A Unifying View on SMT-Based Software Verification

Dirk Beyer
LMU Munich, Germany

Guest Lecture at RWTH Aachen, December 13, 2023

Based on [1]:

Dirk Beyer, Matthias Dangl, Philipp Wendler:

A Unifying View on SMT-Based Software Verification

Journal of Automated Reasoning, Volume 60, Issue 3, 2018.
https://doi.org/10.1007/s10817-017-9432-6

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, ...)
- New: Interpolation-based model checking
  (CPACHECKER)

Motivation

- Theoretical comparison difficult:
  - different conceptual optimizations (e.g., large-block encoding)
  - different presentation

$\rightarrow$ What are their core concepts and key differences?

Motivation

- Theoretical comparison difficult:
  - different conceptual optimizations (e.g., large-block encoding)
  - different presentation

  $\rightarrow$ What are their core concepts and key differences?
- Experimental comparison difficult:
  - implemented in different tools
  - different technical optimizations (e.g., data structures)
  - different front-end and utility code
  - different SMT solver

  $\rightarrow$ Where do performance differences actually come from?

Goals

- Provide a unifying framework for SMT-based algorithms
- Understand differences and key concepts of algorithms
- Determine potential of extensions and combinations
- Provide solid platform for experimental research

Approach

- Understand, and, if necessary, re-formulate the algorithms
- Design a configurable framework for SMT-based algorithms (based upon the CPA framework)
- Use flexibility of adjustable-block encoding (ABE)
- Express existing algorithms using the common framework
- Implement framework (in CPAchecker)

Base: Adjustable-Block Encoding

Originally for predicate abstraction:

- 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 heads (hard-coded)
- Adjustable-Block Encoding:
  introduce block operator "blk" to make it configurable

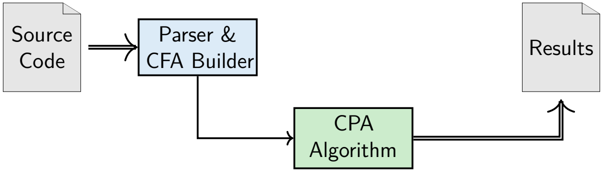
Base: Configurable Program Analysis

Configurable Program Analysis (CPA):

- Beyer, Henzinger, Théoduloz: [2, CAV '07]
- One single unifying algorithm for all algorithms based on state-space exploration
- Configurable components: abstract domain, abstract-successor computation, path sensitivity, ...

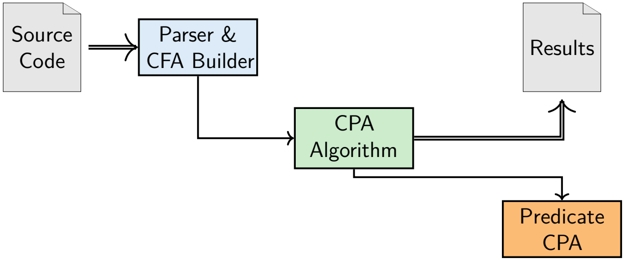
Using the CPA Framework

- CPA Algorithm is a configurable reachability analysis for arbitrary abstract domains



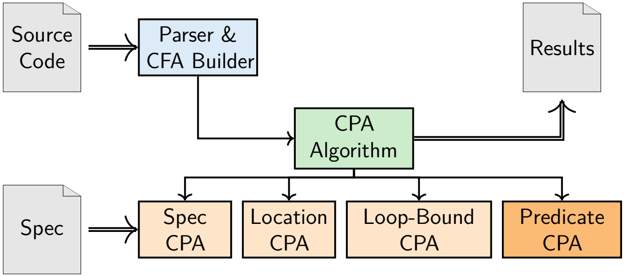
Using the CPA Framework

- CPA Algorithm is a configurable reachability analysis for arbitrary abstract domains
- Provide Predicate CPA for our predicate-based abstract domain



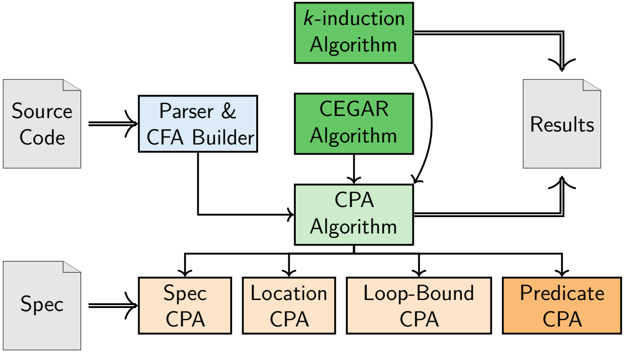
Using the CPA Framework

- CPA Algorithm is a configurable reachability analysis for arbitrary abstract domains
- Provide Predicate CPA for our predicate-based abstract domain
- Reuse other CPAs

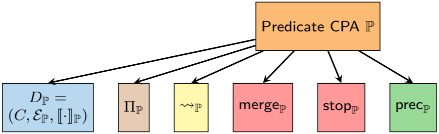


Using the CPA Framework

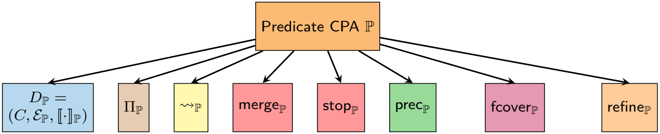
- CPA Algorithm is a configurable reachability analysis for arbitrary abstract domains
- Provide Predicate CPA for our predicate-based abstract domain
- Reuse other CPAs
- Build further algorithms on top that make use of reachability analysis



Predicate CPA



Predicate CPA



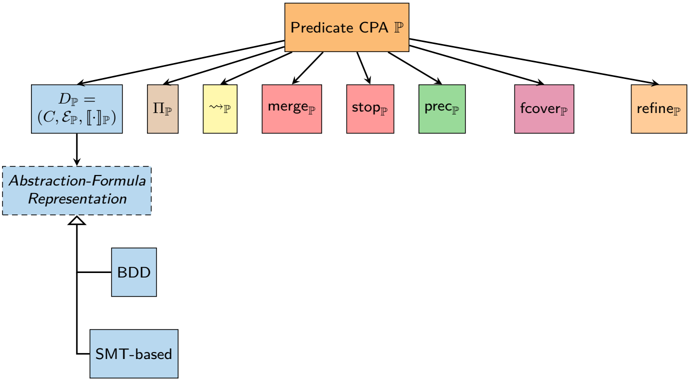
Predicate CPA: Abstract Domain

- Abstract state: (ψ, φ)
  - tuple of abstraction formula ψ and path formula φ (for ABE)
  - conjunction represents state space
  - abstraction formula can be a BDD or an SMT formula
  - path formula is always SMT formula and concrete

Predicate CPA: Abstract Domain

- Abstract state: (ψ, φ)
  - tuple of abstraction formula ψ and path formula φ (for ABE)
  - conjunction represents state space
  - abstraction formula can be a BDD or an SMT formula
  - path formula is always SMT formula and concrete
- Precision: set of predicates (per program location)

Predicate CPA



Predicate CPA: CPA Operators

- Transfer relation:
  - computes strongest post
  - changes only path formula, new abstract state is (ψ, φ')
  - purely syntactic, cheap
  - variety of encodings using different SMT theories possible (different approximations for arithmetic and heap operations)

Predicate CPA: CPA Operators

- Transfer relation:
  - computes strongest post
  - changes only path formula, new abstract state is (ψ, φ')
  - purely syntactic, cheap
  - variety of encodings using different SMT theories possible (different approximations for arithmetic and heap operations)
- Merge operator:
  - standard for ABE: create disjunctions inside block

