Predicate Analysis in CPAchecker

Dirk Beyer, Matthias Dangl, and Philipp Wendler

LMU Munich, Germany

2017-11-28

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, ...)

Open Problems

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

→ What are their core concepts and key differences?

Open Problems

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

→ 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

→ 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

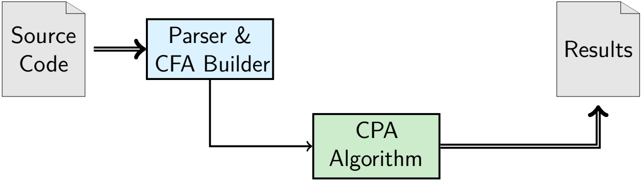
Base: Configurable Program Analysis

Configurable Program Analysis (CPA):

- Beyer, Henzinger, Théoduloz: [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



Source Code

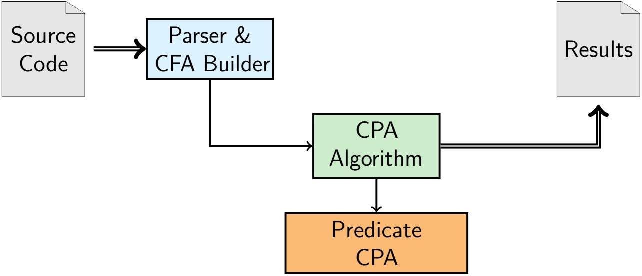
Parser & CFA Builder

CPA Algorithm

Results

Using the CPA Framework

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



Source Code

Parser & CFA Builder

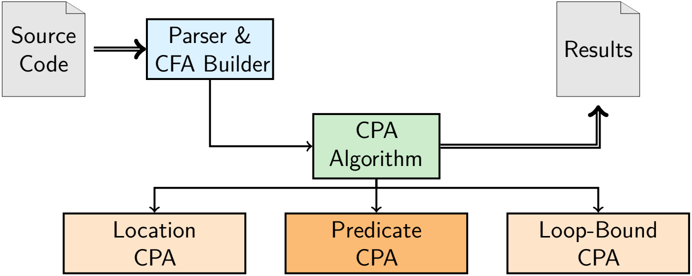
CPA Algorithm

Predicate CPA

Results

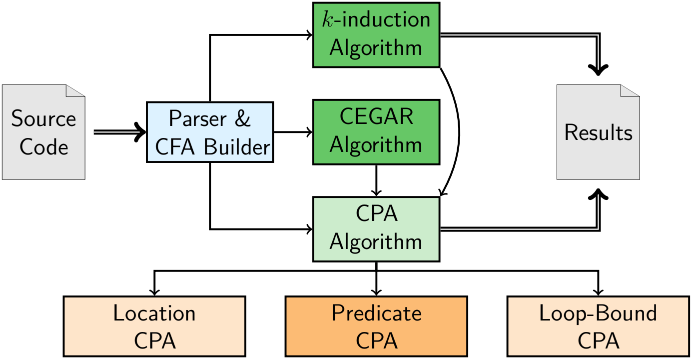
Using the CPA Framework

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

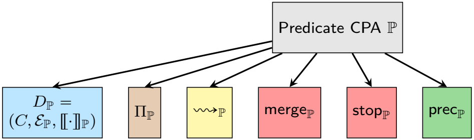


Using the CPA Framework

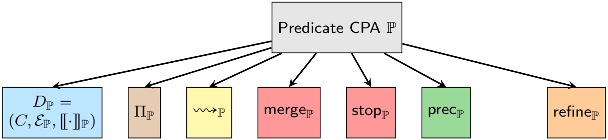
- CPA Algorithm is a configurable reachability analysis for arbitrary abstract domains
- Provide Predicate CPA for our predicate-based abstract domain
- Reuse other CPAs
- Built further algorithms on top that make use of reachability analysis



Predicate CPA



Predicate CPA



Predicate CPA $\mathbb{P}$

$D_{\mathbb{P}} = (C, \mathcal{E}_{\mathbb{P}}, [\![\cdot]\!]_{\mathbb{P}})$

$\Pi_{\mathbb{P}}$

$\rightsquigarrow_{\mathbb{P}}$

$\mathsf{merge}_{\mathbb{P}}$

$\mathsf{stop}_{\mathbb{P}}$

$\mathsf{prec}_{\mathbb{P}}$

$\mathsf{refine}_{\mathbb{P}}$

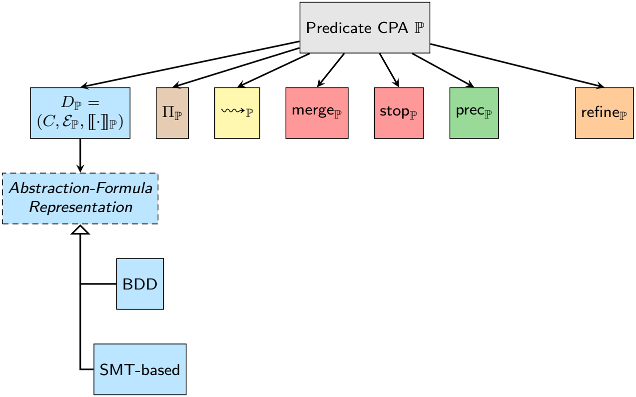
Predicate CPA: Abstract Domain

- Abstract state: (ψ, φ)
  - tuple of abstraction formula ψ and path formula φ (for ABE)
  - conjunctions 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)
  - conjunctions 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 $\mathbb{P}$

$D_{\mathbb{P}} = (C, \mathcal{E}_{\mathbb{P}}, [\![\cdot]\!]_{\mathbb{P}})$

$\Pi_{\mathbb{P}}$

$\rightsquigarrow_{\mathbb{P}}$

merge$_{\mathbb{P}}$

stop$_{\mathbb{P}}$

prec$_{\mathbb{P}}$

refine$_{\mathbb{P}}$

Abstraction-Formula Representation

BDD

SMT-based

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

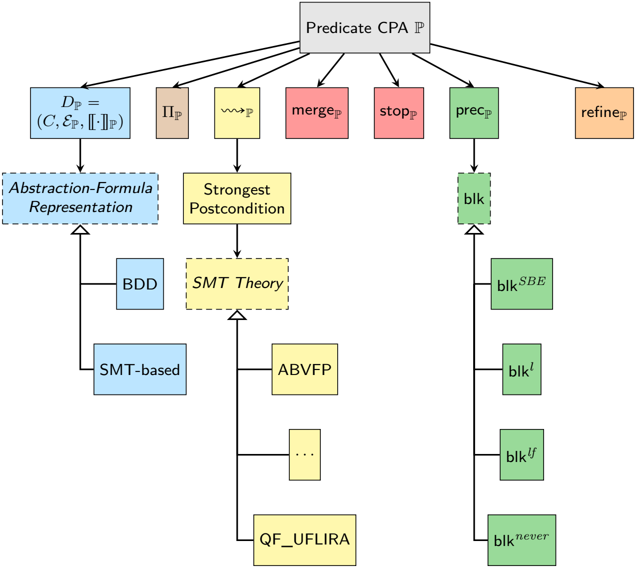
Predicate CPA: CPA Operators

- 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 does something at block ends (as determined by blk)
  - computes abstraction of current abstract state
  - new abstract state is (ψ', true)

