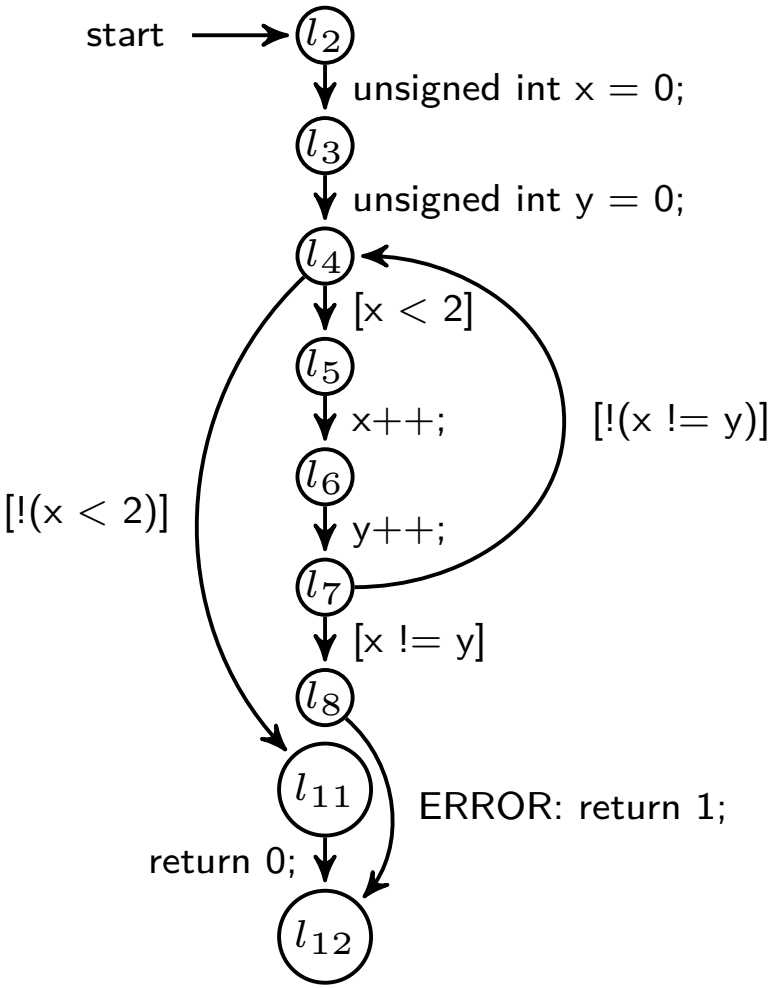


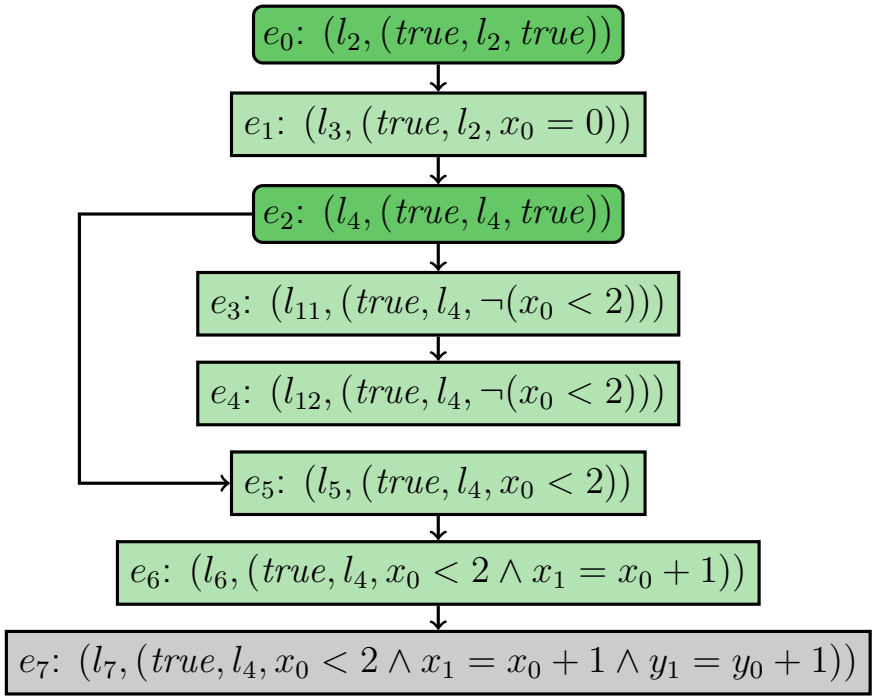
IMPACT: Example



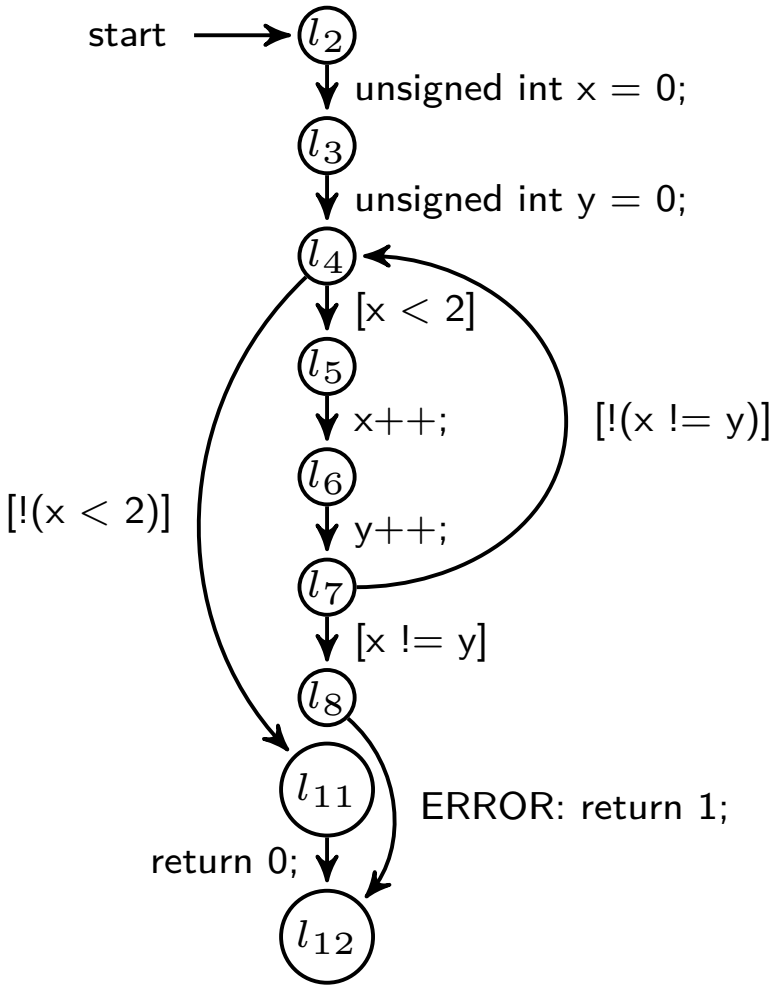


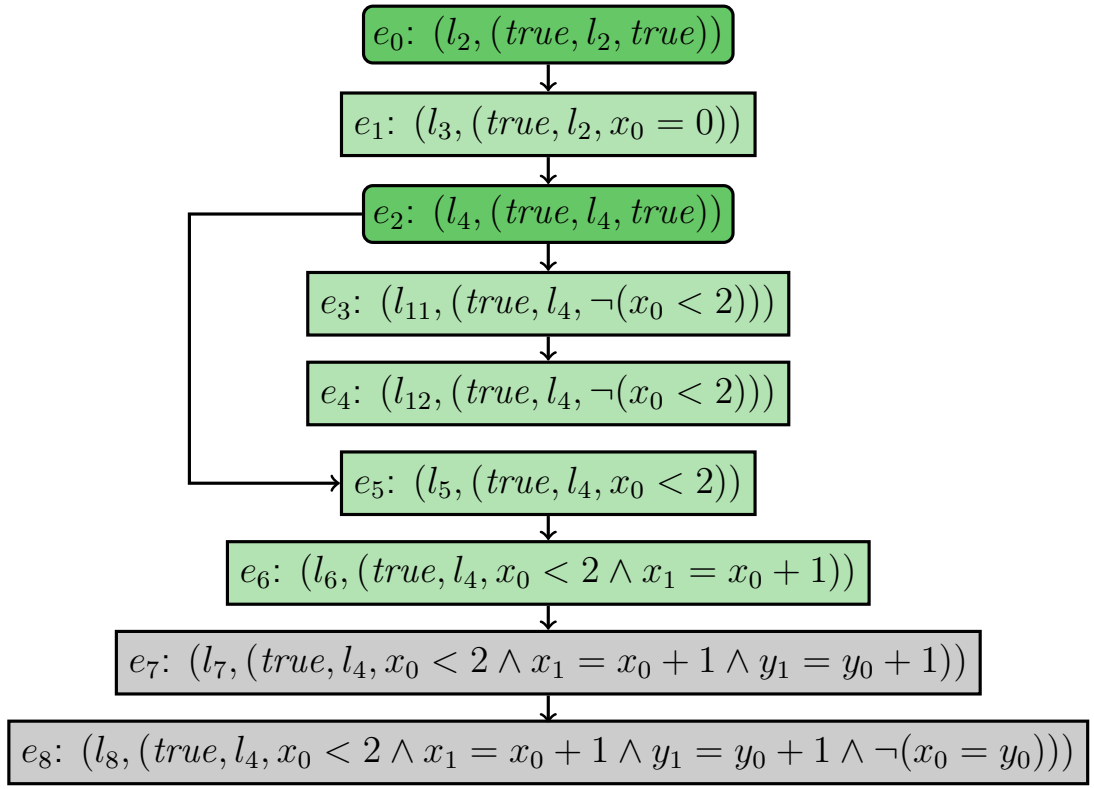
IMPACT: Example



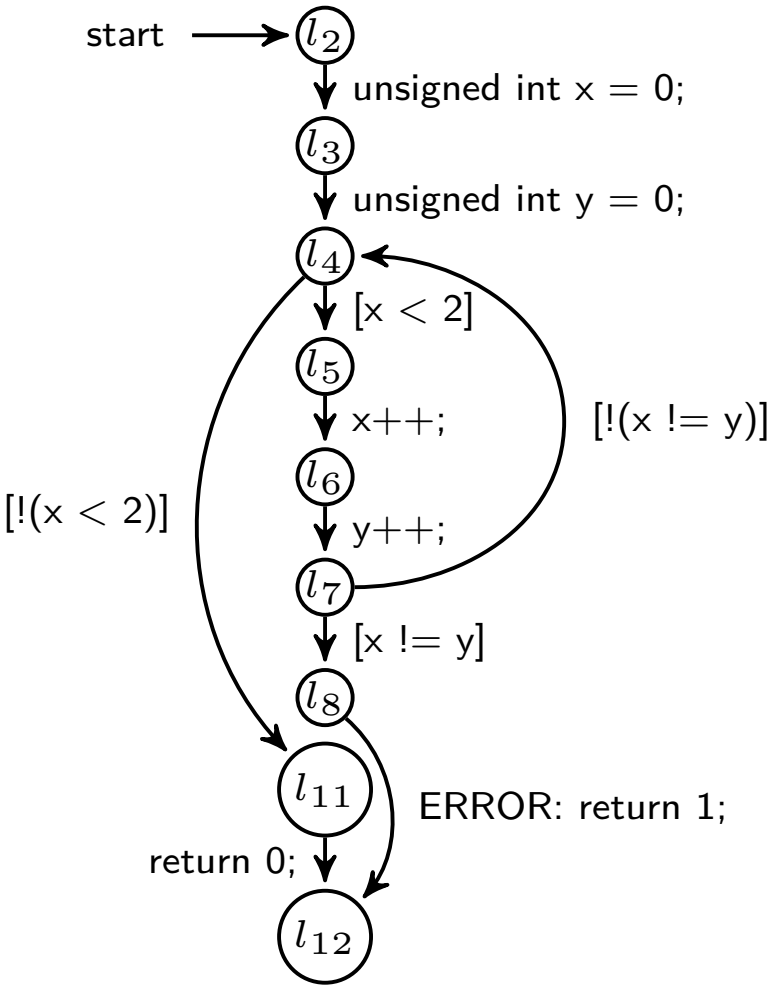


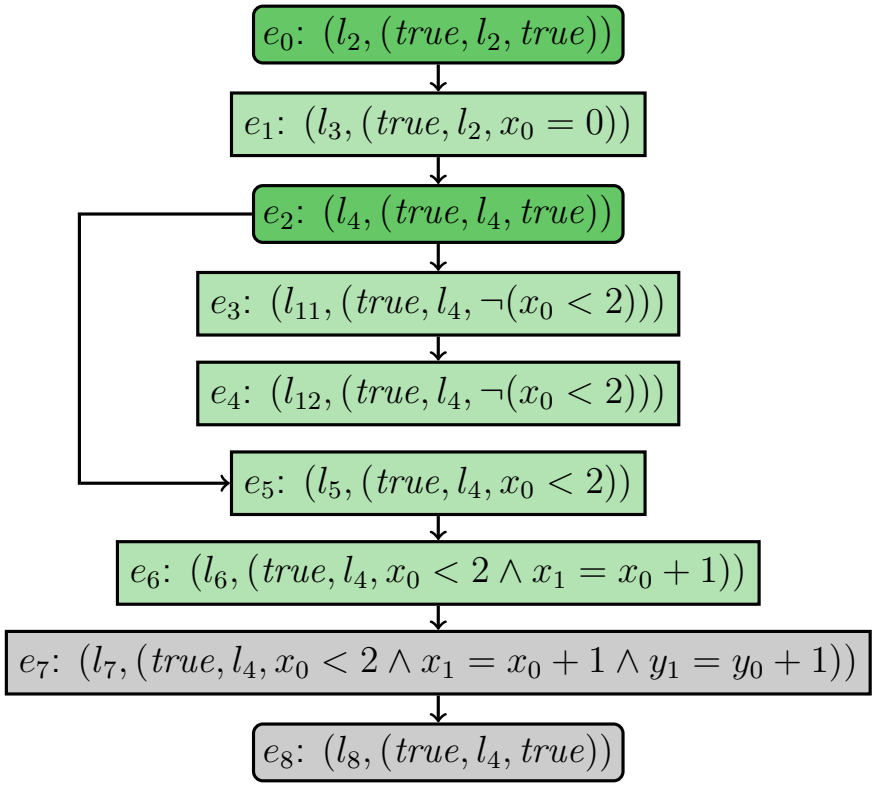