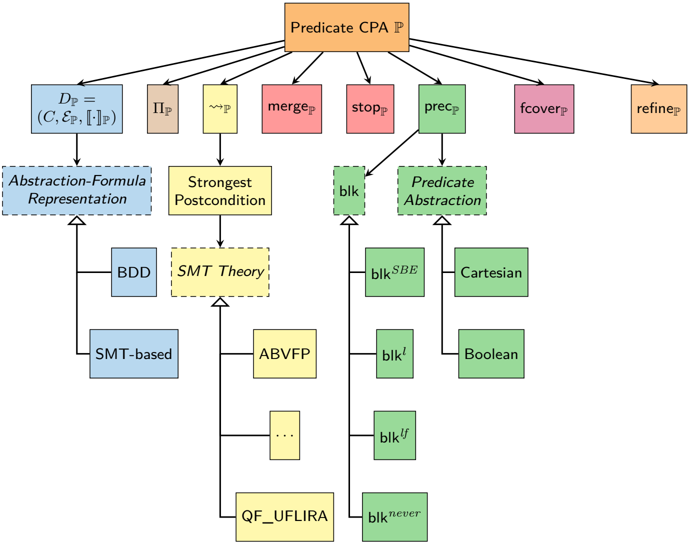
Predicate CPA: CPA Operators

- Transfer relation:
  - computes strongest post
  - changes only path formula, new abstract state is (ψ, φ')
  - purely syntactic, cheap
  - variety of encodings using different SMT theories possible (different approximations for arithmetic and heap operations)
- Merge operator:
  - standard for ABE: create disjunctions inside block
- Stop operator:
  - standard for ABE: check coverage only at block ends

Predicate CPA: CPA Operators

- Transfer relation:
  - computes strongest post
  - changes only path formula, new abstract state is (ψ, φ')
  - purely syntactic, cheap
  - variety of encodings using different SMT theories possible (different approximations for arithmetic and heap operations)
- Merge operator:
  - standard for ABE: create disjunctions inside block
- Stop operator:
  - standard for ABE: check coverage only at block ends
- Precision-adjustment operator:
  - only active at block ends (as determined by blk)
  - computes abstraction of current abstract state
  - new abstract state is $(\psi', true)$

Predicate CPA

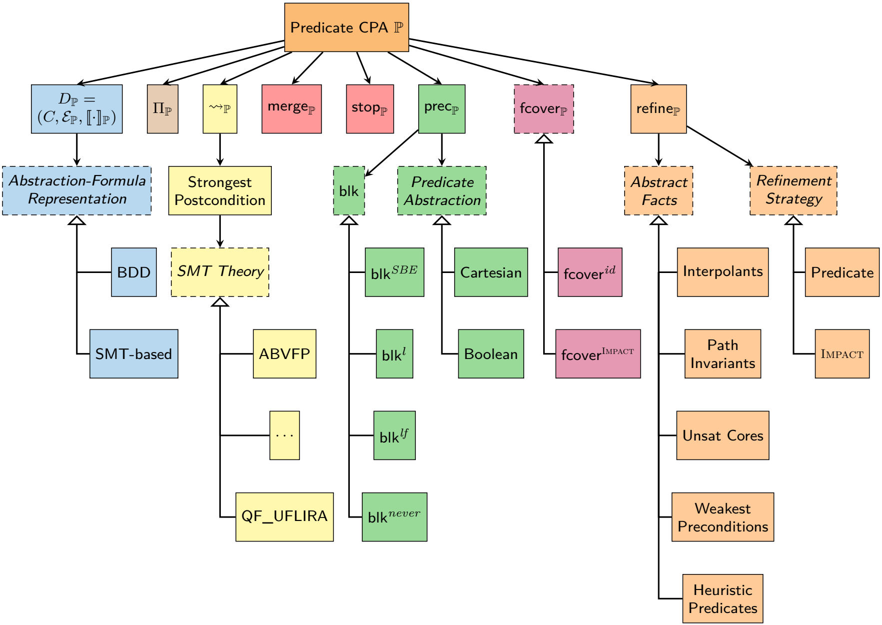


Predicate CPA: Refinement

Four steps:

1. Reconstruct ARG path to abstract error state
2. Check feasibility of path
3. Discover abstract facts, e.g.,
   - interpolants
   - weakest precondition
   - heuristics
4. Refine abstract model
   - add predicates to precision, cut ARG
     or
   - conjoin interpolants to abstract states,
     recheck coverage relation

Predicate CPA



Predicate Abstraction

- Predicate Abstraction
  - [5, CAV '97], [7, POPL '02], [6, POPL '04]
  - Abstract-interpretation technique
  - Abstract domain constructed from a set of predicates π
  - Use CEGAR to add predicates to π (refinement) [4, J. ACM '03]
  - Derive new predicates using Craig interpolation
  - Abstraction formula as BDD

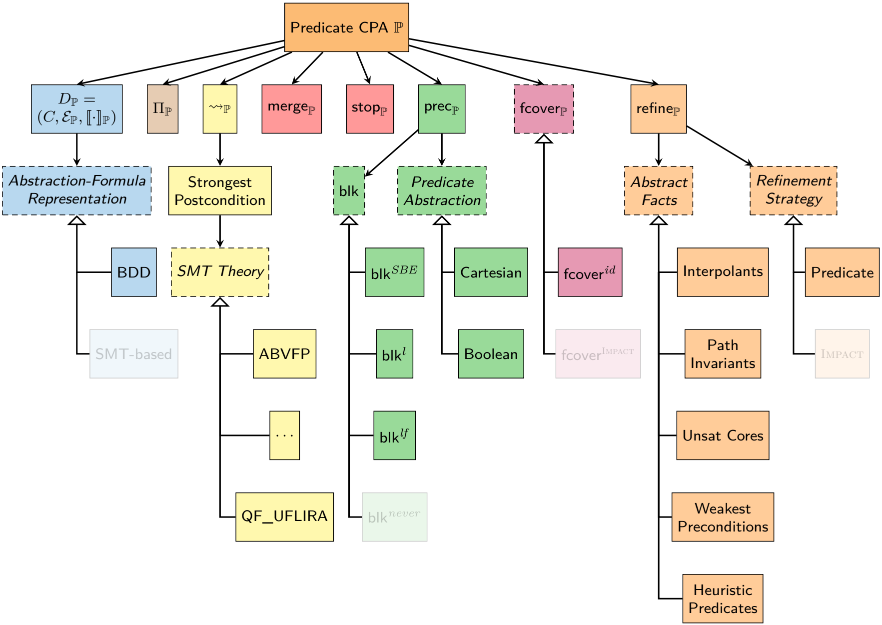
Expressing Predicate Abstraction

- Abstraction Formulas: BDDs
- Block Size (blk): e.g. blk^{SBE} or blk^{l} or blk^{lf}
- Refinement Strategy: add predicates to precision, cut ARG

Use CEGAR Algorithm:

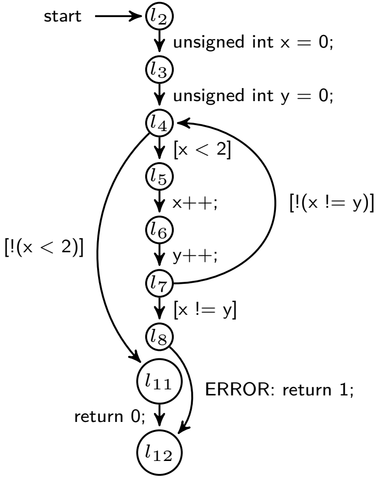
```
while true do
    run CPA Algorithm
    if target state found then
        call refine
        if target state reachable then
            return false
    else
        return true
```

Predicate CPA

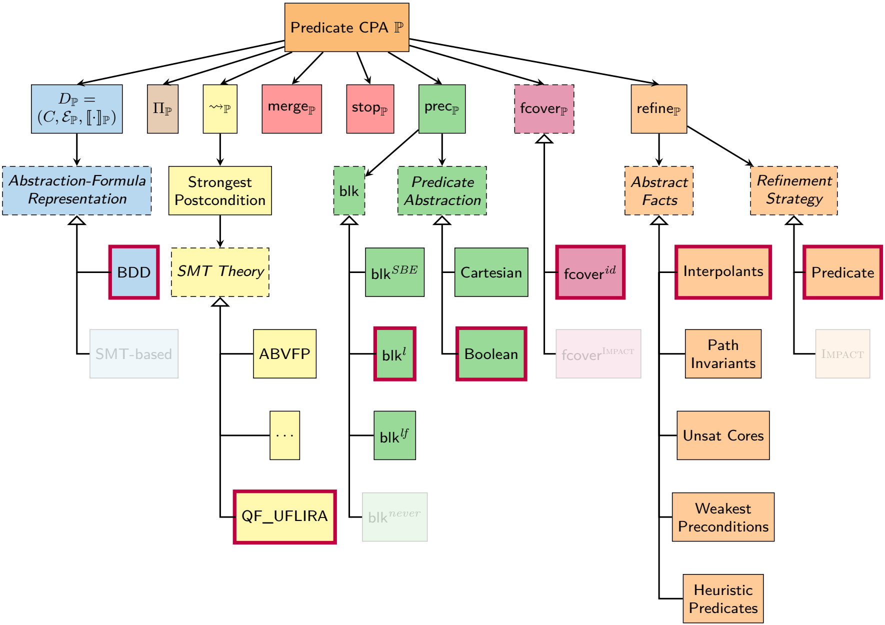


Example Program

```
int main() {
  unsigned int x = 0;
  unsigned int y = 0;
  while (x < 2) {
    x++;
    y++;
    if (x != y) {
      ERROR: return 1;
    }
  }
  return 0;
}
```