Predicate CPA



Predicate CPA: Refinement

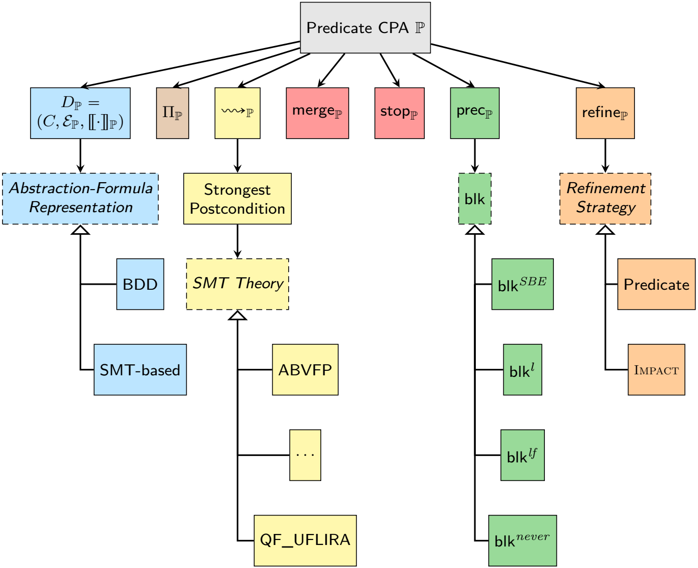
Four steps:

1. Reconstruct ARG path to abstract error state
2. Check feasibility of path
3. Compute interpolants along path
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
  - 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
  - Abstraction formula as BDD

Expressing Predicate Abstraction

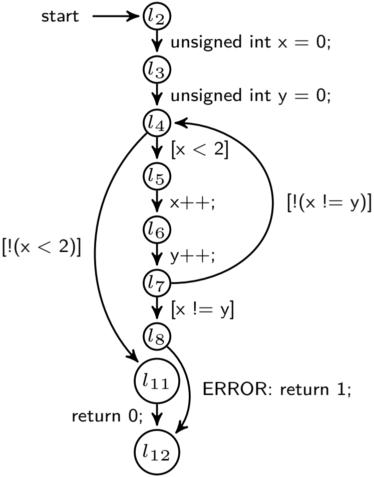
- Abstraction Formulas: BDDs
- Block Size (blk): e.g. blk^{SBE} or blk^{l} or blk^{lf}
- Refinement Strategy: add predicates to precision, cut ARG