IMPACT: Example



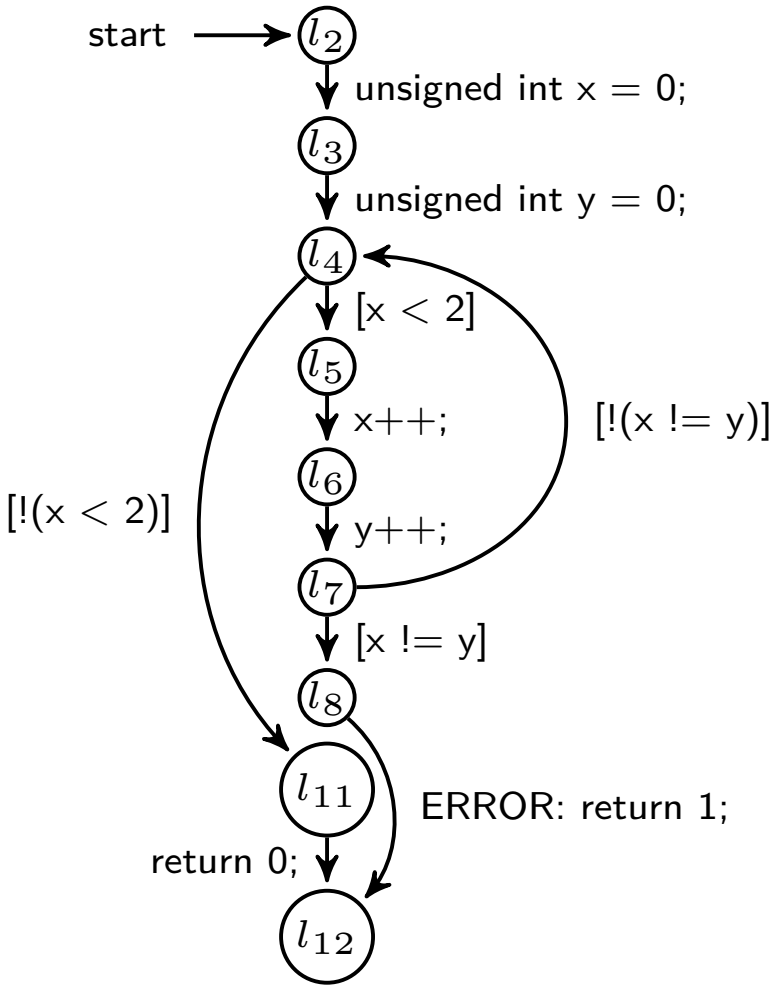


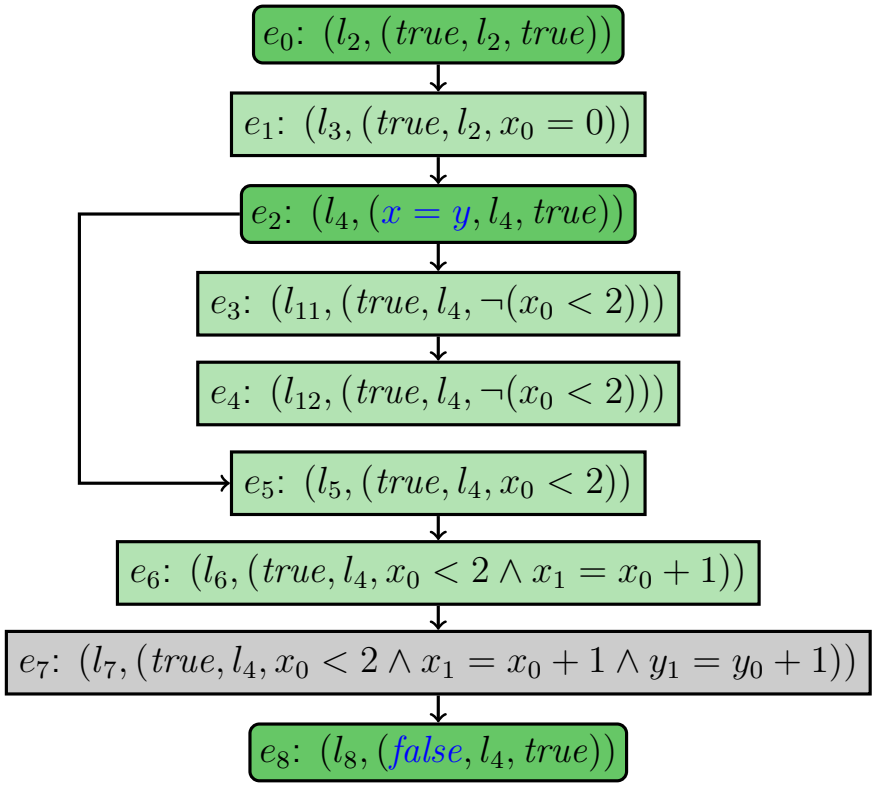
IMPACT: Example



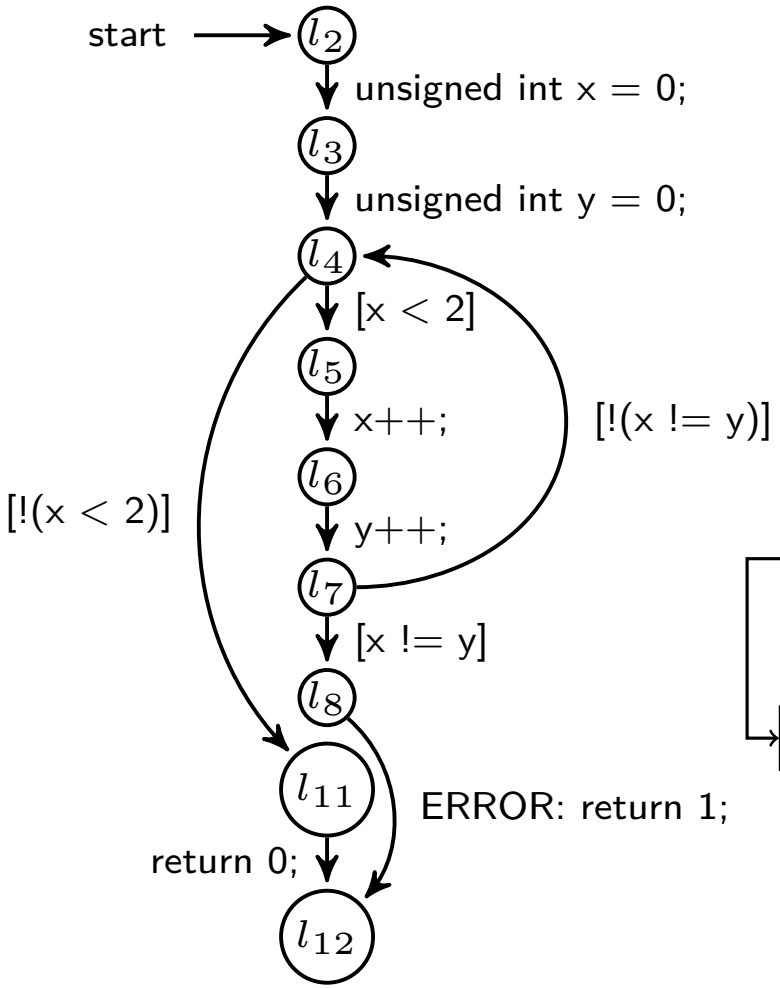


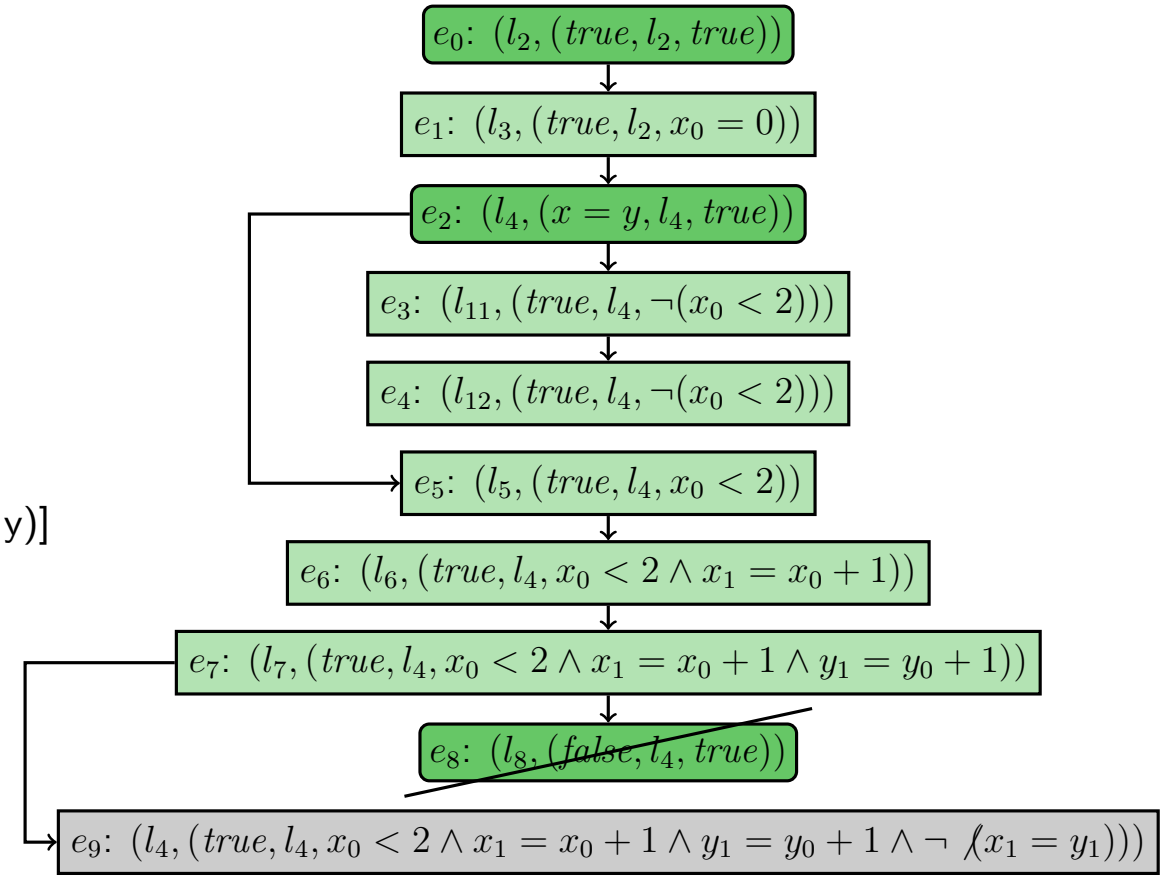
IMPACT: Example



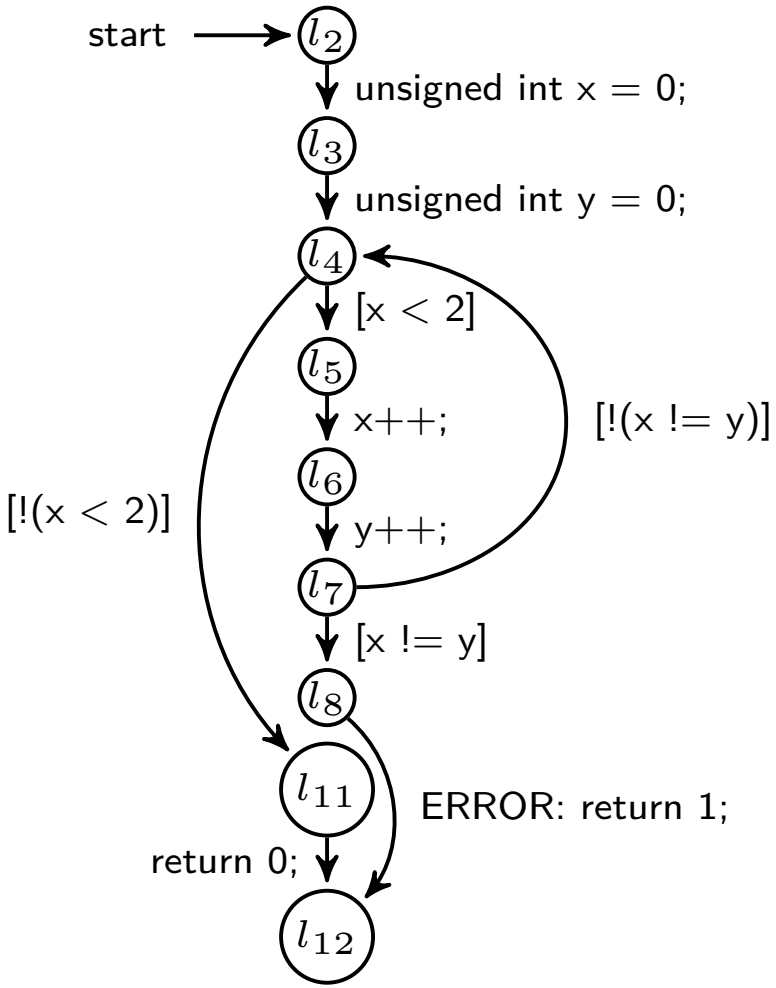