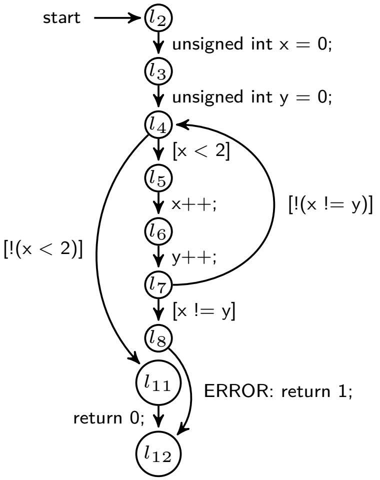


Predicate CPA



Predicate Abstraction: Example

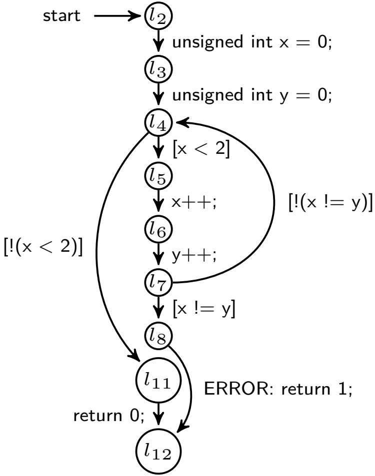
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

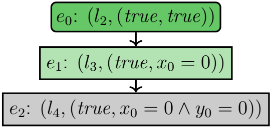


e_0: $(l_2, (true, true))$

Predicate Abstraction: Example

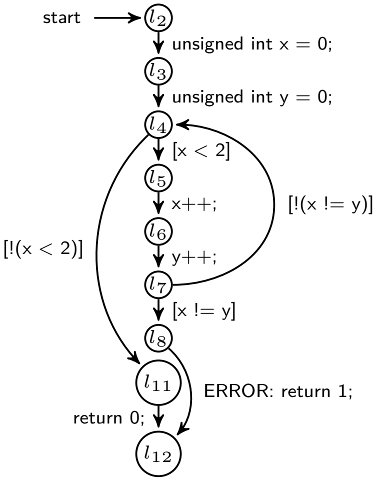
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

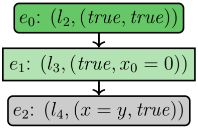




Predicate Abstraction: Example

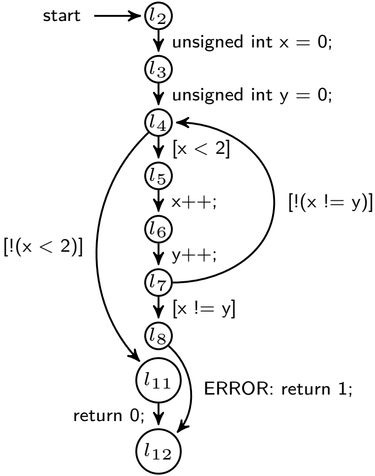
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

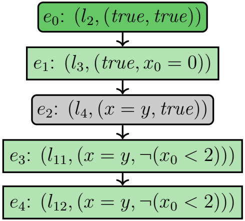




Predicate Abstraction: Example

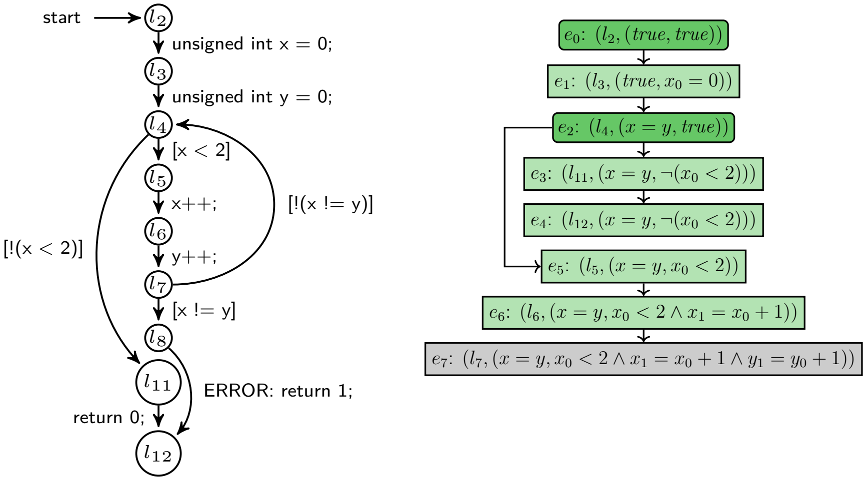
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$





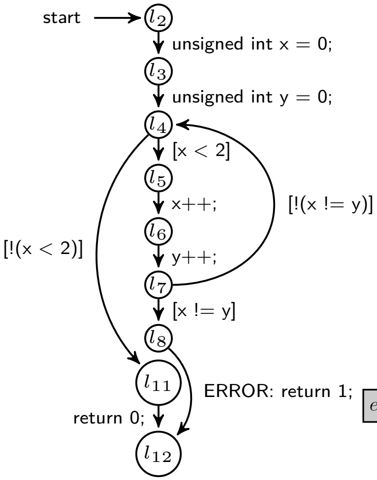
Predicate Abstraction: Example

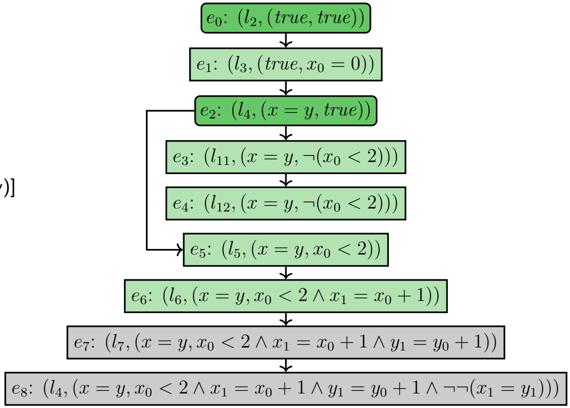
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$



Predicate Abstraction: Example

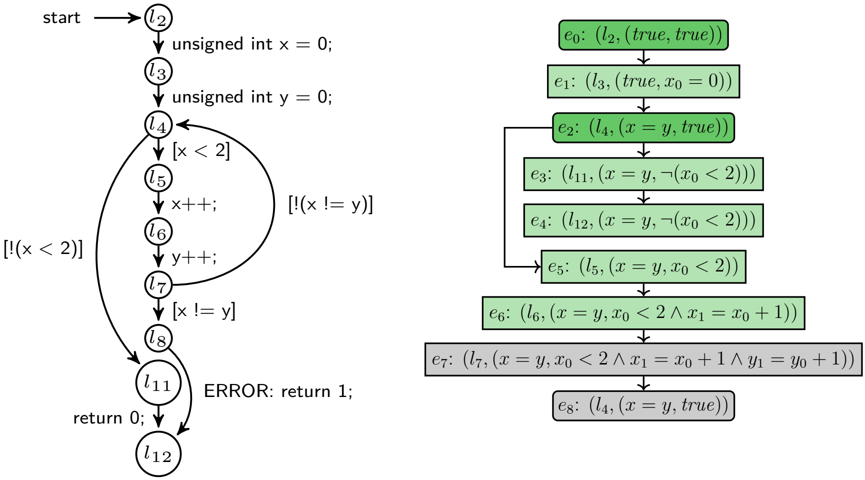
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$