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 $true$
```

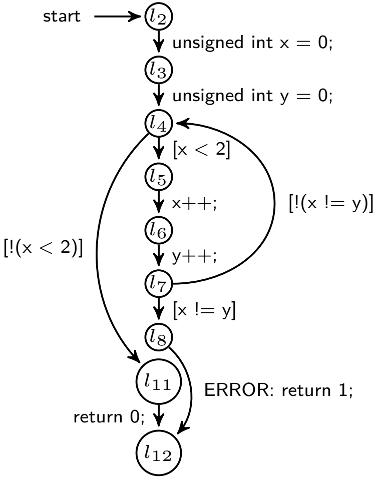
Example Program

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



Predicate Abstraction: Example

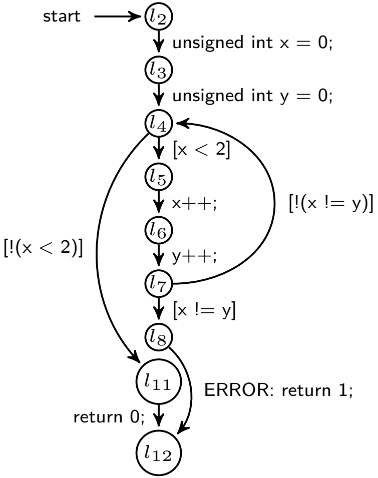
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

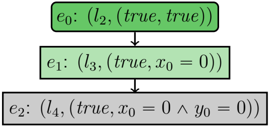


e_0: $(l_2, (true, true))$

Predicate Abstraction: Example

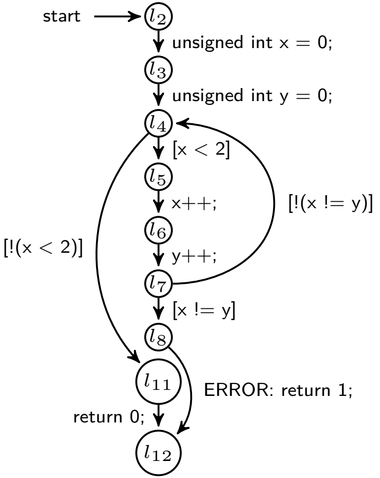
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

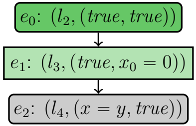




Predicate Abstraction: Example

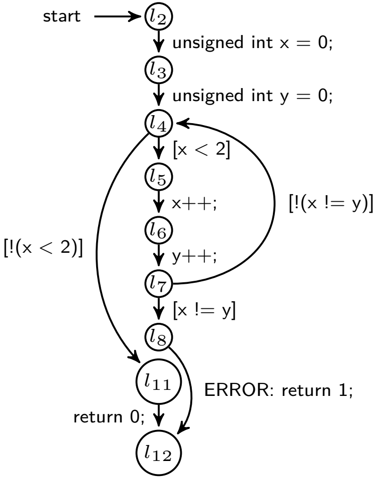
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

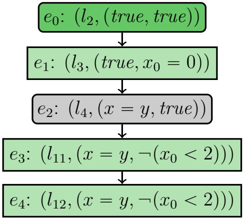




Predicate Abstraction: Example

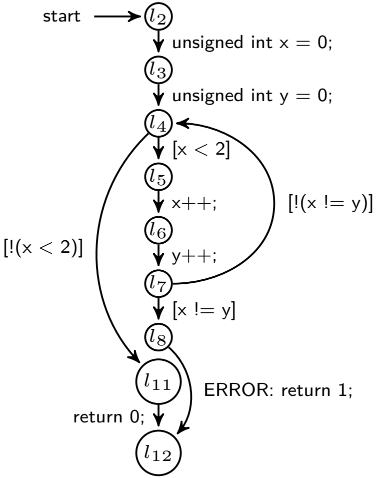
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

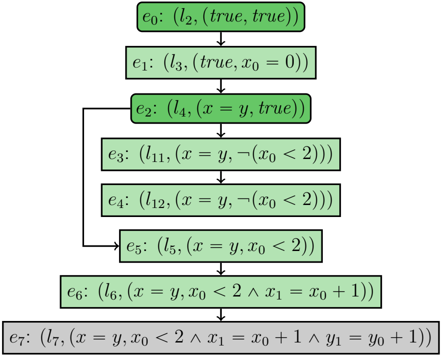




Predicate Abstraction: Example

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





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (x = y, \mathit{true}))$

e_3: $(l_{11}, (x = y, \neg(x_0 < 2)))$

e_4: $(l_{12}, (x = y, \neg(x_0 < 2)))$

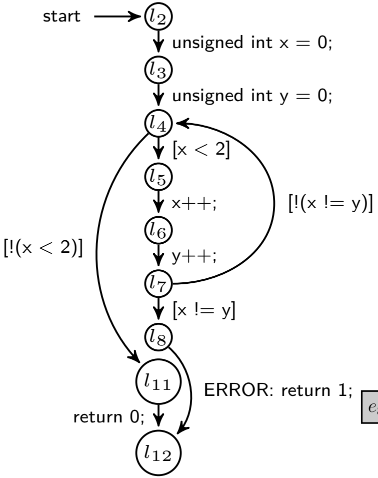
e_5: $(l_5, (x = y, x_0 < 2))$

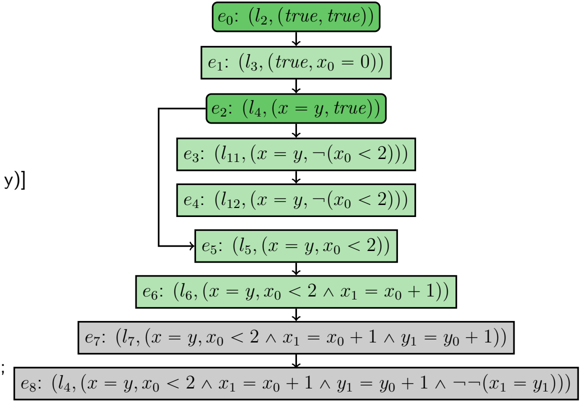
e_6: $(l_6, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1))$

e_7: $(l_7, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

Predicate Abstraction: Example

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





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

e_1: $(l_3, (true, x_0 = 0))$

e_2: $(l_4, (x = y, true))$

e_3: $(l_{11}, (x = y, \neg(x_0 < 2)))$

e_4: $(l_{12}, (x = y, \neg(x_0 < 2)))$

e_5: $(l_5, (x = y, x_0 < 2))$

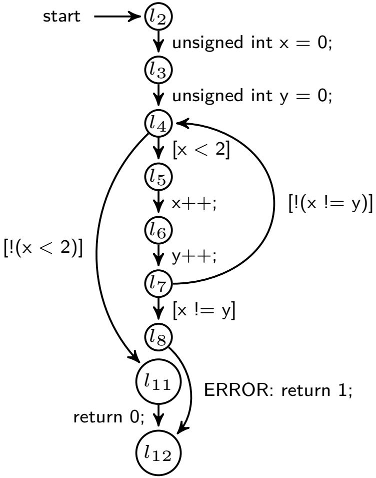
e_6: $(l_6, (x = y, x_0 < 2 \land x_1 = x_0 + 1))$

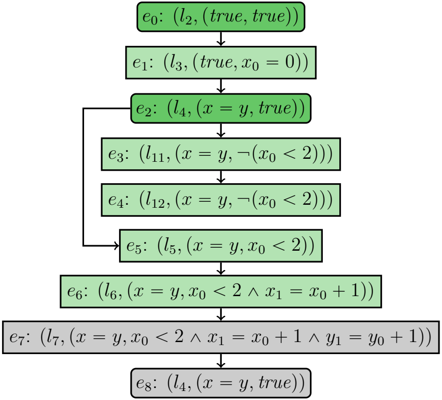
e_7: $(l_7, (x = y, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1))$

e_8: $(l_4, (x = y, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg\neg(x_1 = y_1)))$

Predicate Abstraction: Example

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





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

e_1: $(l_3, (true, x_0 = 0))$

e_2: $(l_4, (x = y, true))$

e_3: $(l_{11}, (x = y, \neg(x_0 < 2)))$

e_4: $(l_{12}, (x = y, \neg(x_0 < 2)))$

e_5: $(l_5, (x = y, x_0 < 2))$

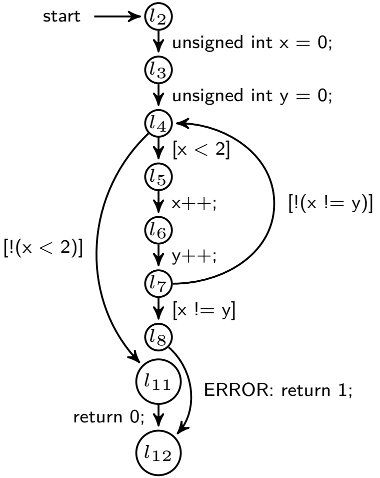