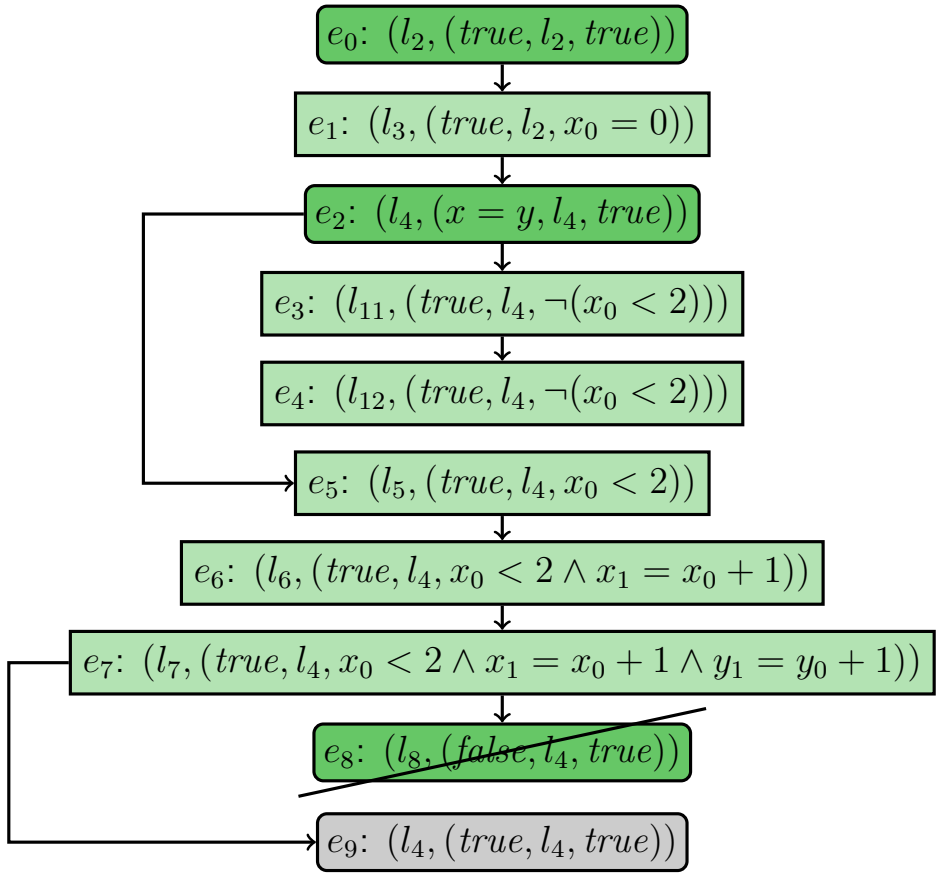
IMPACT: Example



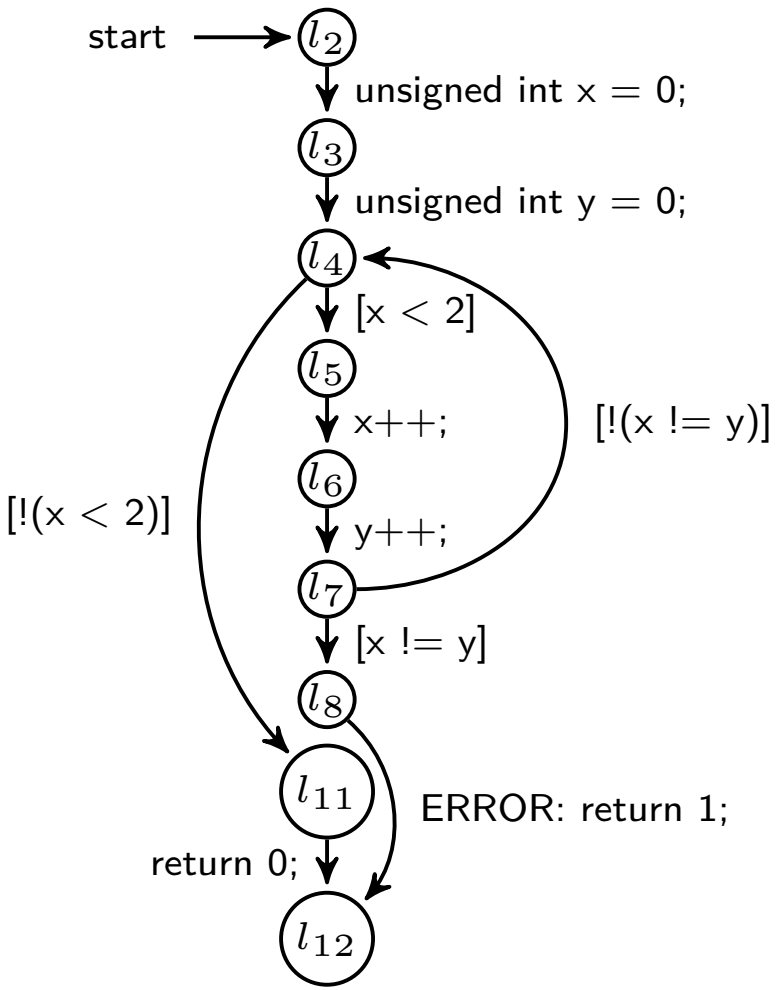


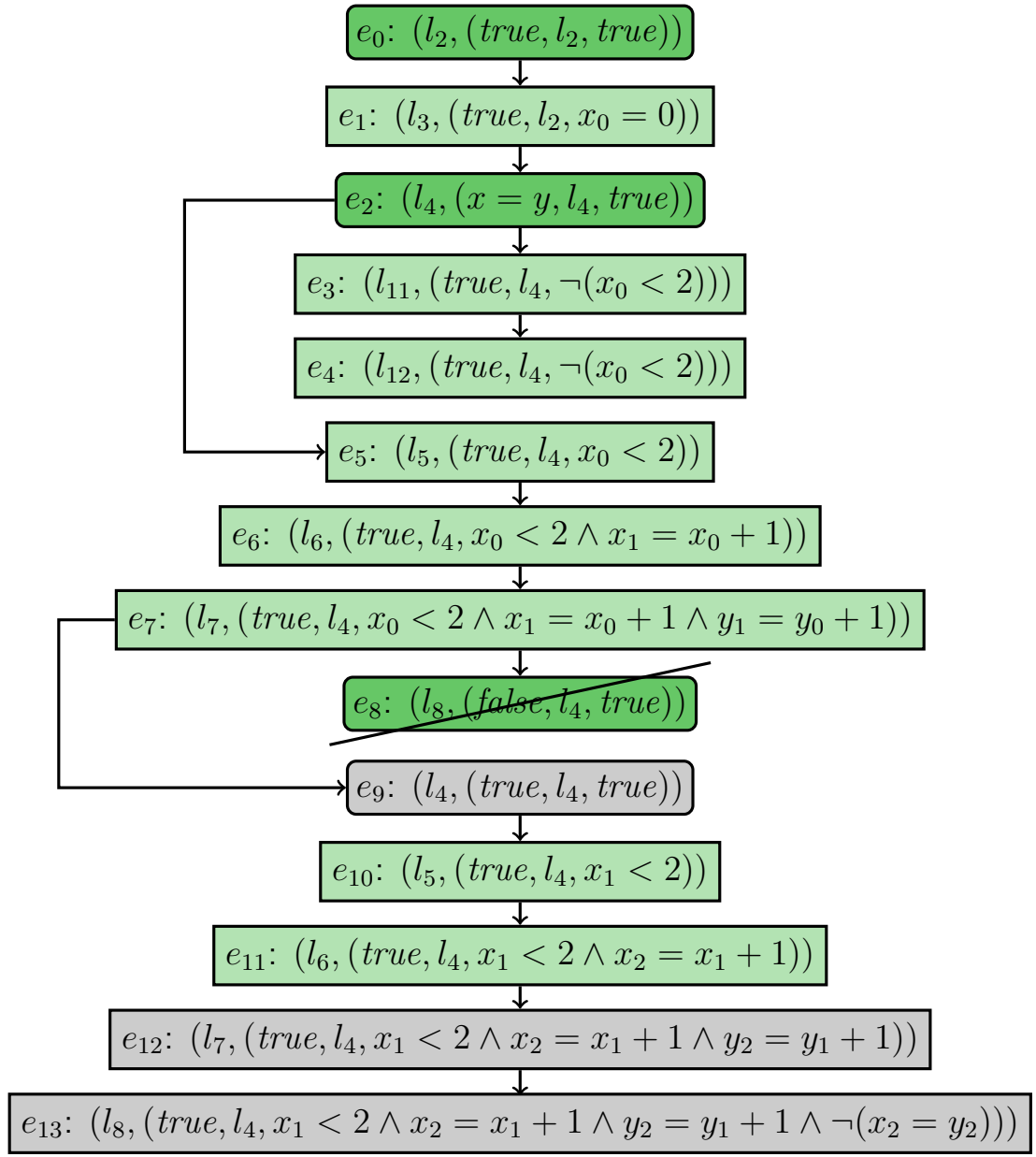
IMPACT: Example



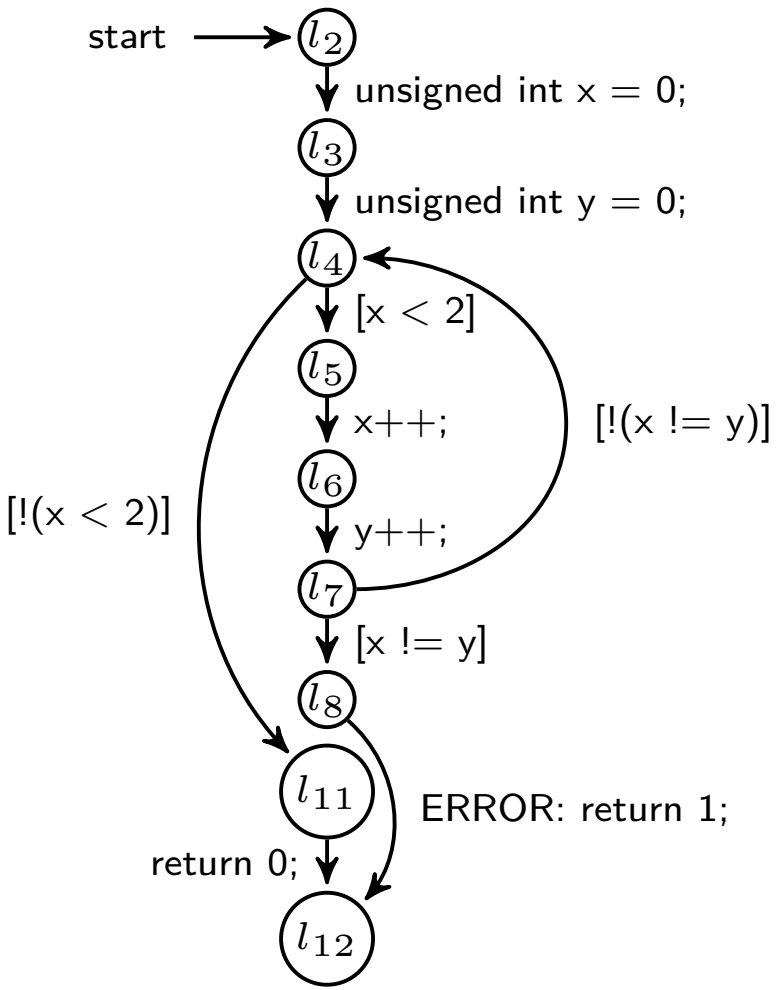


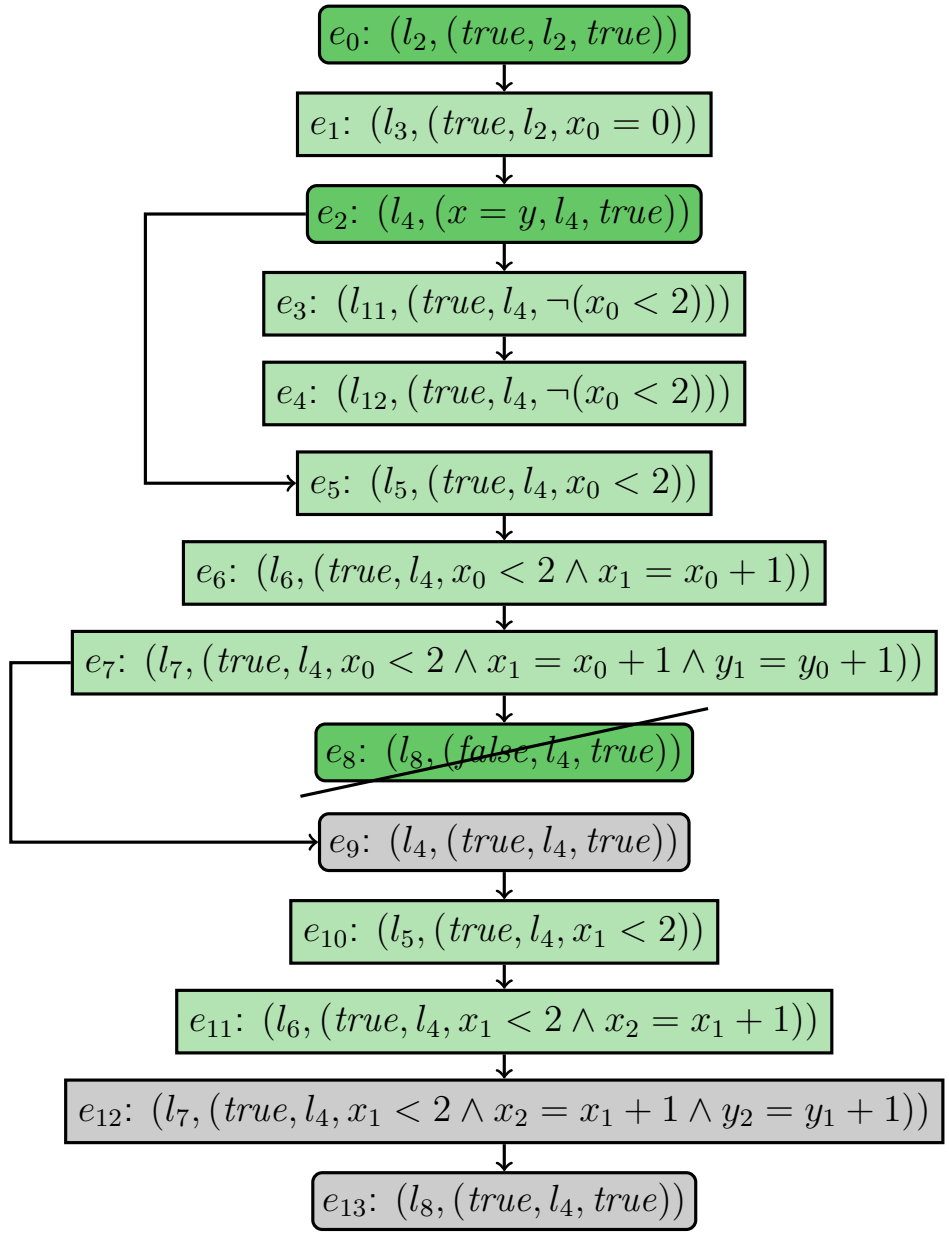
IMPACT: Example