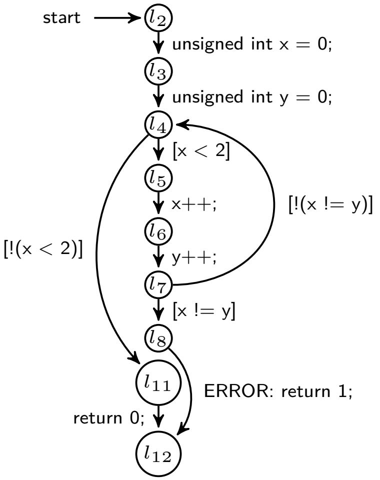
Predicate Abstraction: Example

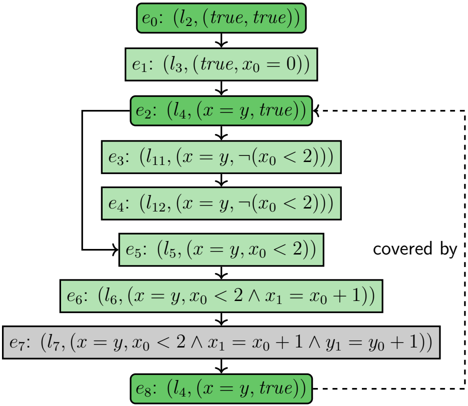
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$



Predicate Abstraction: Example

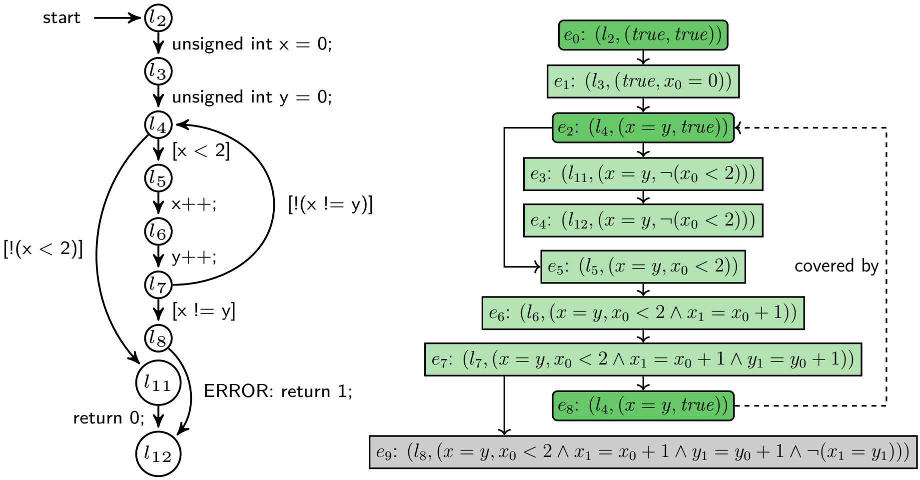
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$





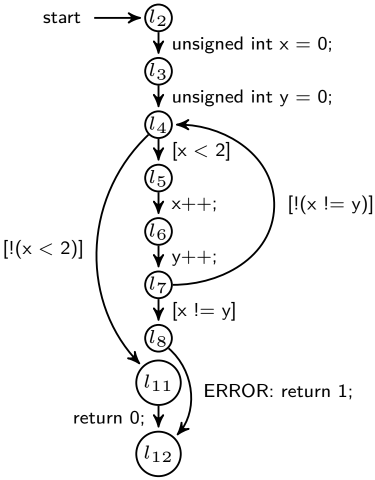
Predicate Abstraction: Example

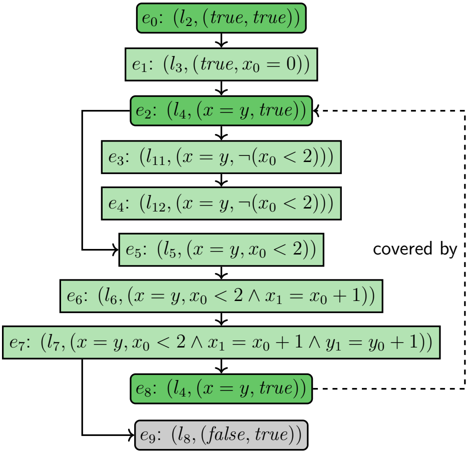
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$



Predicate Abstraction: Example

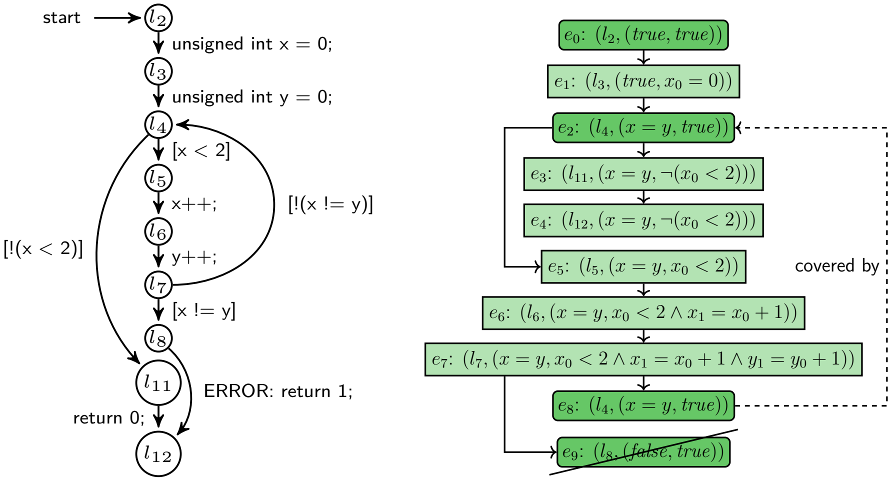
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$





Predicate Abstraction: Example

with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$



Impact

- Impact
  - "Lazy Abstraction with Interpolants" [10, CAV '06]
  - Abstraction is derived dynamically/lazily
  - Solution to avoiding expensive abstraction computations
  - Compute fixed point over three operations
    - Expand
    - Refine
    - Cover
  - Abstraction formula as SMT formula
  - Optimization: forced covering

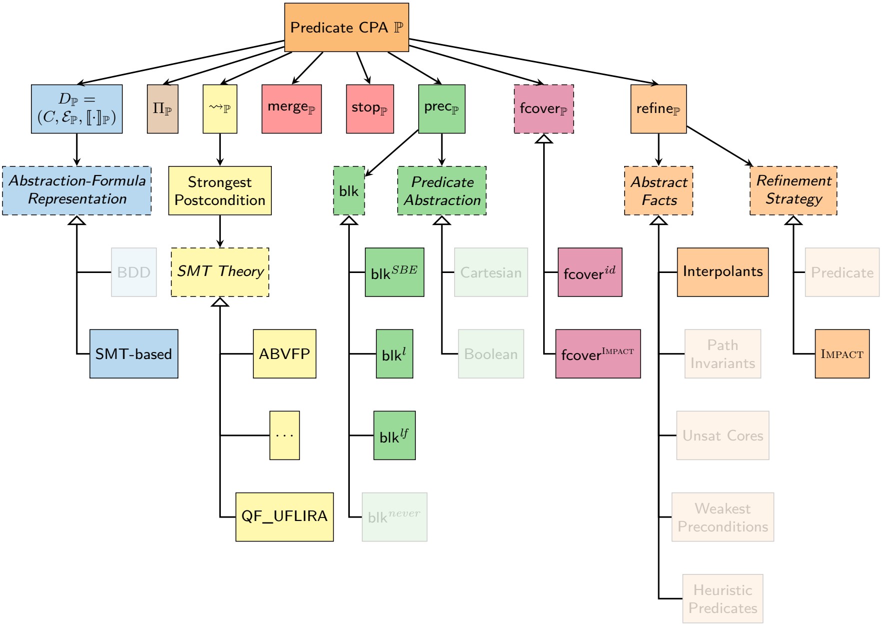
Expressing Impact

- Abstraction Formulas: SMT-based
- Block Size (blk): blk^{SBE} or other (new!)
- Refinement Strategy:
  conjoin interpolants to abstract states,
  recheck coverage relation