e_6: $(l_6, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1))$

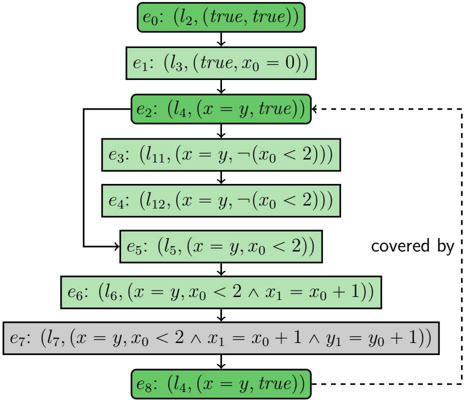
e_7: $(l_7, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

e_8: $(l_4, (x = y, true))$

Predicate Abstraction: Example

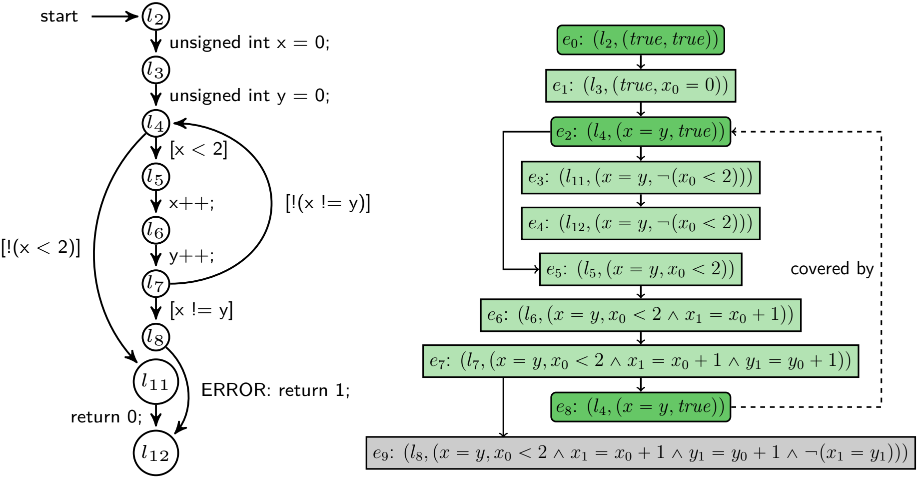
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$





Predicate Abstraction: Example

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



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

e_1: $(l_3, (true, x_0 = 0))$

e_2: $(l_4, (x = y, true))$

e_3: $(l_{11}, (x = y, \neg(x_0 < 2)))$

e_4: $(l_{12}, (x = y, \neg(x_0 < 2)))$

e_5: $(l_5, (x = y, x_0 < 2))$

e_6: $(l_6, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1))$

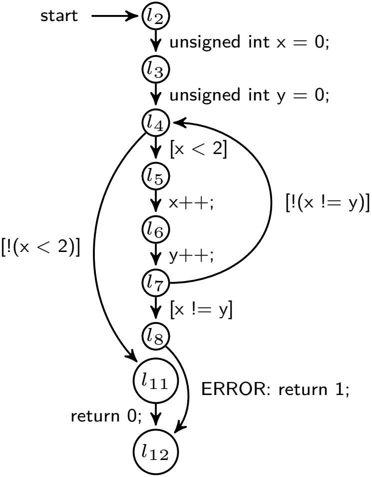
e_7: $(l_7, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

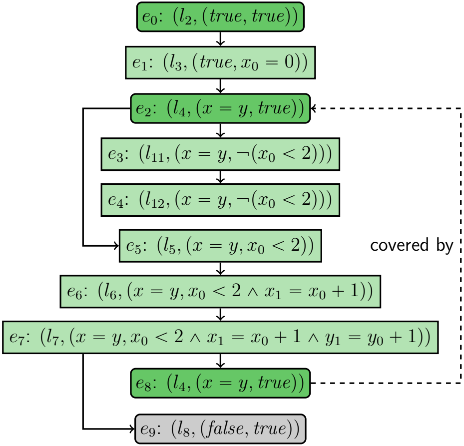
e_8: $(l_4, (x = y, true))$

e_9: $(l_8, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1 \wedge \neg(x_1 = y_1)))$

Predicate Abstraction: Example

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




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

e_1: $(l_3, (true, x_0 = 0))$

e_2: $(l_4, (x = y, true))$

e_3: $(l_{11}, (x = y, \neg(x_0 < 2)))$

e_4: $(l_{12}, (x = y, \neg(x_0 < 2)))$

e_5: $(l_5, (x = y, x_0 < 2))$

e_6: $(l_6, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1))$

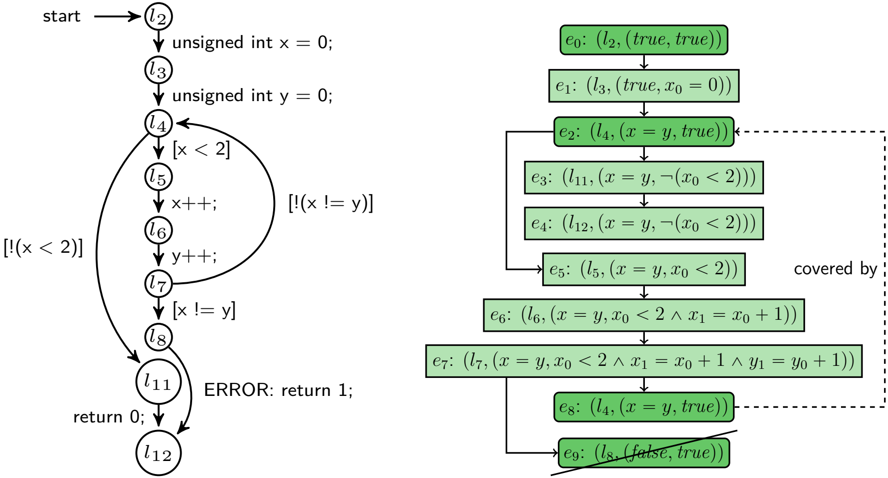
e_7: $(l_7, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

e_8: $(l_4, (x = y, true))$

e_9: $(l_8, (false, true))$

Predicate Abstraction: Example

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



e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (x = y, \mathit{true}))$

e_3: $(l_{11}, (x = y, \neg(x_0 < 2)))$

e_4: $(l_{12}, (x = y, \neg(x_0 < 2)))$

e_5: $(l_5, (x = y, x_0 < 2))$

e_6: $(l_6, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1))$

e_7: $(l_7, (x = y, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

e_8: $(l_4, (x = y, \mathit{true}))$

~~e_9: $(l_8, (\mathit{false}, \mathit{true}))$~~

IMPACT

- IMPACT
  - "Lazy Abstraction with Interpolants"
  - McMillan: [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
  - Quick exploration of the state space

Expressing IMPACT

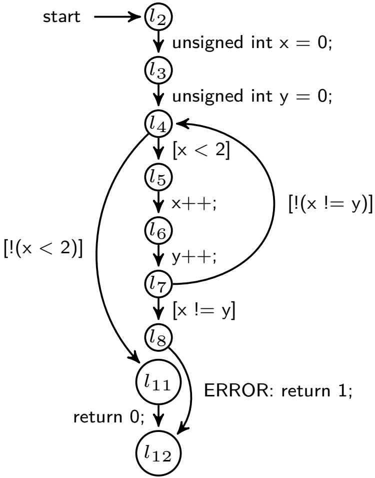
- Abstraction Formulas: SMT-based
- Block Size (blk): blk^{SBE} or other (new!)
- Refinement Strategy:
  conjoin interpolants to abstract states,
  recheck coverage relation

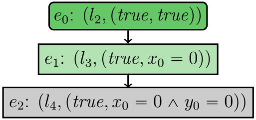
Furthermore:

- Use CEGAR Algorithm
- Precision stays empty
  → predicate abstraction never computed

IMPACT: Example

with blk^l





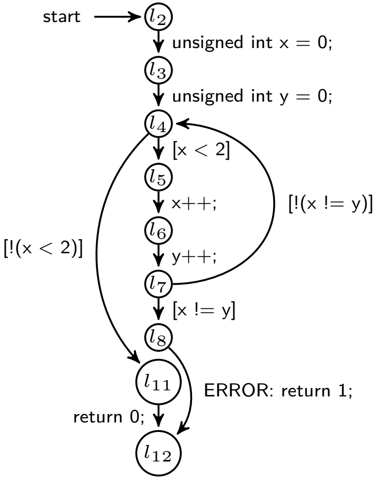
e_0: $(l_2, (\mathit{true}, \mathit{true}))$

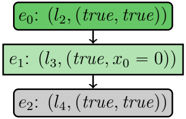
e_1: $(l_3, (\mathit{true}, x_0 = 0))$

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

IMPACT: Example

with blk^l





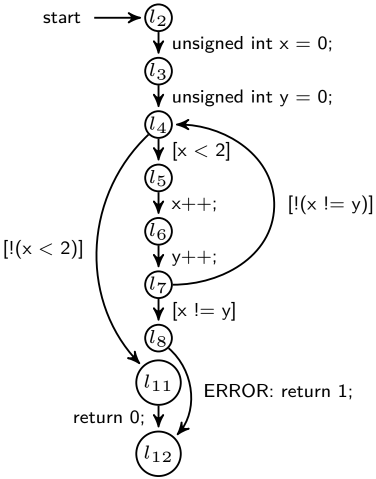
e_0: $(l_2, (\mathit{true}, \mathit{true}))$

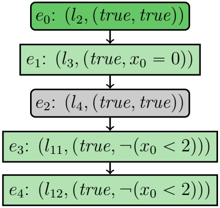
e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (\mathit{true}, \mathit{true}))$

IMPACT: Example

with blk^l





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

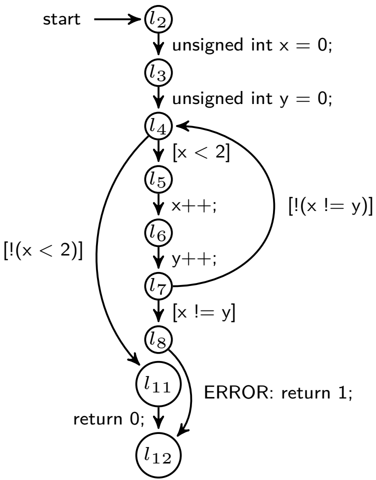
e_2: $(l_4, (\mathit{true}, \mathit{true}))$

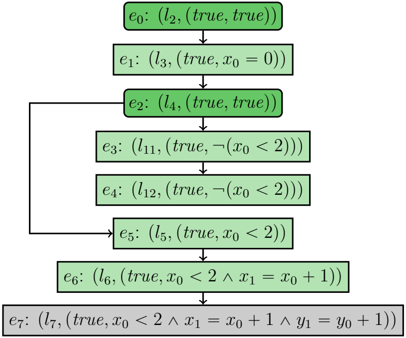
e_3: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

e_4: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

IMPACT: Example

with blk^l





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (\mathit{true}, \mathit{true}))$

e_3: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

e_4: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

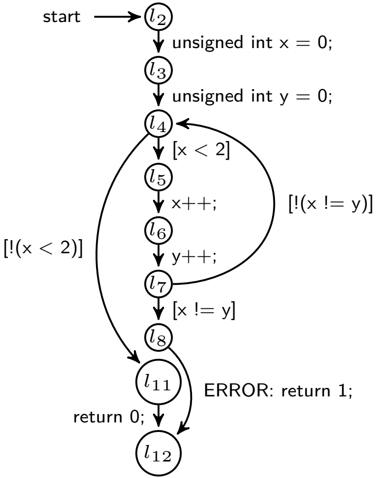
e_5: $(l_5, (\mathit{true}, x_0 < 2))$

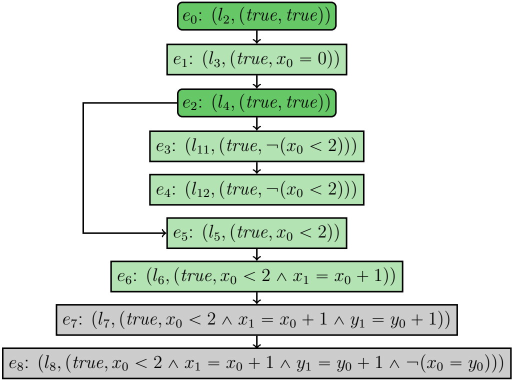
e_6: $(l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1))$

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

IMPACT: Example

with blk^l





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (\mathit{true}, \mathit{true}))$

e_3: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

e_4: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

e_5: $(l_5, (\mathit{true}, x_0 < 2))$

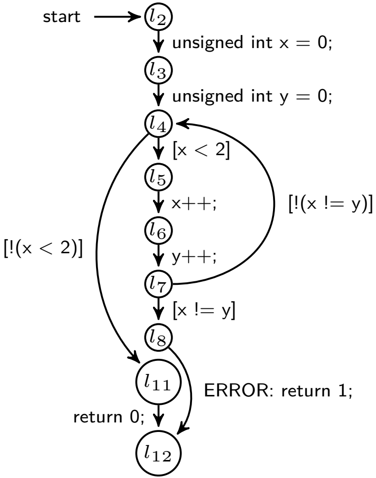
e_6: $(l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1))$

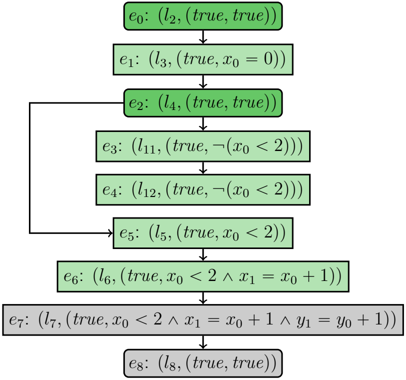
e_7: $(l_7, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

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

IMPACT: Example

with blk^l





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (\mathit{true}, \mathit{true}))$

e_3: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

e_4: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

e_5: $(l_5, (\mathit{true}, x_0 < 2))$

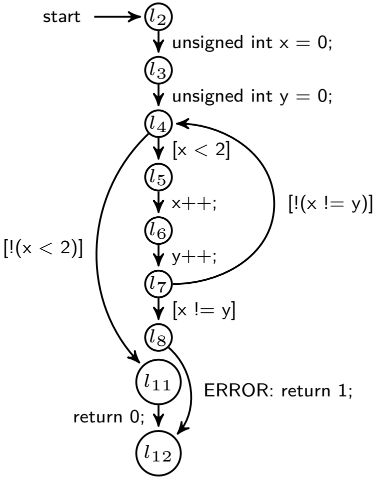