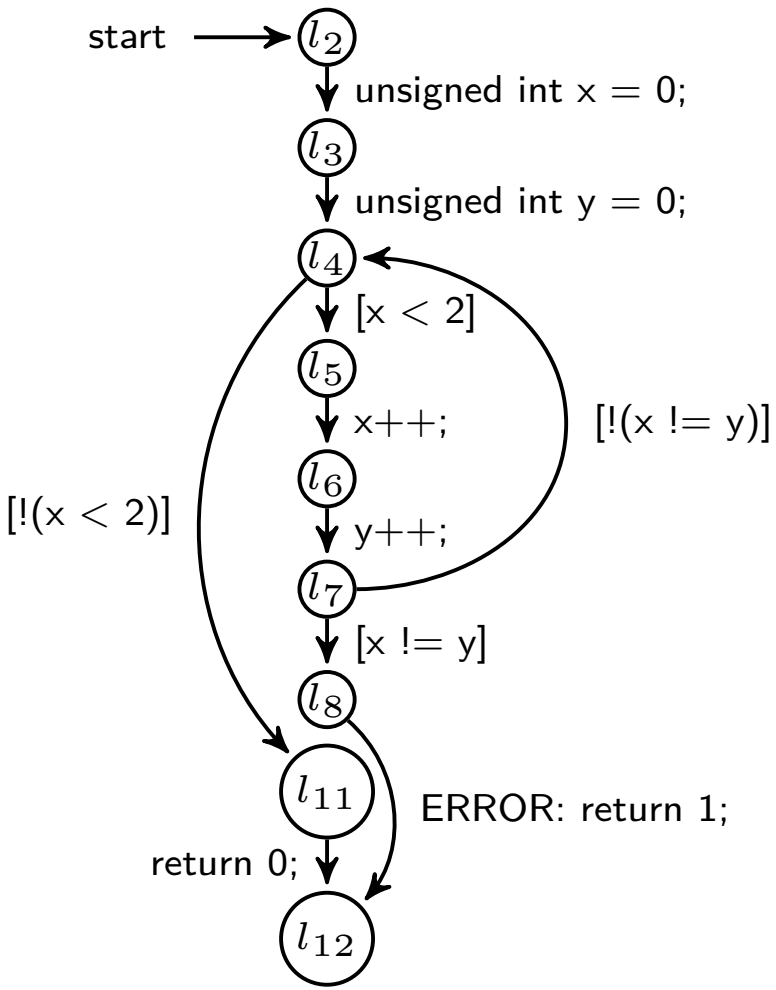


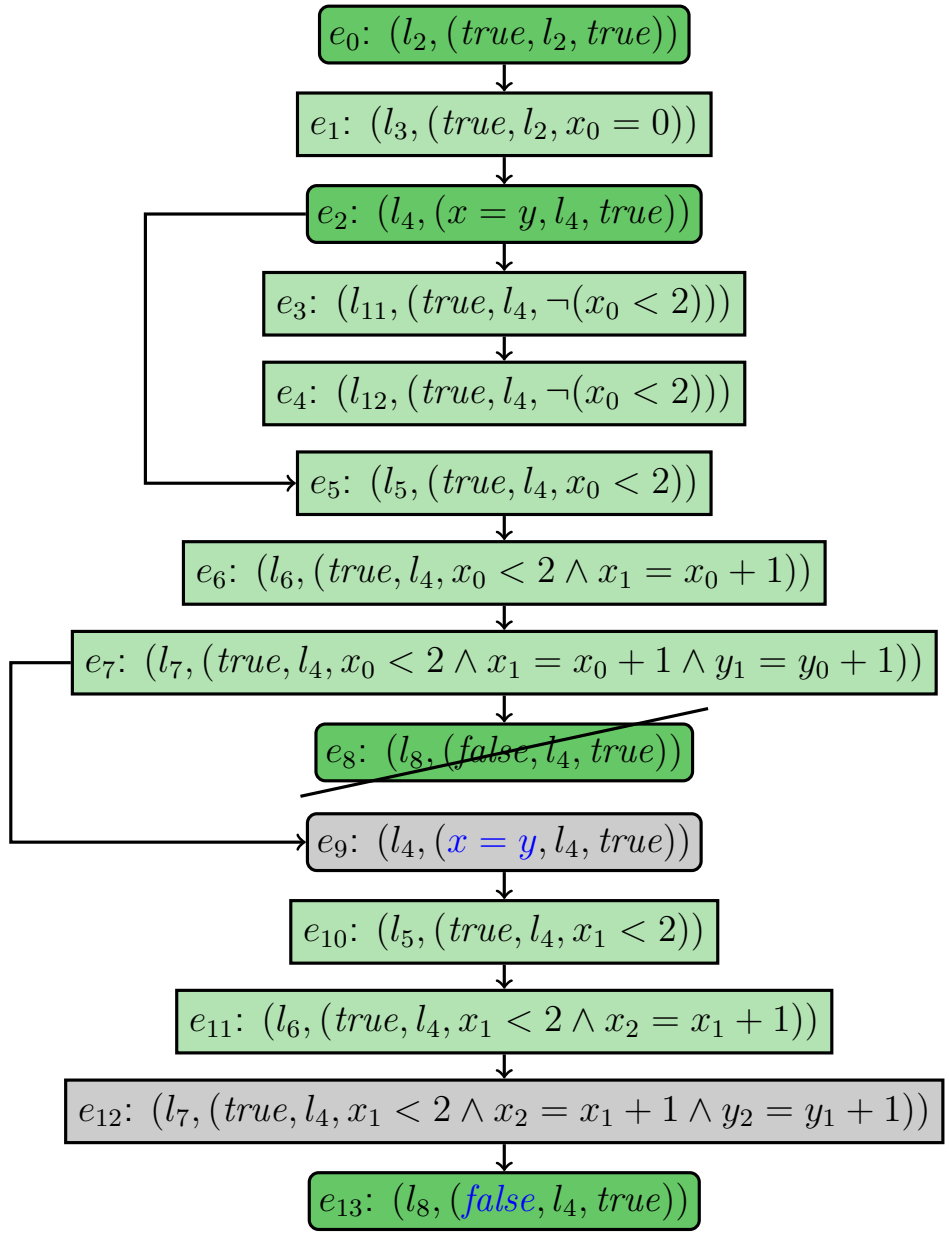
IMPACT: Example



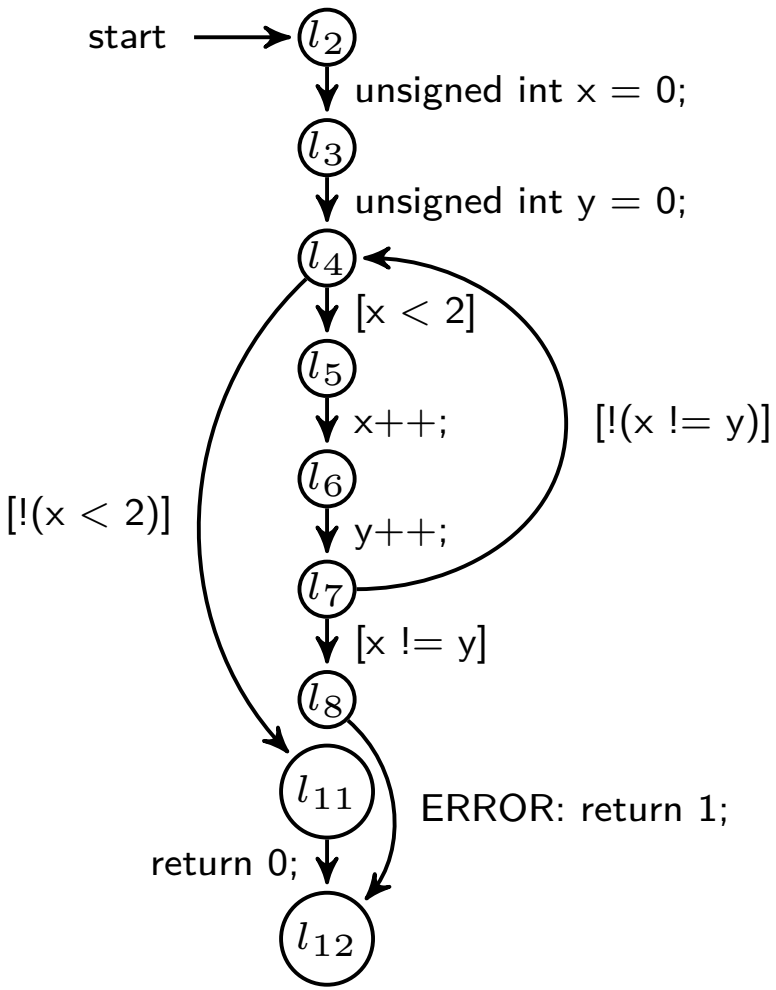


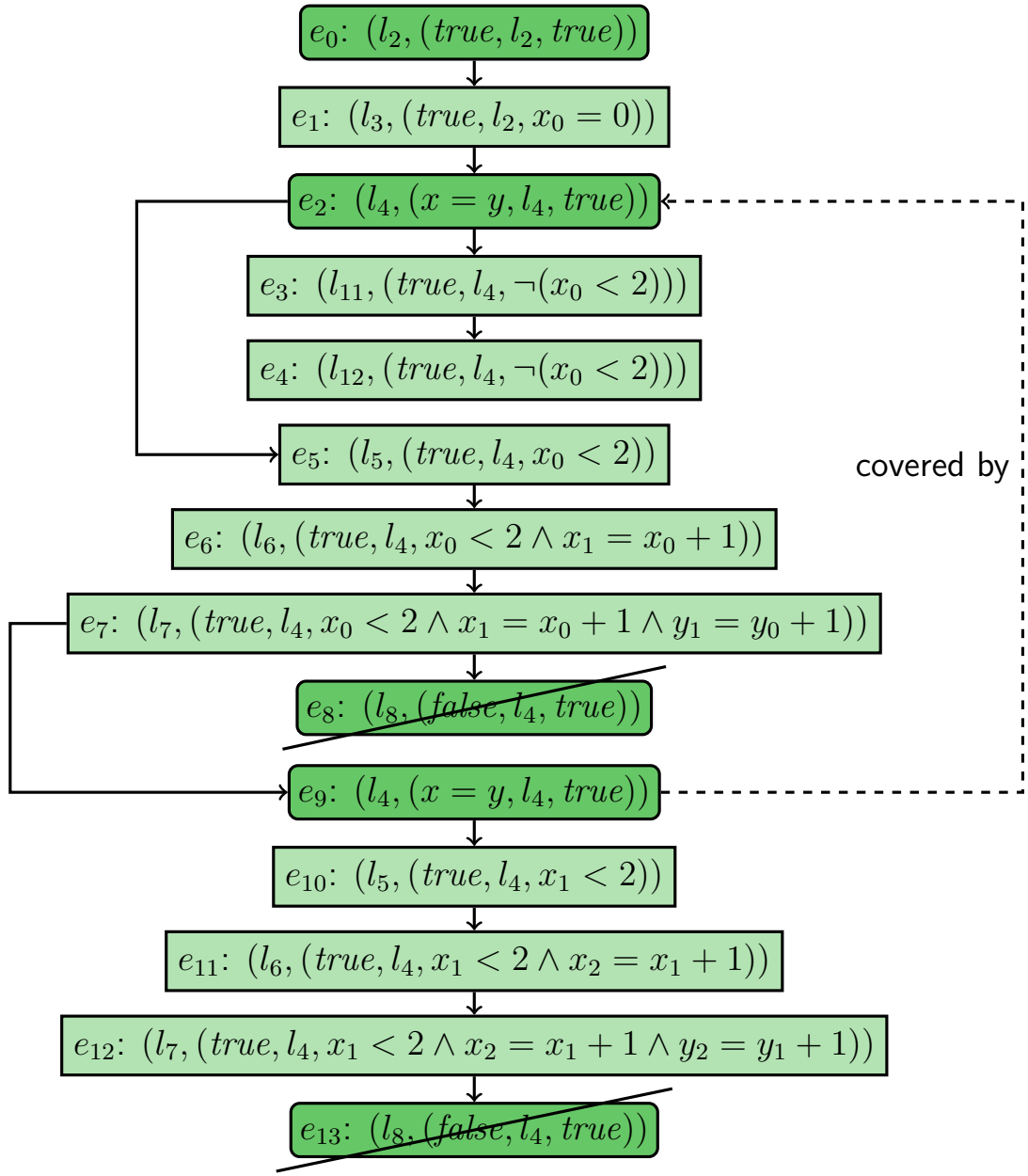
IMPACT: Example



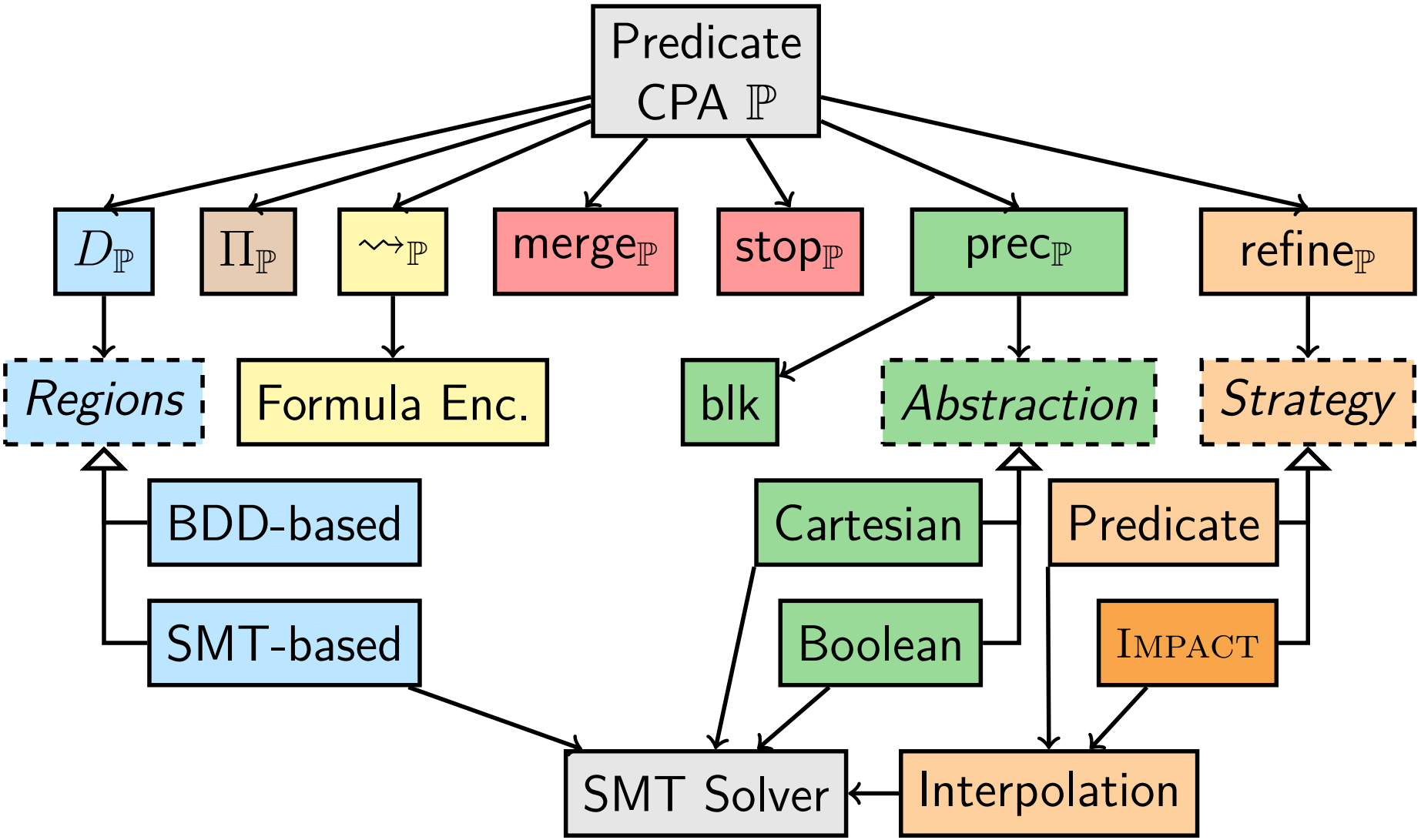


IMPACT: Example