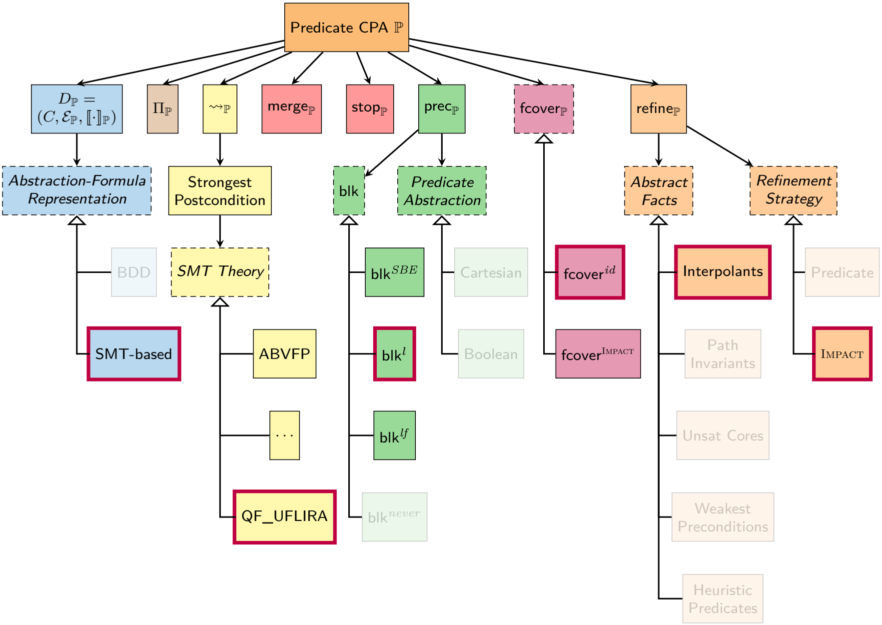
Furthermore:

- Use CEGAR Algorithm
- Precision stays empty
  $\rightarrow$ predicate abstraction never computed