e_6: $(l_6, (\mathit{true}, x_0 < 2 \land x_1 = x_0 + 1))$

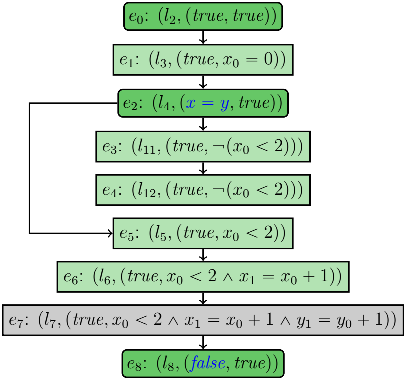
e_7: $(l_7, (\mathit{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1))$

e_8: $(l_8, (\mathit{true}, \mathit{true}))$

IMPACT: Example

with blk^l





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (x = y, \mathit{true}))$

e_3: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

e_4: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

e_5: $(l_5, (\mathit{true}, x_0 < 2))$

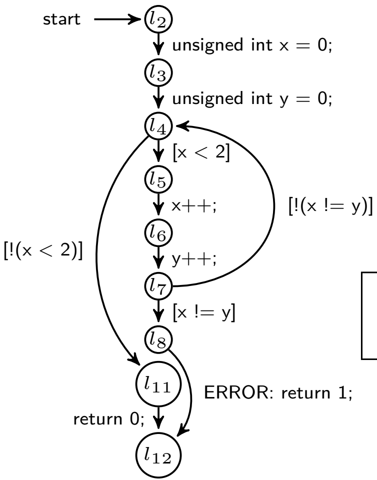
e_6: $(l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1))$

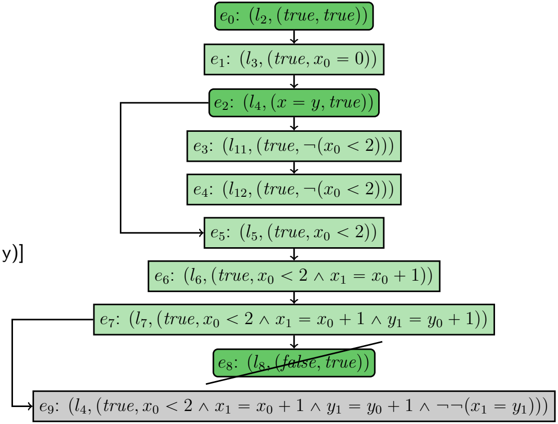
e_7: $(l_7, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

e_8: $(l_8, (\mathit{false}, \mathit{true}))$

IMPACT: Example

with blk^l





$e_0\colon\ (l_2, (\mathit{true}, \mathit{true}))$

$e_1\colon\ (l_3, (\mathit{true}, x_0 = 0))$

$e_2\colon\ (l_4, (x = y, \mathit{true}))$

$e_3\colon\ (l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

$e_4\colon\ (l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

$e_5\colon\ (l_5, (\mathit{true}, x_0 < 2))$

$e_6\colon\ (l_6, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1))$

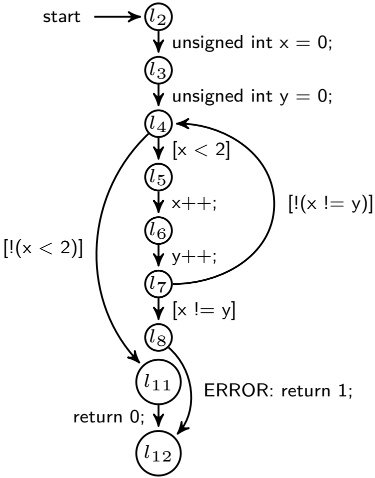
$e_7\colon\ (l_7, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

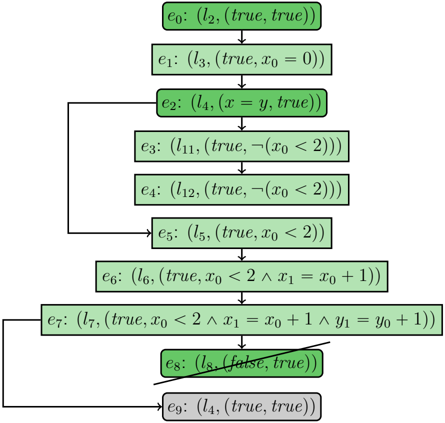
$\cancel{e_8\colon\ (l_8, (\mathit{false}, \mathit{true}))}$

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

IMPACT: Example

with blk^l





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (x = y, \mathit{true}))$

e_3: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

e_4: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

e_5: $(l_5, (\mathit{true}, x_0 < 2))$

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

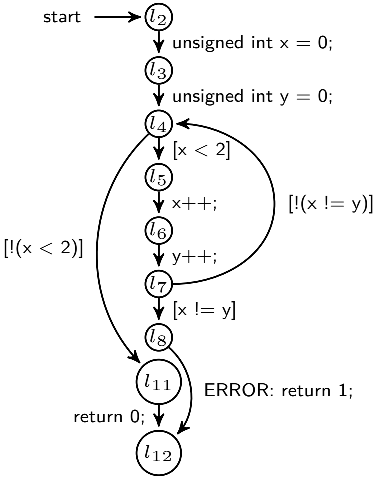
e_7: $(l_7, (\mathit{true}, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

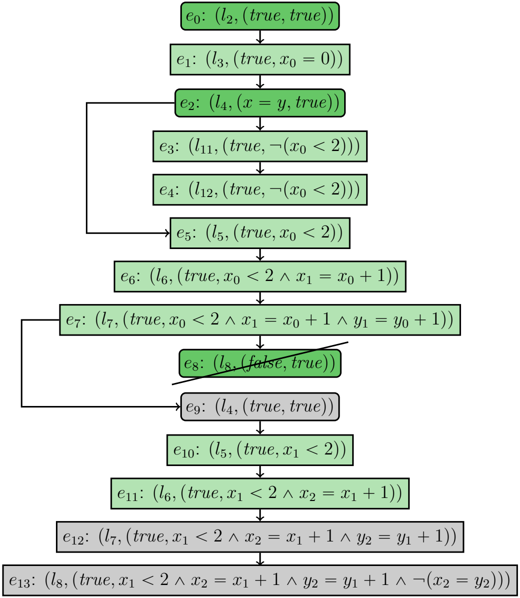
~~e_8: $(l_8, (\mathit{false}, \mathit{true}))$~~

e_9: $(l_4, (\mathit{true}, \mathit{true}))$

IMPACT: Example

with blk^l





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

e_1: $(l_3, (true, x_0 = 0))$

e_2: $(l_4, (x = y, true))$

e_3: $(l_{11}, (true, \neg(x_0 < 2)))$

e_4: $(l_{12}, (true, \neg(x_0 < 2)))$

e_5: $(l_5, (true, x_0 < 2))$

e_6: $(l_6, (true, x_0 < 2 \wedge x_1 = x_0 + 1))$