Predicate CPA extended for IMPACT



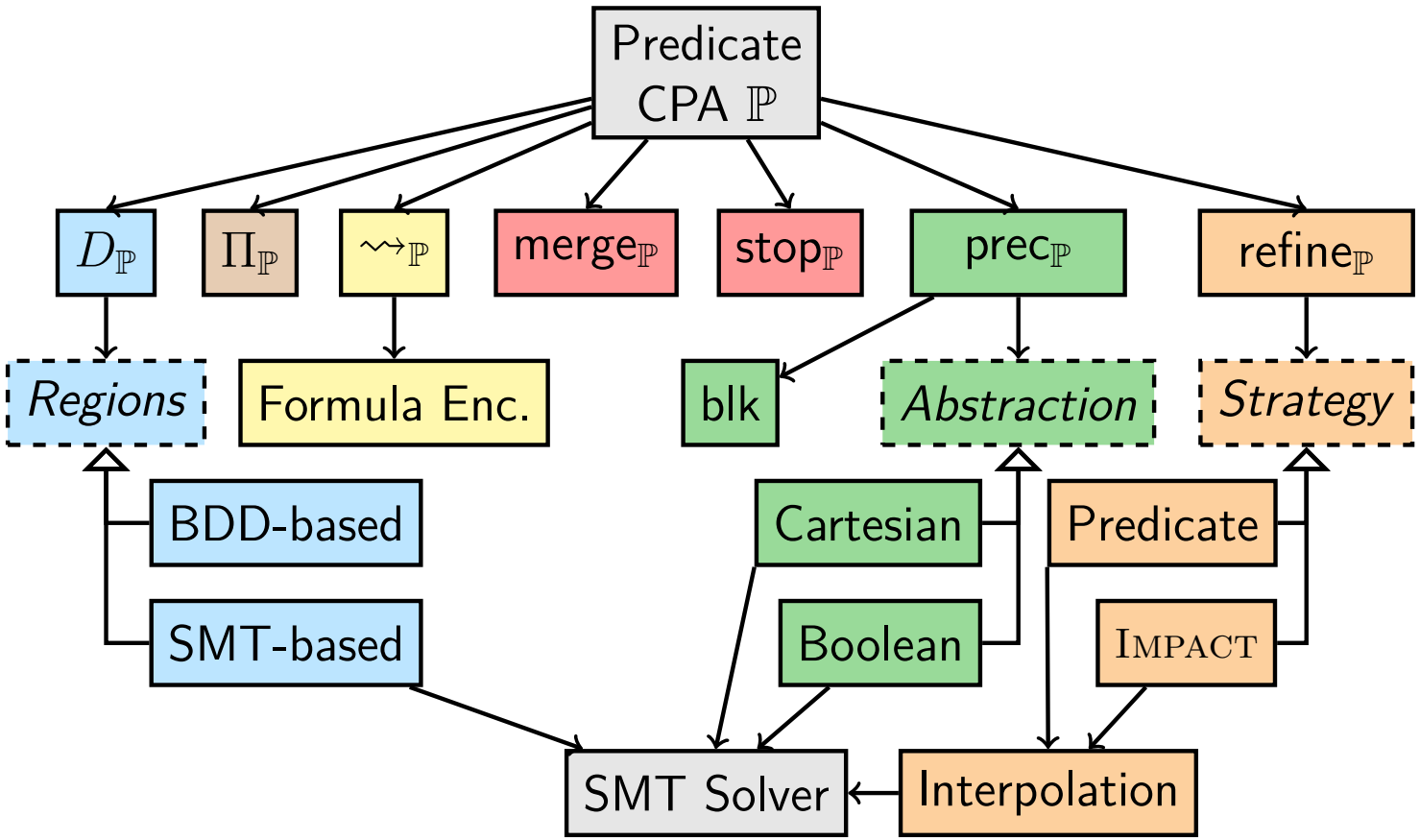
Bounded Model Checking

- Bounded Model Checking:
  - Biere, Cimatti, Clarke, Zhu: [TACAS'99]
  - No abstraction
  - Unroll loops up to a loop bound k
  - Check that P holds in the first k iterations:

$$\bigwedge_{i=1}^{k} P(i)$$

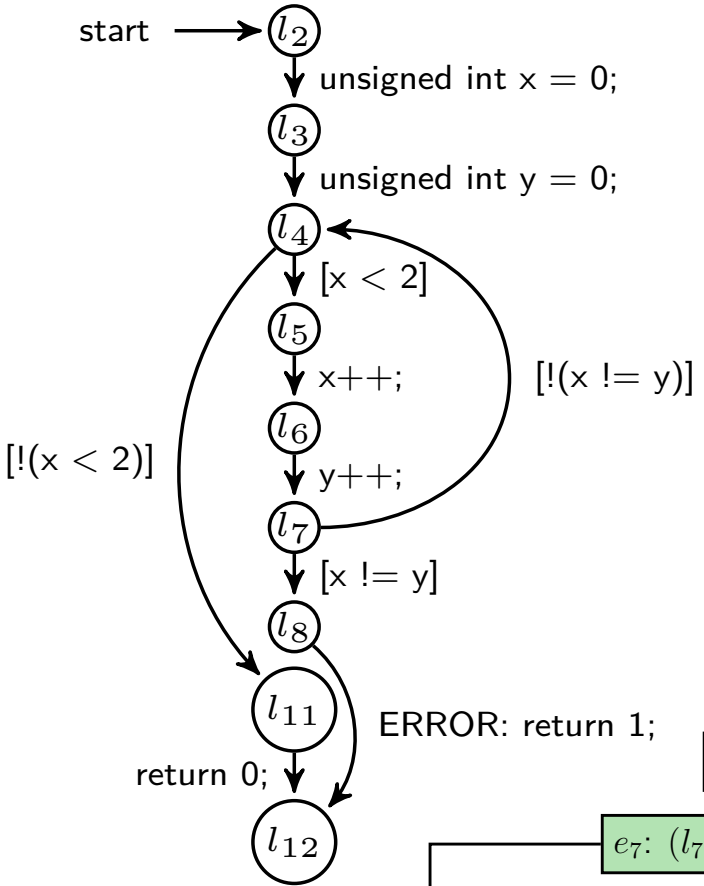
  - Good for finding bugs

Reuse Predicate CPA to Build BMC Query



Just add a CPA $\mathbb{LB}$ for counting and bounding loop iterations

Bounded Model Checking: Example



e_0: $(l_2, (true, l_2, true), \{l_4 \mapsto -1\})$

e_1: $(l_3, (true, l_2, x_0 = 0), \{l_4 \mapsto -1\})$

e_2: $(l_4, (true, l_2, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

e_3: $(l_{11}, (true, l_2, x_0 = 0 \wedge y_0 = 0 \wedge \neg(x_0 < 2)), \{l_4 \mapsto 0\})$

e_4: $(l_{12}, (true, l_2, x_0 = 0 \wedge y_0 = 0 \wedge \neg(x_0 < 2)), \{l_4 \mapsto 0\})$

e_5: $(l_5, (true, l_2, x_0 = 0 \wedge y_0 = 0 \wedge x_0 < 2), \{l_4 \mapsto 0\})$

e_6: $(l_6, (true, l_2, x_0 = 0 \wedge y_0 = 0 \wedge x_0 < 2 \wedge x_1 = x_0 + 1), \{l_4 \mapsto 0\})$

e_7: $(l_7, (true, l_2, x_0 = 0 \wedge y_0 = 0 \wedge x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1), \{l_4 \mapsto 0\})$

e_8: $(l_8, (true, l_2, x_0 = 0 \wedge y_0 = 0 \wedge x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(x_1 = y_1)), \{l_4 \mapsto 0\})$

e_9: $(l_{12}, (true, l_2, x_0 = 0 \wedge y_0 = 0 \wedge x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(x_1 = y_1)), \{l_4 \mapsto 0\})$

e_{10}: $(l_4, (true, l_2, x_0 = 0 \wedge y_0 = 0 \wedge x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

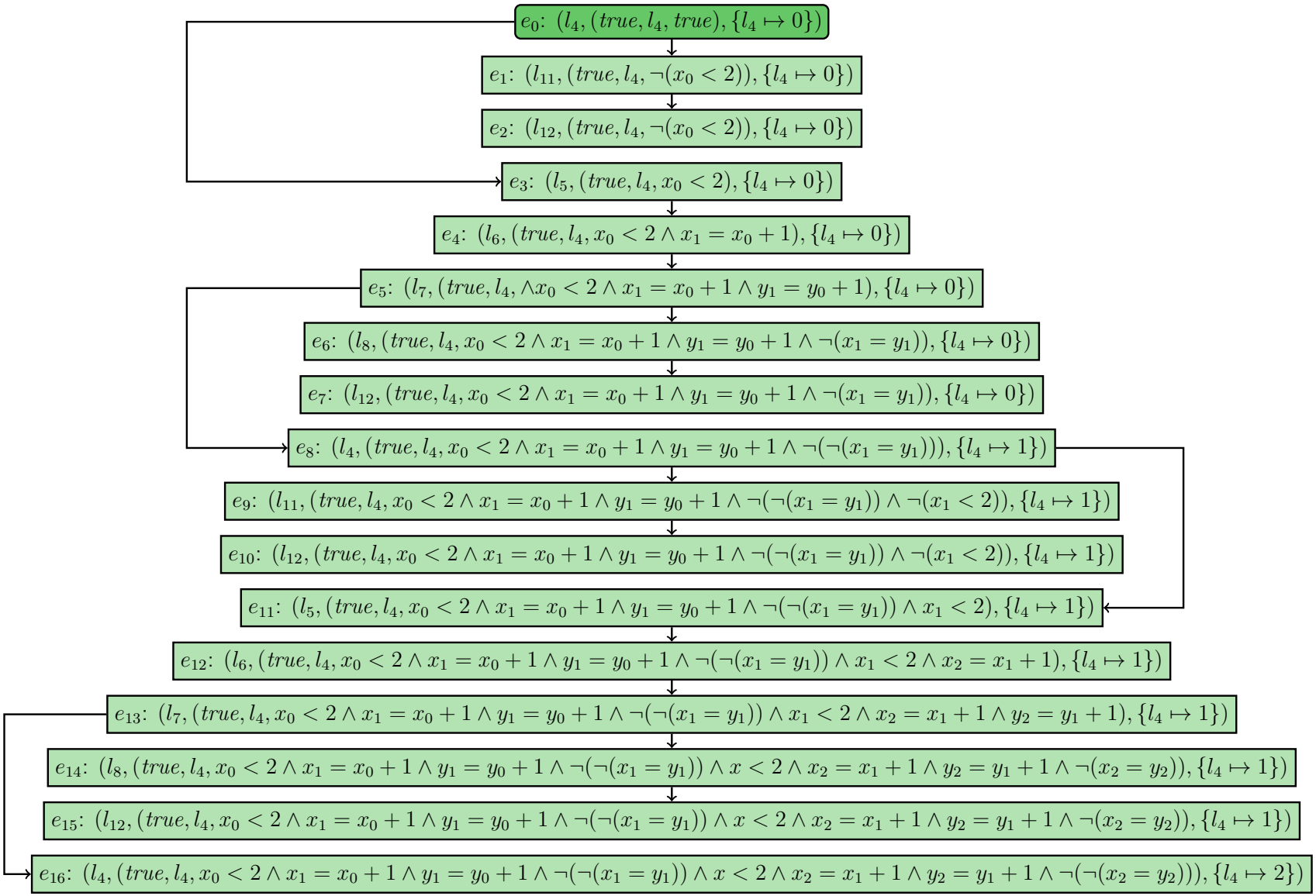
k-Induction

- k-Induction generalizes the induction principle:
  - No abstraction
  - Base case: Check that P holds in the first k iterations:
    $\rightarrow$ Equivalent to BMC with loop bound k
  - Step case: Check that the safety property is k-inductive:

$$\forall n : \left(\left(\bigwedge_{i=1}^{k} P(n+i-1) \right) \implies P(n+k) \right)$$

  - Stronger hypothesis is more likely to succeed
  - Add auxiliary invariants
  - Kahsai, Tinelli: [PDMC'11]
  - Heavy-weight proof technique

k-Induction: Example



k-Induction with Auxiliary Invariants

Induction:

1: $k = 1$
2: **while** !finished **do**
3: BMC(k)
4: Induction(k, invariants)
5: $k{+}{+}$

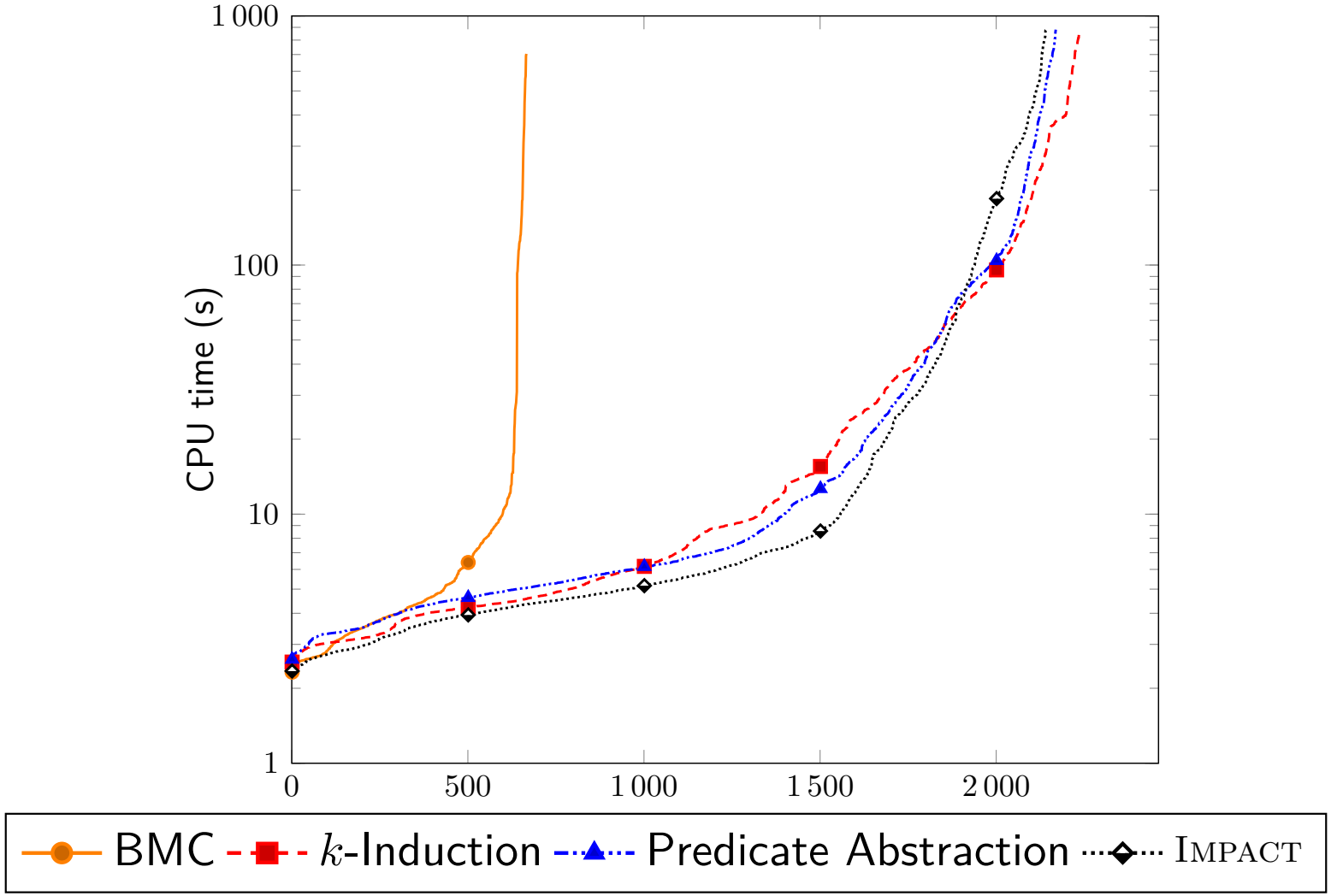
Invariant generation:

1: prec = <weak>
2: invariants = $\emptyset$
3: **while** !finished **do**
4: invariants = GenInv(prec)
5: prec = RefinePrec(prec)

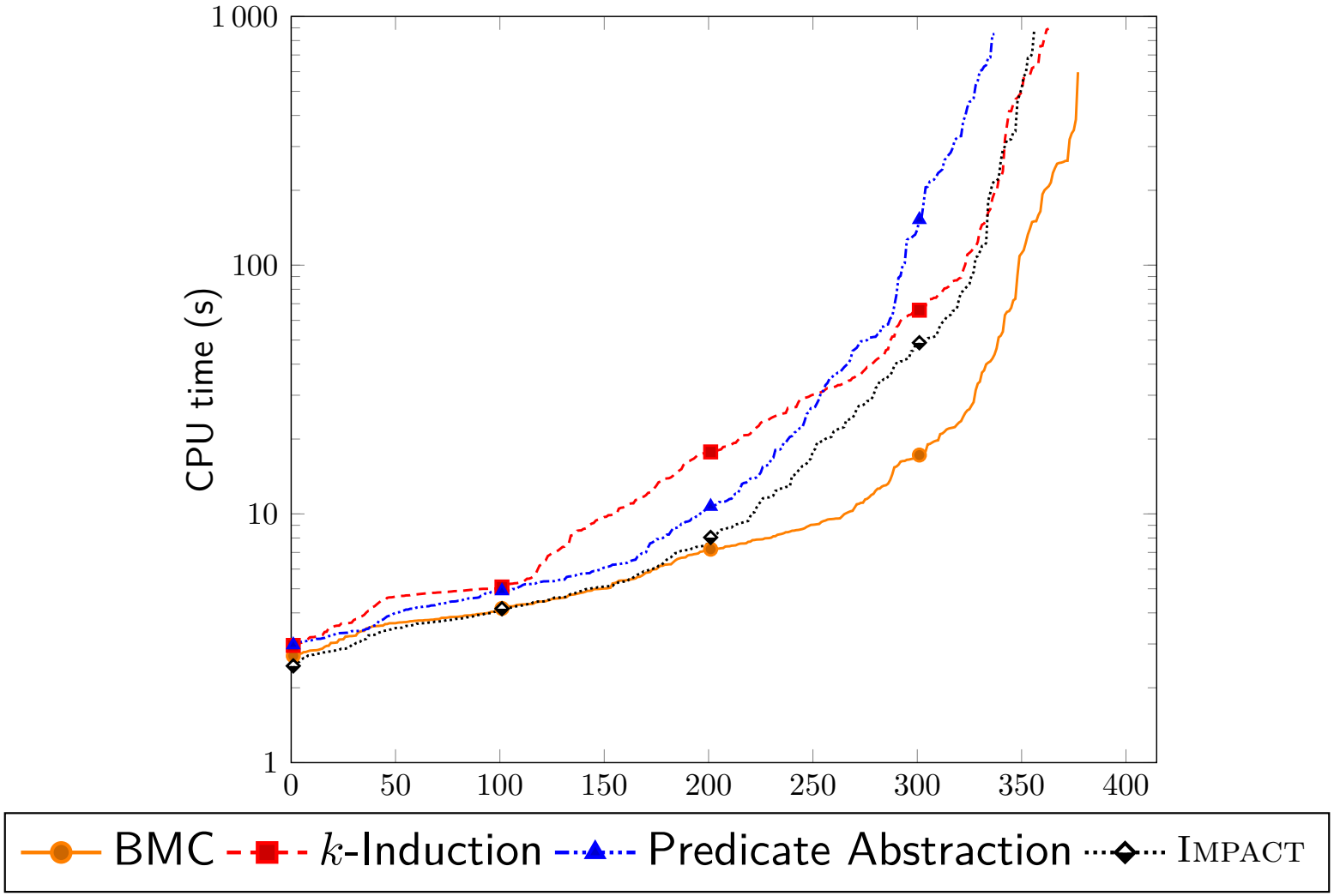
Experimental Comparison

- 5 287 verification tasks taken from SV-COMP'17
- 15 min timeout (CPU time)
- 15 GB memory
- SMT solver: MathSAT 5
- SMT theory: QF_UFBVFP
- Measured with BenchExec

All 3 913 bug-free tasks



All 1 374 tasks with known bugs

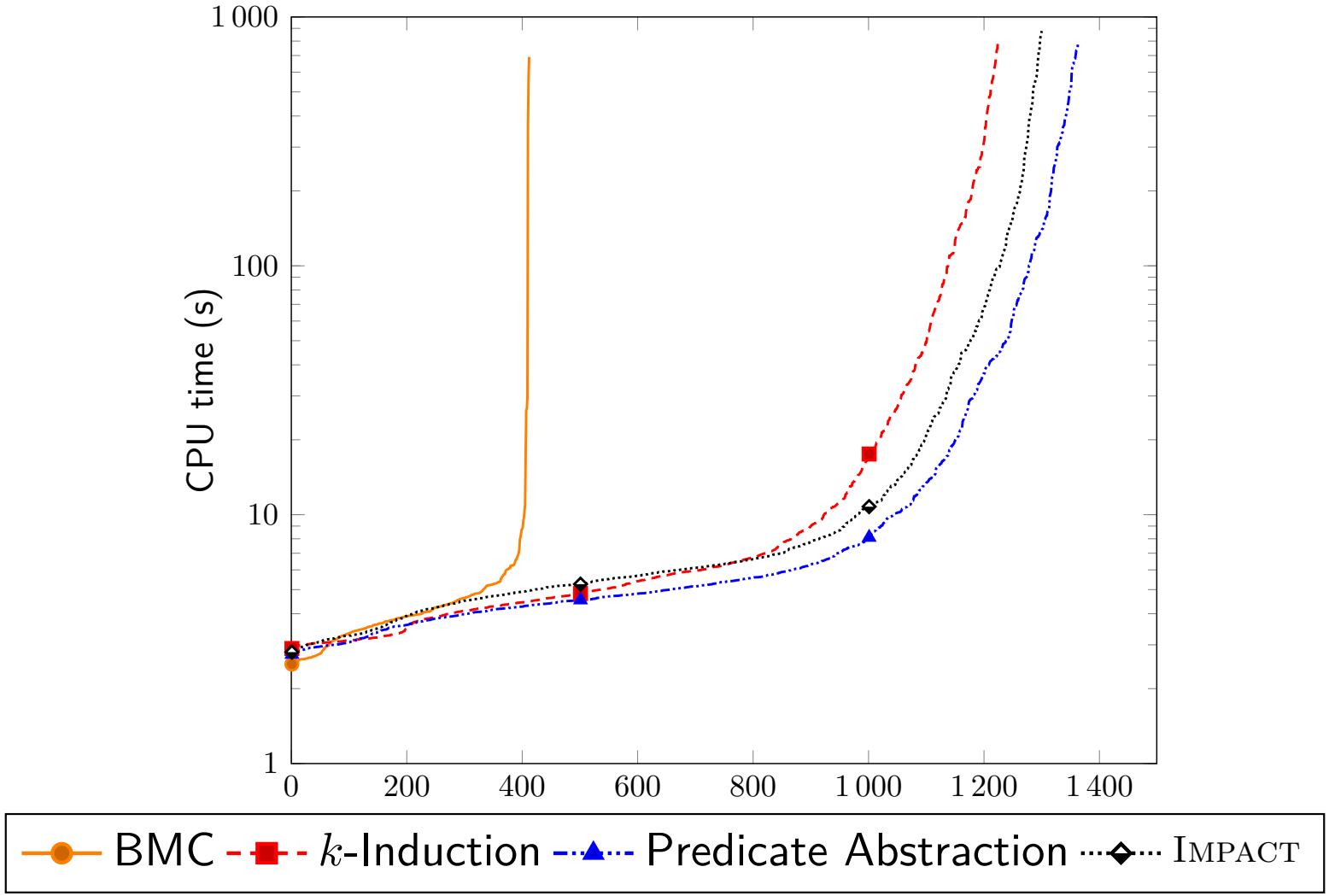


Category: Device Drivers

- Several thousands LOC per task
- Complex structures
- Pointer arithmetics

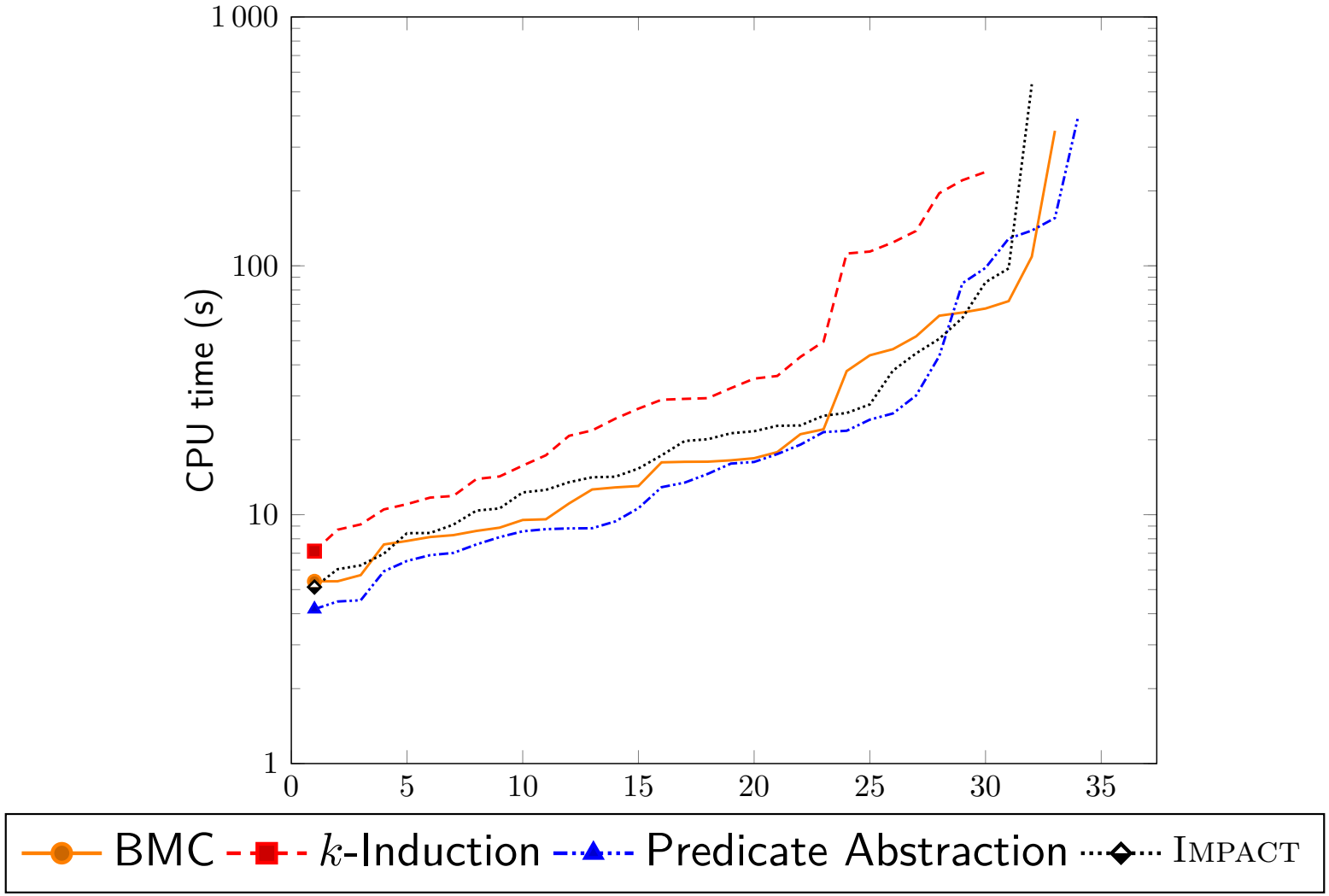
Category: Device Drivers

2 440 bug-free tasks:



Category: Device Drivers

355 tasks with known bugs:



Category: Event Condition Action Systems

- Several thousand LOC per task
- Auto-generated
- Only integer variables
- Linear and non-linear arithmetics
- Complex and dense control structure