Predicate CPA

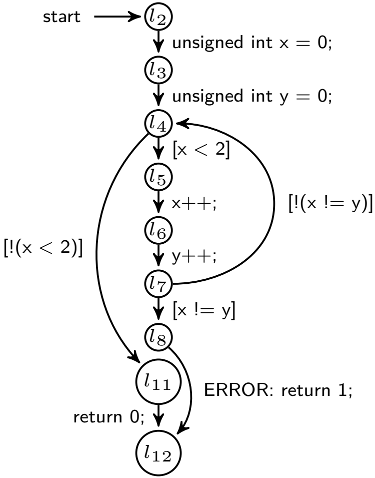


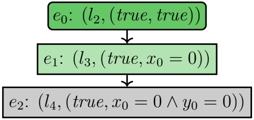
Predicate CPA



IMPACT: Example

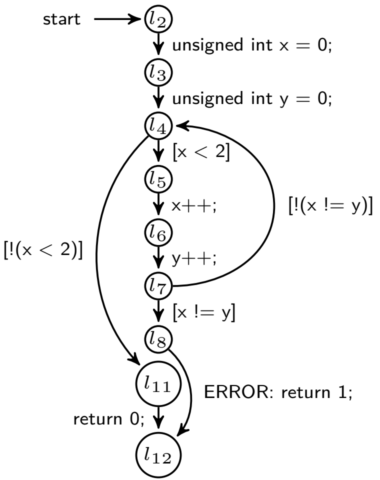
with blk^l

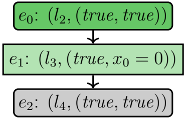




IMPACT: Example

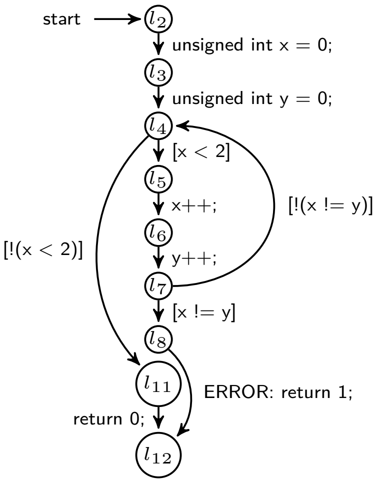
with blk^l

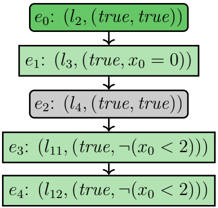




Impact: Example

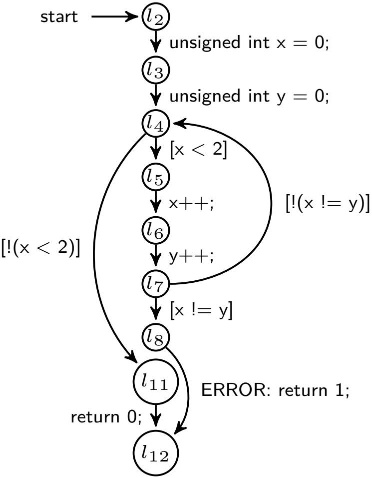
with blk^l

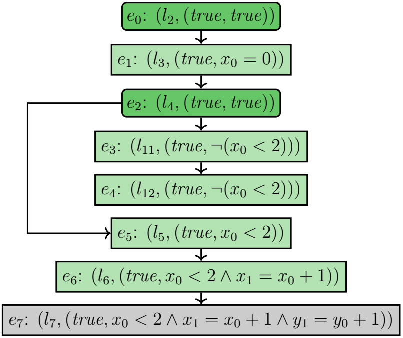




Impact: Example

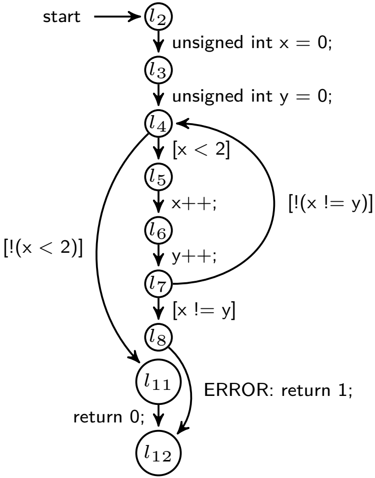
with blk^l

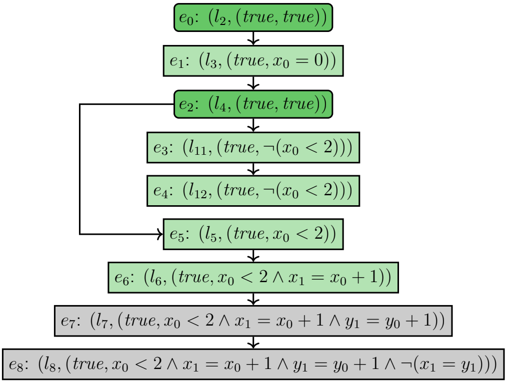




IMPACT: Example

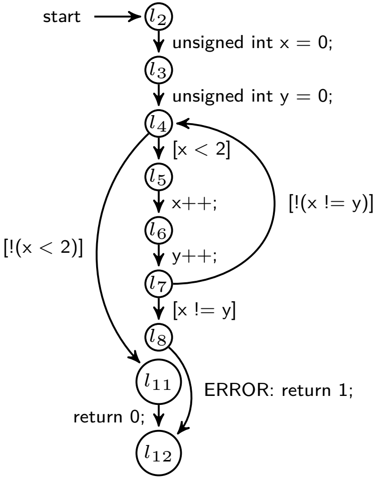
with blk^l

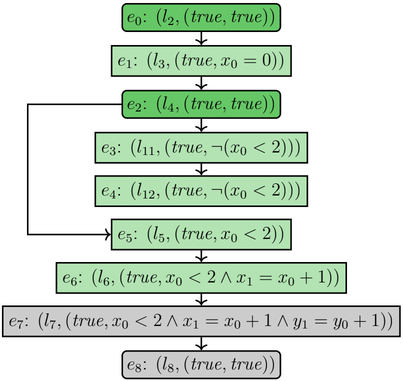




Impact: Example

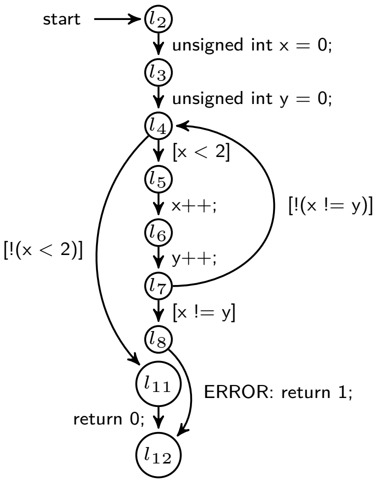
with blk^l

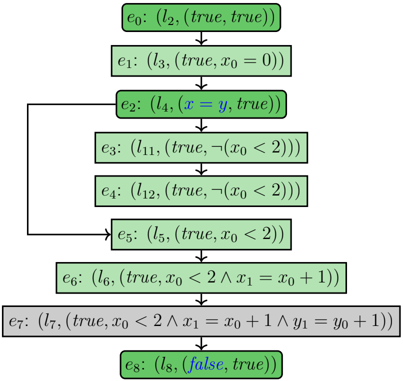




IMPACT: Example

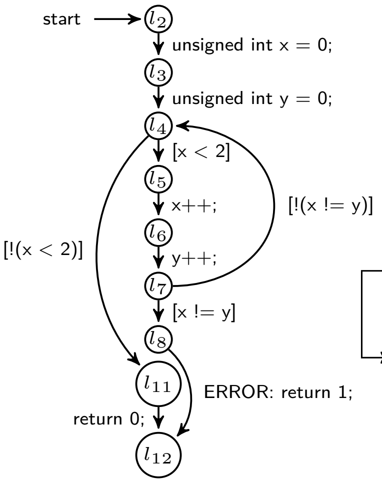
with blk^l

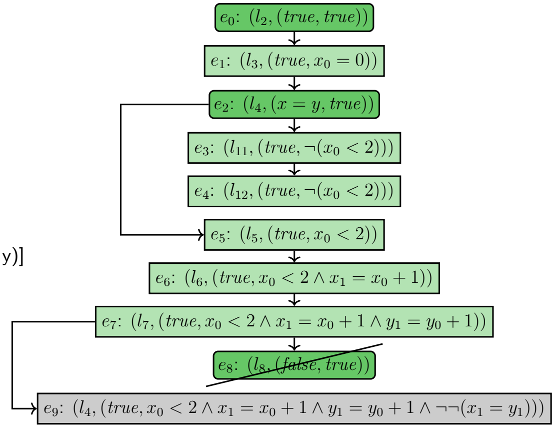




Impact: Example

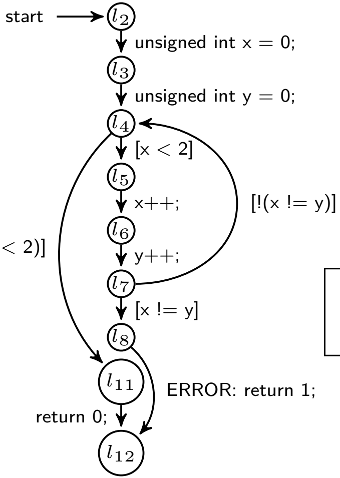
with blk^l

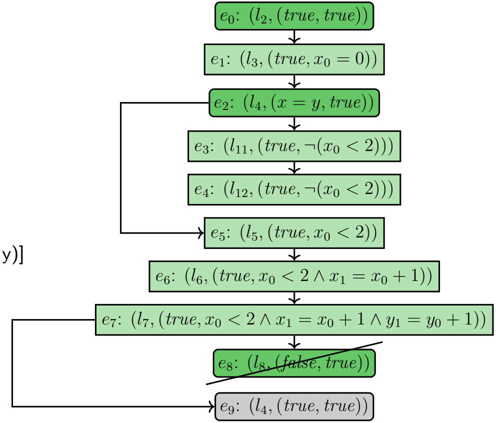




IMPACT: Example

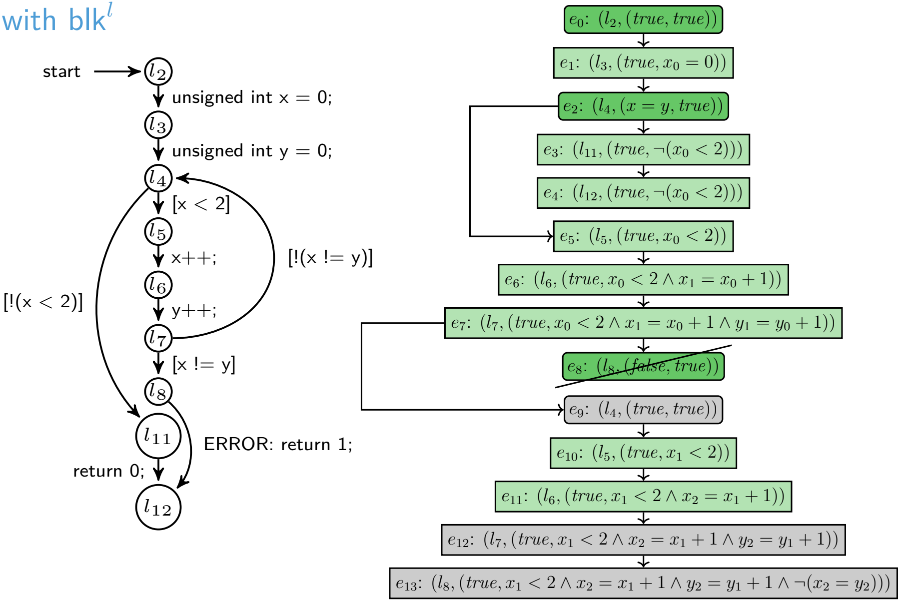
with blk^l





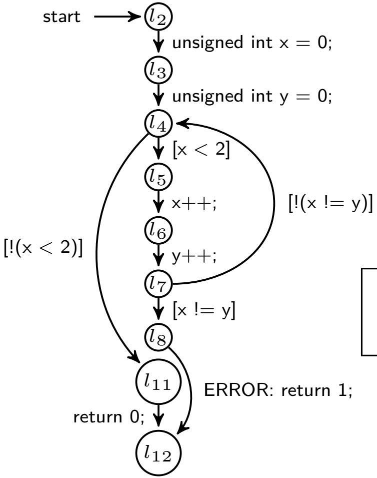
Impact: Example

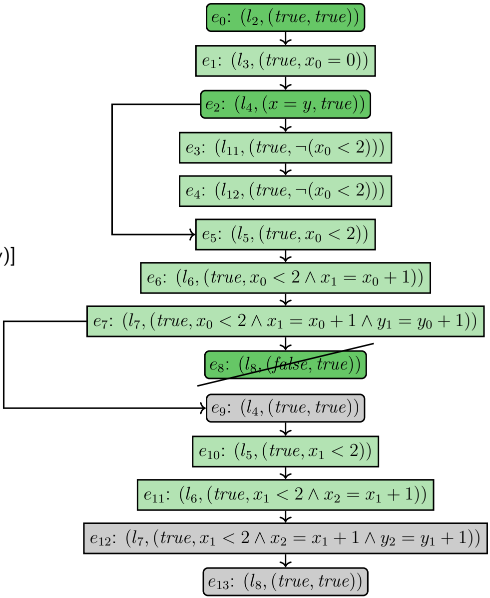
with blk^l



Impact: Example

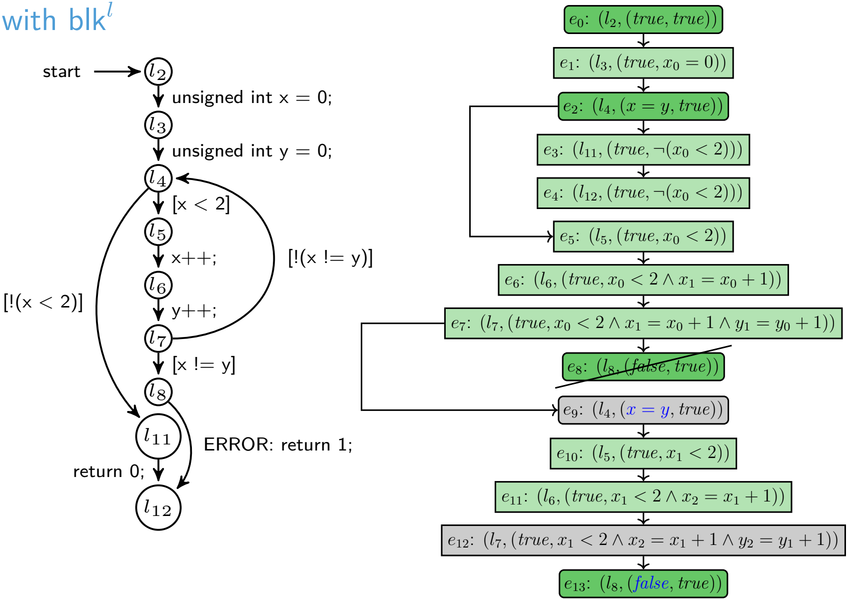
with blk^l





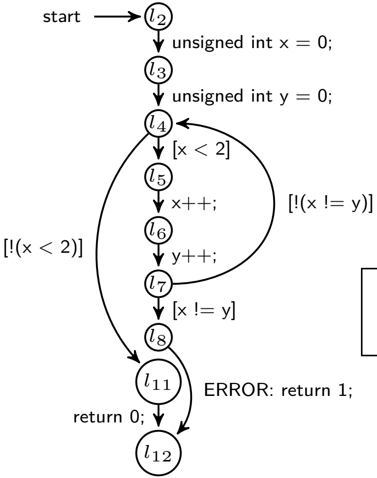
Impact: Example

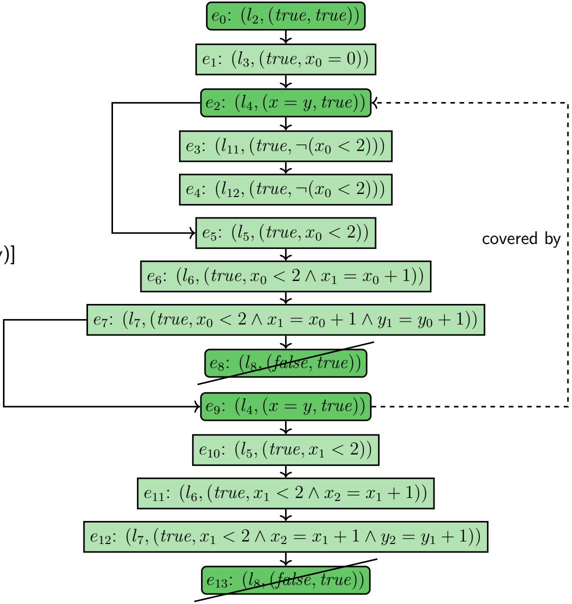
with blk^l



Impact: Example

with blk^l





Bounded Model Checking

- Bounded Model Checking:
  - Biere, Cimatti, Clarke, Zhu: [3, TACAS '99]
  - No abstraction
  - Unroll loops up to a loop bound k
  - Check that P holds in the first k iterations:

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

Expressing BMC

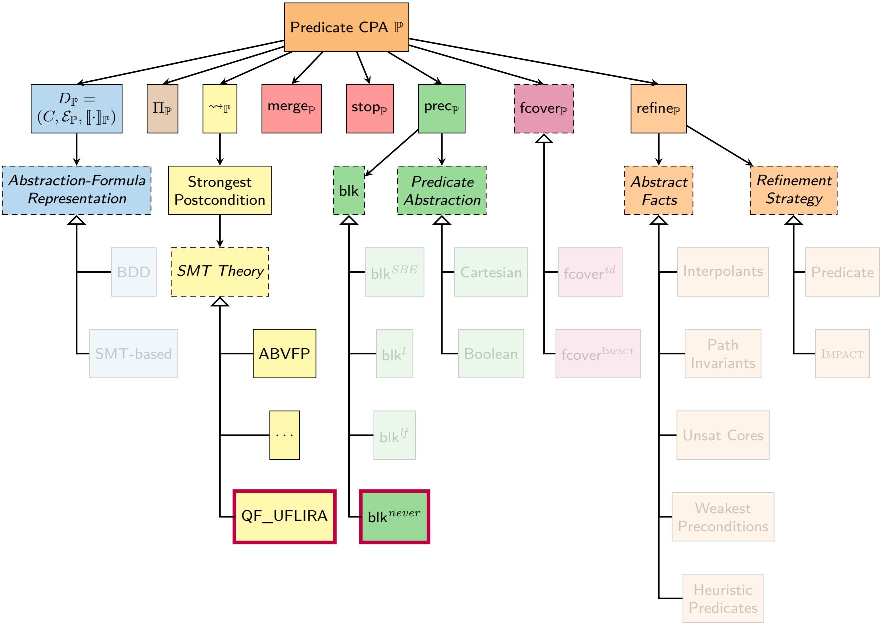
- Block Size (blk): blk^{never}

Furthermore:

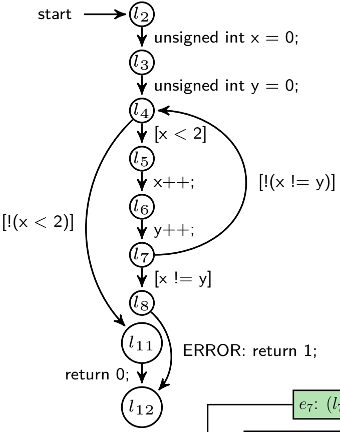
- Add CPA for bounding state space (e.g., loop bounds)
- Choices for abstraction formulas and refinement irrelevant because block end never encountered
- Use Algorithm for iterative BMC:

```
k = 1
while !finished do
    run CPA Algorithm
    check feasibility of each abstract error state
    k++
```

Predicate CPA



Bounded Model Checking: Example with $k = 1$



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

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

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

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

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

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

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

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

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

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

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

1-Induction

- 1-Induction:
  - Base case: Check that the safety property holds in the first loop iteration:

    $$P(1)$$