e_7: $(l_7, (true, x_0 < 2 \wedge x_1 = x_0 + 1 \wedge y_1 = y_0 + 1))$

~~e_8: $(l_8, (false, true))$~~

e_9: $(l_4, (true, true))$

e_{10}: $(l_5, (true, x_1 < 2))$

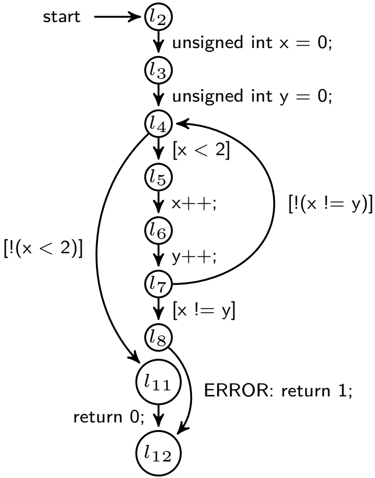
e_{11}: $(l_6, (true, x_1 < 2 \wedge x_2 = x_1 + 1))$

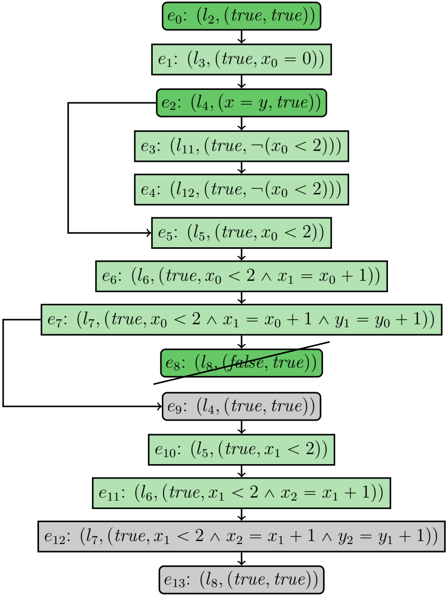
e_{12}: $(l_7, (true, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1))$

e_{13}: $(l_8, (true, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1 \wedge \neg(x_2 = y_2)))$

IMPACT: Example

with blk^l





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (x = y, \mathit{true}))$

e_3: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

e_4: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

e_5: $(l_5, (\mathit{true}, x_0 < 2))$

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

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

~~e_8: $(l_8, (\mathit{false}, \mathit{true}))$~~

e_9: $(l_4, (\mathit{true}, \mathit{true}))$

e_{10}: $(l_5, (\mathit{true}, x_1 < 2))$

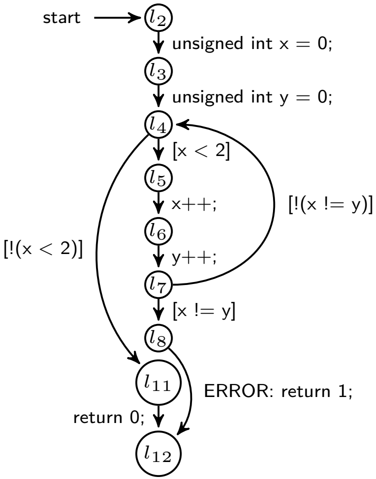
e_{11}: $(l_6, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1))$

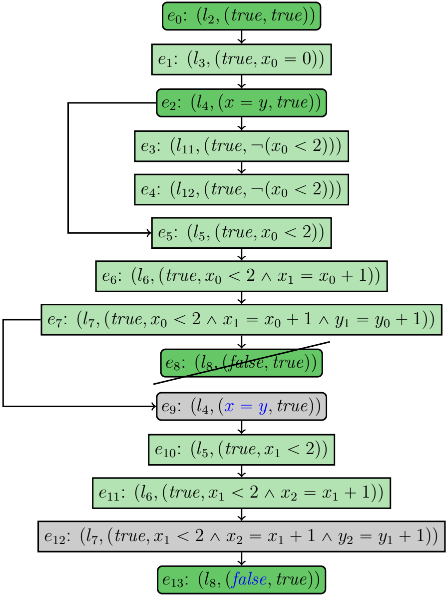
e_{12}: $(l_7, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1))$

e_{13}: $(l_8, (\mathit{true}, \mathit{true}))$

IMPACT: Example

with blk^{l}





e_0: $(l_2, (\mathit{true}, \mathit{true}))$

e_1: $(l_3, (\mathit{true}, x_0 = 0))$

e_2: $(l_4, (x = y, \mathit{true}))$

e_3: $(l_{11}, (\mathit{true}, \neg(x_0 < 2)))$

e_4: $(l_{12}, (\mathit{true}, \neg(x_0 < 2)))$

e_5: $(l_5, (\mathit{true}, x_0 < 2))$

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

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

~~e_8: $(l_8, (\mathit{false}, \mathit{true}))$~~

e_9: $(l_4, (x = y, \mathit{true}))$

e_{10}: $(l_5, (\mathit{true}, x_1 < 2))$

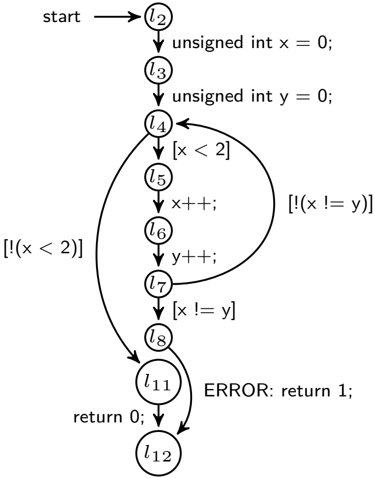
e_{11}: $(l_6, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1))$

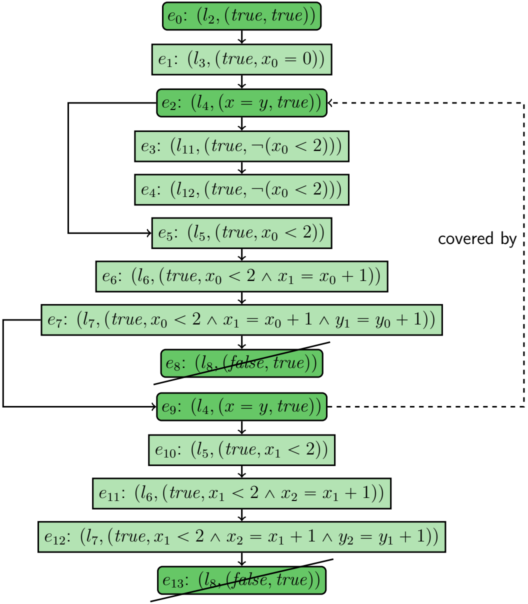
e_{12}: $(l_7, (\mathit{true}, x_1 < 2 \wedge x_2 = x_1 + 1 \wedge y_2 = y_1 + 1))$

e_{13}: $(l_8, (\mathit{false}, \mathit{true}))$

IMPACT: Example

with blk^l





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:

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

Expressing BMC

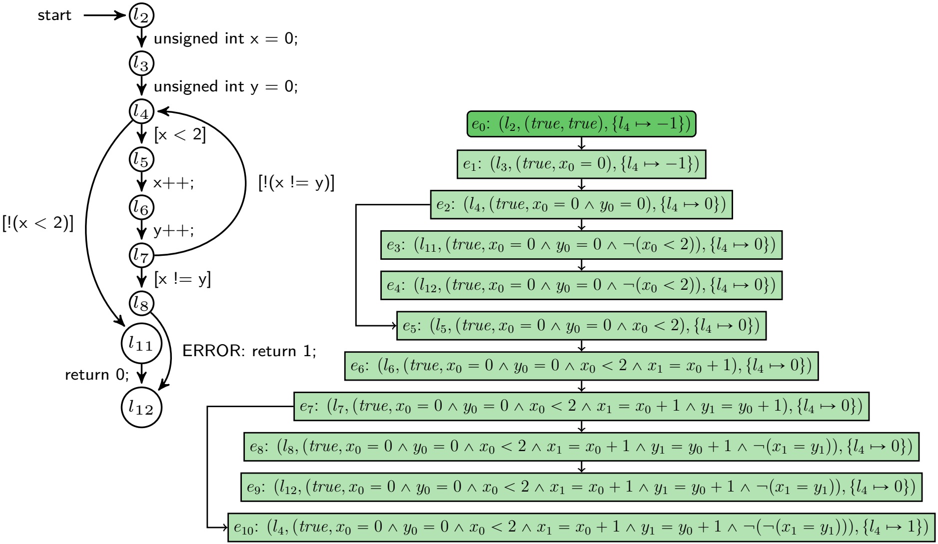
- Block Size (blk): blk^{never}

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
    $k{+}{+}$
```