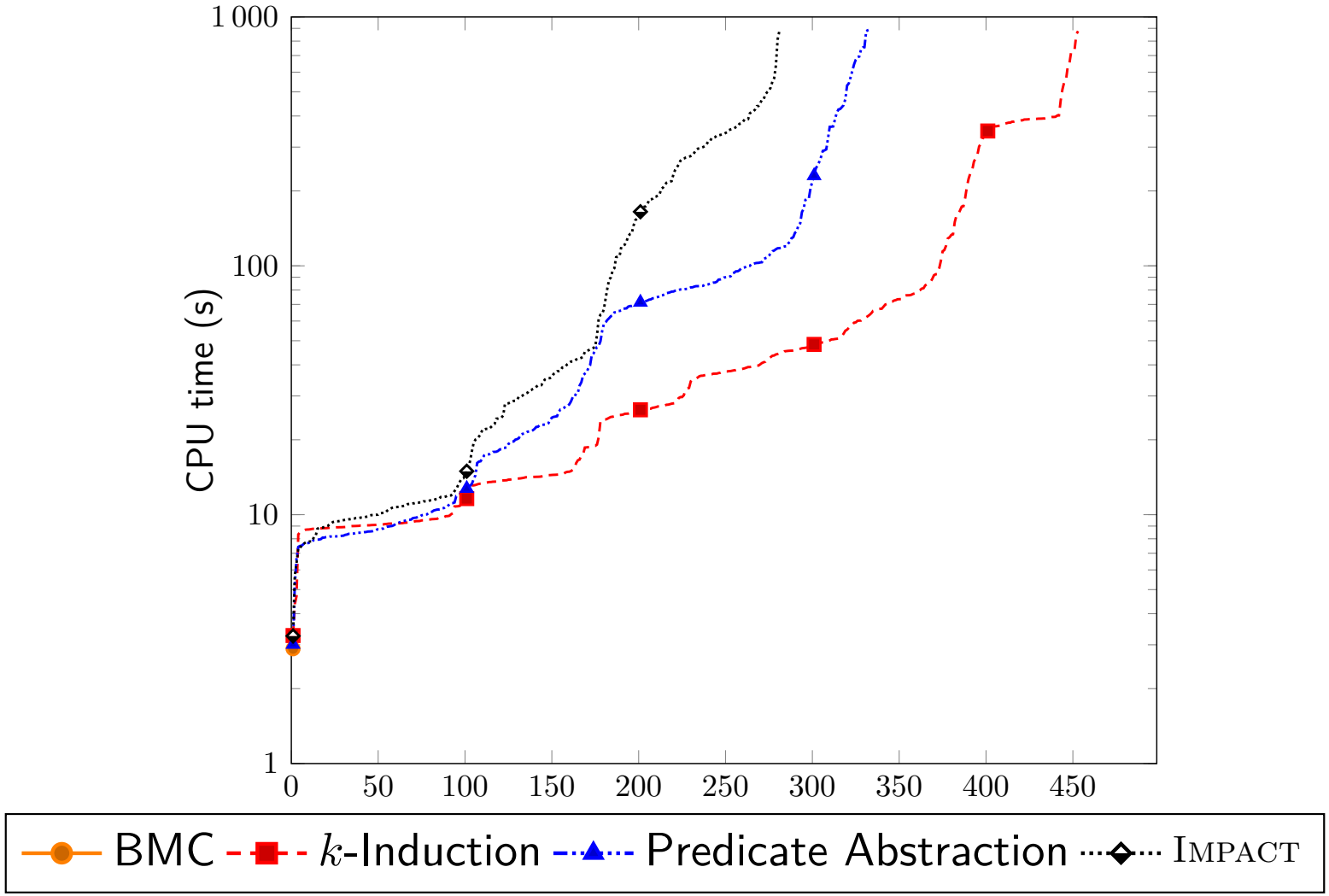
Category: Event Condition Action Systems

- Several thousand LOC per task
- Auto-generated
- Only integer variables
- Linear and non-linear arithmetics
- Complex and dense control structure

```
if (((a24==3) && (((a18==10) && ((input == 6)
    && ((115 < a3) && (306 >= a3))))
    && (a15==4)))) {
  a3 = (((a3 * 5) + -583604) * 1);
  a24 = 0;
  a18 = 8;
  return -1;
}
```

Category: Event Condition Action Systems

738 bug-free tasks:



Category: Event Condition Action Systems

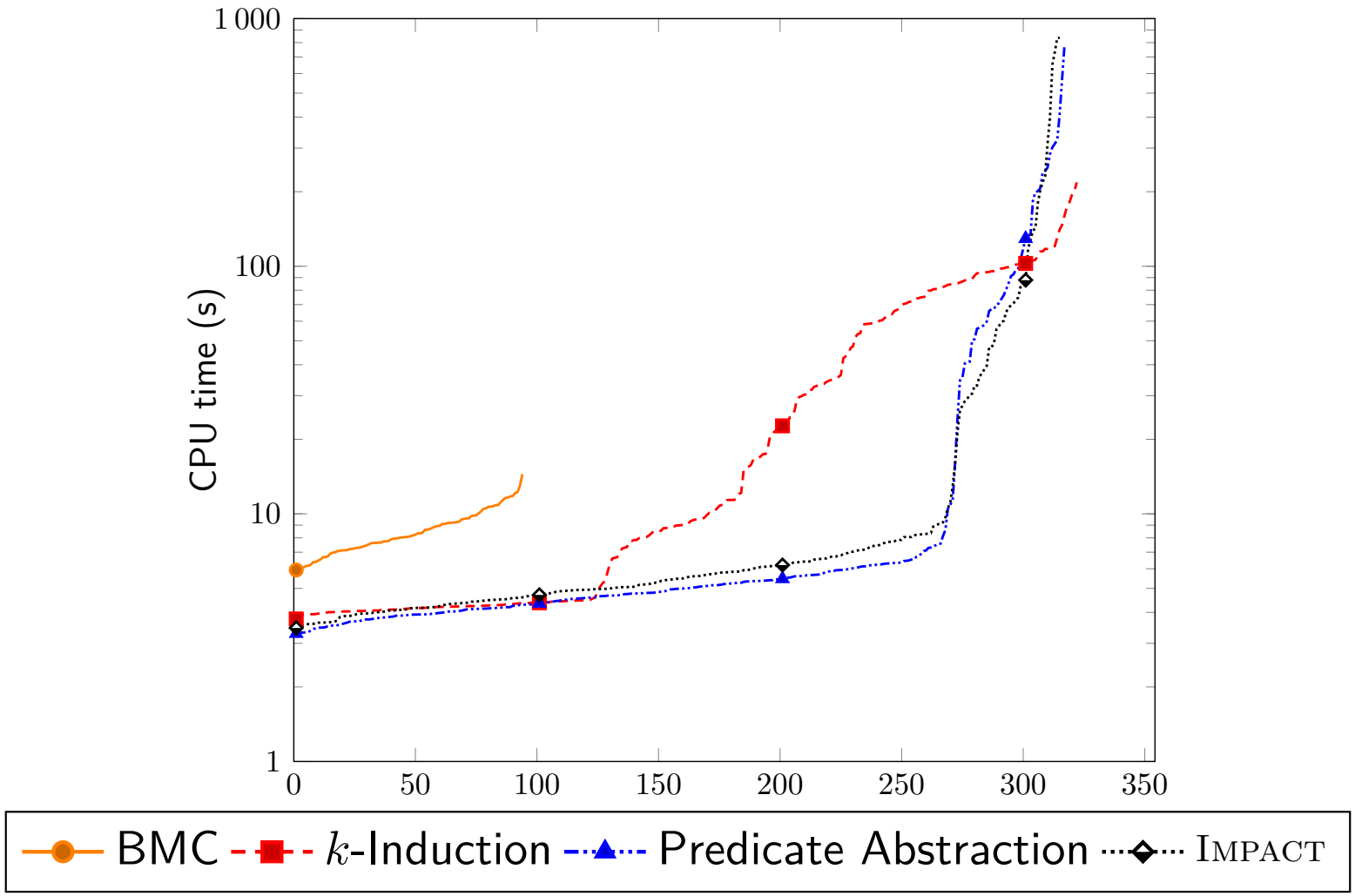
411 tasks with known bugs: Only BMC and k-induction find one bug (the same one).

Category: Product Lines

- Several hundred LOC
- Mostly integer variables, some structs
- Mostly simple linear arithmetics
- Lots of property-independent code

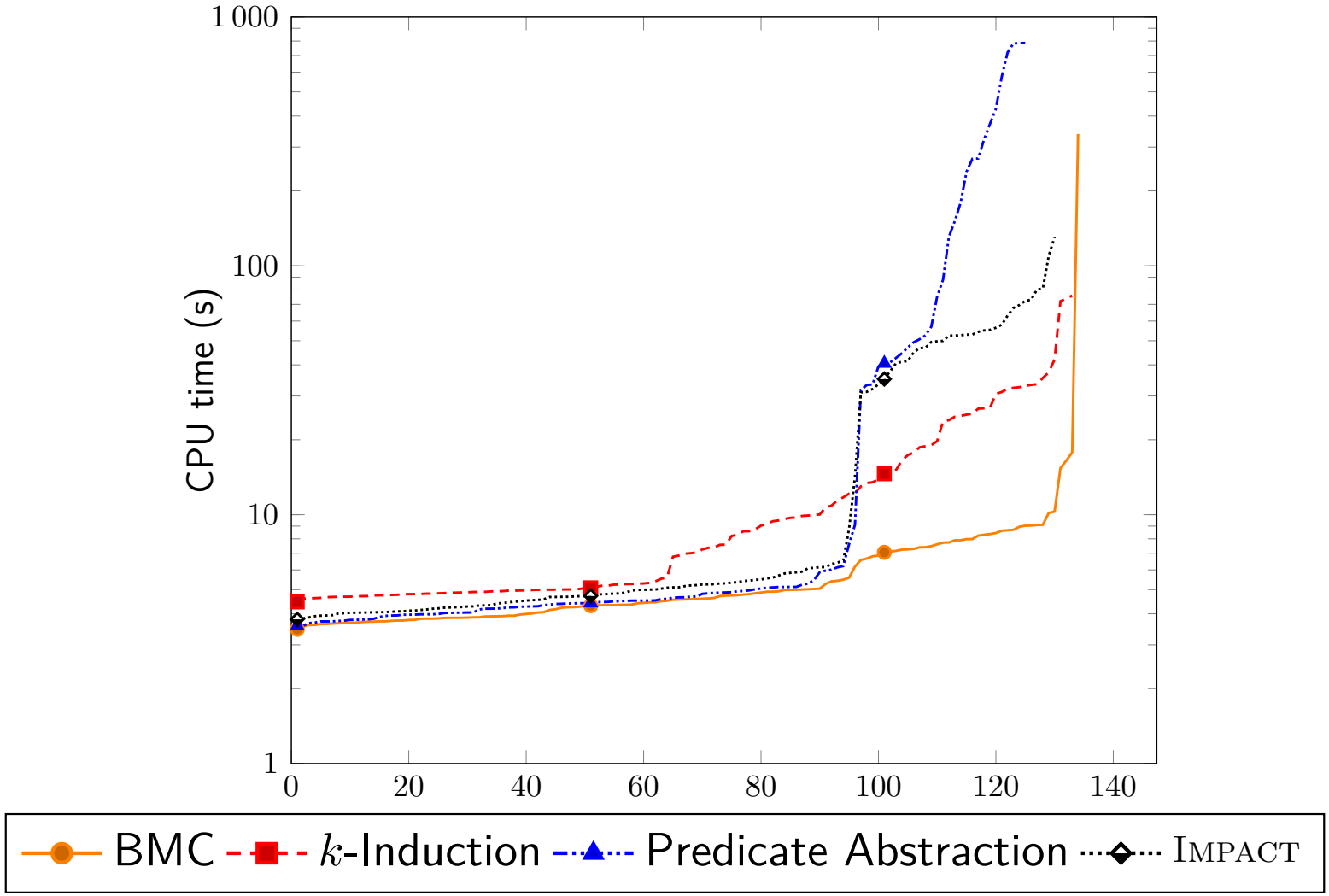
Category: Product Lines

332 bug-free tasks:



Category: Product Lines

265 tasks with known bugs:



Summary

We reconfirm that

- BMC is a good bug hunter
- k-Induction is a heavy-weight proof technique: effective, but slow
- CEGAR makes abstraction techniques (Predicate Abstraction, IMPACT) scalable
- IMPACT is lazy, and explores the state space and finds bugs quicker
- Predicate Abstraction is eager, and prunes irrelevant parts and finds proofs quicker

Outlook

- Find a way to integrate PDR into this framework
- Combine PDR with k-Induction