    $\rightarrow$ Equivalent to BMC with loop bound 1
  - Step case: Check that the safety property is 1-inductive:

    $$\forall n : (P(n) \Rightarrow P(n+1))$$

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:

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

  - Stronger hypothesis is more likely to succeed
  - Add auxiliary invariants
  - Kahsai, Tinelli: [8, PDMC'11]

k-Induction with Auxiliary Invariants

Induction:

$k = 1$
while !finished **do**
    BMC(k)
    Induction(k, invariants)
    k++

Invariant generation:

prec = <weak>
invariants = $\emptyset$
while !finished **do**
    invariants = GenInv(prec)
    prec = RefinePrec(prec)

k-Induction: Example with $k = 1$ (and loop bound $k + 1 = 2$)

e_0: $(l_4, (\mathit{true}, \mathit{true}), \{l_4 \mapsto 0\})$

e_1: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)), \{l_4 \mapsto 0\})$

e_2: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)), \{l_4 \mapsto 0\})$

e_3: $(l_5, (\mathit{true}, x_0 < 2), \{l_4 \mapsto 0\})$

e_4: $(l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1), \{l_4 \mapsto 0\})$

e_5: $(l_7, (\mathit{true}, \wedge x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1), \{l_4 \mapsto 0\})$

e_6: $(l_8, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(x_1 = y_1)), \{l_4 \mapsto 0\})$

e_7: $(l_{12}, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(x_1 = y_1)), \{l_4 \mapsto 0\})$

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

e_9: $(l_{11}, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1)) \wedge \neg(x_1 < 2)), \{l_4 \mapsto 1\})$

e_{10}: $(l_{12}, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1)) \wedge \neg(x_1 < 2)), \{l_4 \mapsto 1\})$

e_{11}: $(l_5, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1)) \wedge x_1 < 2), \{l_4 \mapsto 1\})$

e_{12}: $(l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1)) \wedge x_1 < 2 \wedge x_2 = x_1 + 1), \{l_4 \mapsto 1\})$

e_{13}: $(l_7, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1)) \wedge x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1), \{l_4 \mapsto 1\})$

e_{14}: $(l_8, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1)) \wedge x < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(x_2 = y_2)), \{l_4 \mapsto 1\})$

e_{15}: $(l_{12}, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1)) \wedge x < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(x_2 = y_2)), \{l_4 \mapsto 1\})$

e_{16}: $(l_4, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1)) \wedge x < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(\neg(x_2 = y_2))), \{l_4 \mapsto 2\})$

Interpolation and SAT-Based Model Checking

- McMillan: [9, CAV '03]
- Interpolation-based model checking (IMC)
  - Construct fixed points by interpolants derived from unsatisfiable BMC queries
  - Originally designed for finite-state systems (circuit); recently adopted for programs

Expressing IMC

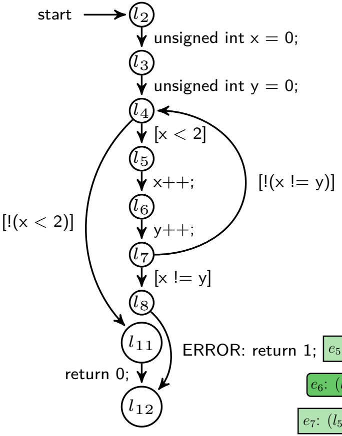
- Block Size (blk): blk^l

Furthermore:

- Use *block formulas* to partition BMC queries
  - Already recorded in predicate abstract state: (ψ, φ, σ)
- IMC algorithm (on top of CPA Algorithm):

```
k = 1
while !finished do
    run CPA Algorithm
    check feasibility of each abstract error state
    partition unsatisfiable BMC queries
    construct fixed points by interpolants
    k++
```

IMC: Example (error path to l_8 with one loop unrolling)



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

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

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

e_3: $(l_5, (\mathit{true}, x_0 < 2, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

e_4: $(l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

e_5: $(l_7, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

e_6: $(l_4, (\mathit{true}, \mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

e_7: $(l_5, (\mathit{true}, x_1 < 2, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

e_8: $(l_6, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

e_9: $(l_7, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

e_{10}: $(l_8, (\mathit{true}, \mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(x_2 = y_2)), \{l_4 \mapsto 1\})$

IMC: Example (error path to l_8 with one loop unrolling)

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

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

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

e_3: $(l_5, (\mathit{true}, x_0 < 2, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

e_4: $(l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

e_5: $(l_7, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

e_6: $(l_4, (\mathit{true}, \mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

e_7: $(l_5, (\mathit{true}, x_1 < 2, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

e_8: $(l_6, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

e_9: $(l_7, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

e_{10}: $(l_8, (\mathit{true}, \mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(x_2 = y_2)), \{l_4 \mapsto 1\})$

$$\underbrace{x_0 = 0 \wedge y_0 = 0 \wedge x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))}_{\text{Formula } A} \wedge$$

$$\underbrace{x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(x_2 = y_2)}_{\text{Formula } B}$$

interpolant: $x_1 = y_1$

IMC: Example (error path to l_8 with one loop unrolling)

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

$\downarrow$

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

$\downarrow$

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

$\downarrow$

e_3: $(l_5, (\mathit{true}, x_0 < 2, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

$\downarrow$

e_4: $(l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

$\downarrow$

e_5: $(l_7, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1, x_0 = 0 \wedge y_0 = 0), \{l_4 \mapsto 0\})$

$\downarrow$

e_6: $(l_4, (\mathit{true}, \mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

$\downarrow$

e_7: $(l_5, (\mathit{true}, x_1 < 2, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

$\downarrow$

e_8: $(l_6, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

$\downarrow$

e_9: $(l_7, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})$

$\downarrow$

e_{10}: $(l_8, (\mathit{true}, \mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(x_2 = y_2)), \{l_4 \mapsto 1\})$

$$\underbrace{x_0 = y_0 \wedge x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(\neg(x_1 = y_1))}_{\text{Formula } A} \wedge$$

$$\underbrace{x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(x_2 = y_2)}_{\text{Formula } B}$$ fixed point $x = y$ reached

Insights

- BMC naturally follows by increasing block size to whole (bounded) program

Insights

- BMC naturally follows by increasing block size to whole (bounded) program
- Difference between predicate abstraction and IMPACT:
  - BDDs vs. SMT-based formulas: costly abstractions vs. costly coverage checks
  - Recompute ARG vs. rechecking coverage
  - We know that only these differences are relevant!
  - Predicate abstraction pays for creating more general abstract model
  - IMPACT is lazier but this can lead to many refinements $\rightarrow$ forced covering or large blocks help

Evaluation: Usefulness of Framework CPAchecker

- 5 existing approaches successfully integrated
- Ongoing projects for integration of further approaches
- Interesting insights learned about these approaches
- High configurability allows new combinations and hybrid approaches
- Already used as base for other successful research projects

Evaluation: Usefulness of Implementation

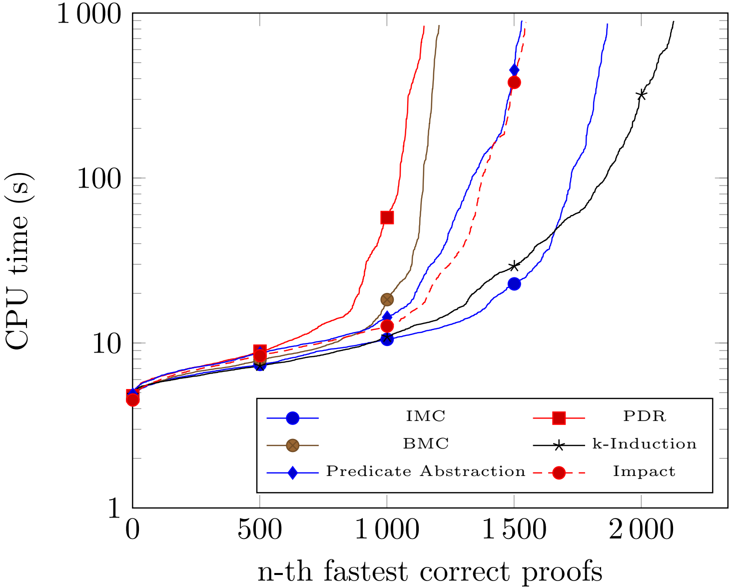
- Used in other research projects
- Used as part of many SV-COMP submissions, 27 gold medals

- Also competitive stand-alone
- Awarded Gödel medal by Kurt Gödel Society

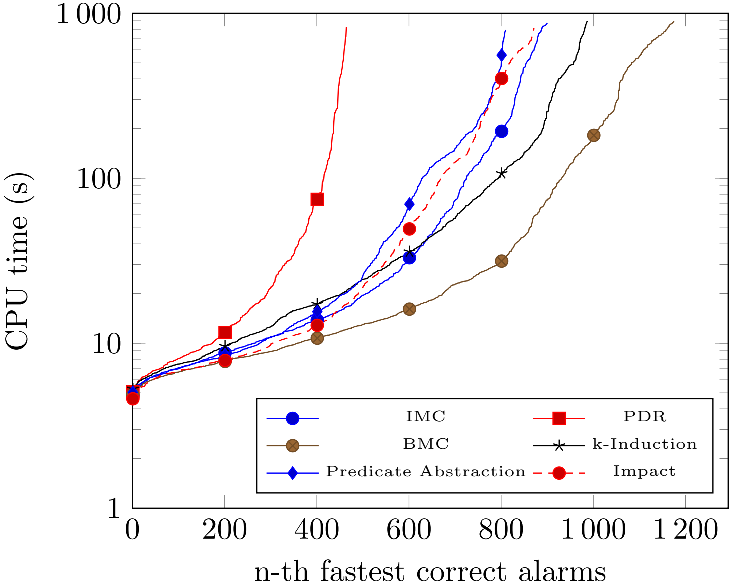
Experimental Evaluation

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

Quantile Plot: Safe Tasks



Quantile Plot: Unsafe Tasks



Experimental Comparison of Algorithms: Summary

We reconfirm that

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

SMT Solver Can Make a Difference

Now, which do you think is better, i.e., solves more tasks?

k-Induction

Predicate Abstraction

SMT Solver Can Make a Difference

Now, which do you think is better, i.e., solves more tasks?

(A)

k-Induction
solves $29\,\%$ more tasks

(B)

Predicate Abstraction
solves $3\,\%$ more tasks

SMT Solver Can Make a Difference

Now, which do you think is better, i.e., solves more tasks?

(A)	(B)
k-Induction solves $29\,\%$ more tasks	Predicate Abstraction solves $3\,\%$ more tasks
Z3 with bitprecise arithmetic	MathSAT5 with linear arithmetic

Depending on configuration, either (A) or (B) can be true!

Technical details (e.g., choice of SMT theory)
influence evaluation of algorithms

References I

[1] 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

[2] Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable software verification: Concretizing the convergence of model checking and program analysis. In: Proc. CAV. pp. 504–518. LNCS 4590, Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_51

[3] Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Proc. TACAS. pp. 193–207. LNCS 1579, Springer (1999). https://doi.org/10.1007/3-540-49059-0_14

[4] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643

[5] Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Proc. CAV. pp. 72–83. LNCS 1254, Springer (1997). https://doi.org/10.1007/3-540-63166-6_10

[6] 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

References II

[7] Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proc. POPL. pp. 58–70. ACM (2002). https://doi.org/10.1145/503272.503279

[8] 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

[9] 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

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