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

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

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

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

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

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

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

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

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

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 → forced covering or large blocks help

SMT Study: Motivation

Which do you think is “better”,
i.e., solves more SV-COMP tasks?

- k-Induction
- Predicate abstraction

SMT Study: Motivation

- Predicate abstraction solves 3 % more tasks than k-induction

SMT Study: Motivation

- Predicate abstraction solves 3 % more tasks than k-induction
- k-Induction solves 29 % more tasks than predicate abstraction

SMT Study: Motivation

- Predicate abstraction solves 3 % more tasks than k-induction:
  MathSAT5 with linear arithmetic
- k-Induction solves 29 % more tasks than predicate abstraction:
  Z3 with bitprecise arithmetic

Comparison of SMT Solvers and Theories

- Which SMT solver should CPAchecker use by default?
- Which formula encoding?
- Which of these should we use for benchmarks in papers?

Comparison of SMT Solvers and Theories

- Which SMT solver should CPAchecker use by default?
- Which formula encoding?
- Which of these should we use for benchmarks in papers?

- Large study made possible by our framework
- Produced some interesting insights
- Prepare for changes in CPAchecker

Comparison of SMT Solvers and Theories

- Which SMT solver should CPAchecker use by default?
- Which formula encoding?
- Which of these should we use for benchmarks in papers?

- Large study made possible by our framework
- Produced some interesting insights
- Prepare for changes in CPAchecker

- SV-COMP'17 benchmark set
  (only reachability, without recursion and concurrency)
- 5594 verification tasks
- 15 min time limit, 15 GB memory limit
- On Apollon cluster

SMT Study: 120 Configurations

BMC | k-Induction | Impact | Pred. Abs

×

MathSAT5 | Princess | SMTInterpol | Z3

×

Bitprecise | Linear | Linear unsound

×

with Quantifiers | Quantifier-free

×

Arrays | UFs

Point of View: SMT Solvers

- Princess is never competitive
- Interpolation in Z3 is unmaintained since 2015
- Bitvector interpolation in Z3 produces up to 24 % crashes
- MathSAT5 has known interpolation problem for bitvectors, but problem occurs rarely

Point of View: Theories and Encodings

- Unsound linear encoding always the easiest (as expected)

Point of View: Theories and Encodings

- Unsound linear encoding always the easiest (as expected)
- Correctness as expected:
  BV $>$ sound LIRA $>$ unsound LIRA

Point of View: Theories and Encodings

- Unsound linear encoding always the easiest (as expected)
- Correctness as expected:
  BV $>$ sound LIRA $>$ unsound LIRA
- Effectivity for $\mathrm{Z3}$ as expected:
  BV $<$ sound LIRA $<$ unsound LIRA

Point of View: Theories and Encodings

- Unsound linear encoding always the easiest (as expected)
- Correctness as expected:
  BV $>$ sound LIRA $>$ unsound LIRA
- Effectivity for Z3 as expected:
  BV $<$ sound LIRA $<$ unsound LIRA
- Effectivity for MathSAT5:
  sound LIRA $<$ BV $\approx$ unsound LIRA
  (but BV needs more CPU time)

$\Rightarrow$ MathSAT5 is really good with bitvectors.

Point of View: Theories and Encodings

- Unsound linear encoding always the easiest (as expected)
- Correctness as expected:
  BV $>$ sound LIRA $>$ unsound LIRA
- Effectivity for Z3 as expected:
  BV $<$ sound LIRA $<$ unsound LIRA
- Effectivity for MathSAT5:
  sound LIRA $<$ BV $\approx$ unsound LIRA
  (but BV needs more CPU time)
- Effectivity for SMTInterpol:
  sound LIRA $\ll$ unsound LIRA

$\Rightarrow$ MathSAT5 is really good with bitvectors.

Point of View: Theories and Encodings

- Unsound linear encoding always the easiest (as expected)
- Correctness as expected:
  BV $>$ sound LIRA $>$ unsound LIRA
- Effectivity for Z3 as expected:
  BV $<$ sound LIRA $<$ unsound LIRA
- Effectivity for MathSAT5:
  sound LIRA $<$ BV $\approx$ unsound LIRA
  (but BV needs more CPU time)
- Effectivity for SMTInterpol:
  sound LIRA $\ll$ unsound LIRA

$\Rightarrow$ MathSAT5 is really good with bitvectors.

$\Rightarrow$ Sound LIRA encoding rarely makes sense.

Point of View: Algorithms

- Mostly, the best configurations of MathSAT5, SMTInterpol, and Z3 are close for each algorithm
    - Gives confidence for valid comparison of algorithm
    - But outlier exists:
      Z3 is worse than others for predicate abstraction

Point of View: Algorithms

- Mostly, the best configurations of MathSAT5, SMTInterpol, and Z3 are close for each algorithm
  - Gives confidence for valid comparison of algorithm
  - But outlier exists:
    Z3 is worse than others for predicate abstraction
- Predicate abstraction and Impact suffer most from disjunctions of sound LIRA encoding.

Point of View: Arrays and Quantifiers

- Little difference with/without arrays/quantifiers

⇒ Arrays don't hurt:
we should find a way to get better array predicates

Point of View: Arrays and Quantifiers

- Little difference with/without arrays/quantifiers

⇒ Arrays don't hurt:
we should find a way to get better array predicates

- But quantifiers would restrict solver choice too much (PRINCESS and Z3)

SMT Study: Final Conclusions

- Choice of theories, solver, and encoding details affects comparisons of algorithms!
- For now:
  use MathSAT5 with bitvectors and arrays if possible
  - Upcoming default for CPAchecker
  - Possible problems for users: license, native binary
  - Next-best choice:
    SMTInterpol with unsound linear arithmetic
  - No improvement of situation in sight