Predicate-Based Model Checking

Dirk Beyer

LMU Munich, Germany

Based on:

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
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, ...)

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

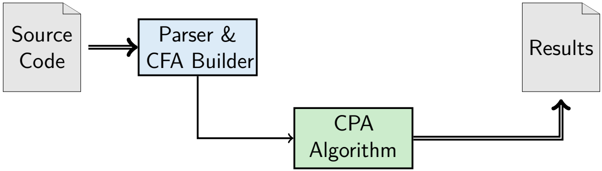
Base: Configurable Program Analysis

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, ...

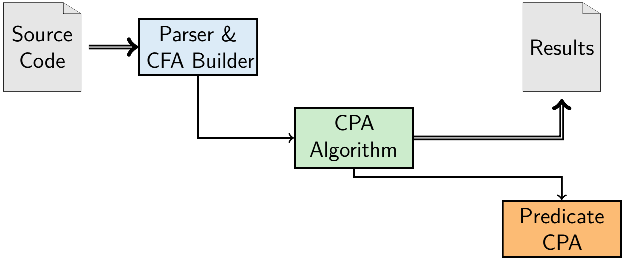
Using the CPA Framework

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



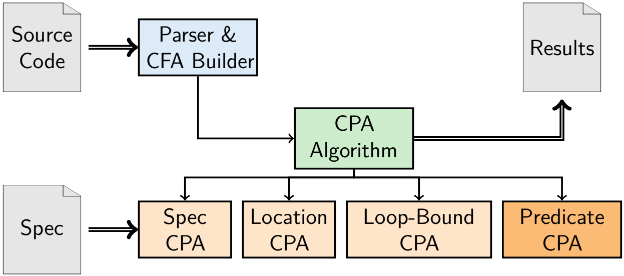
Using the CPA Framework

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



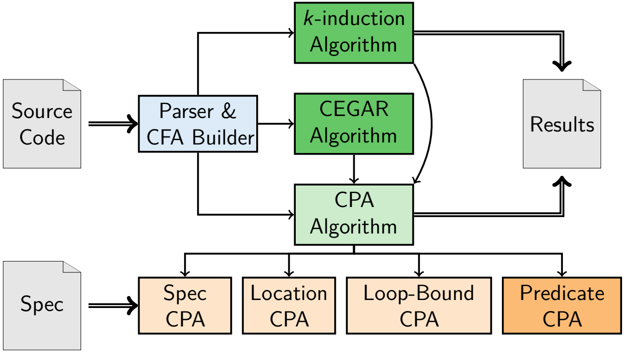
Using the CPA Framework

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

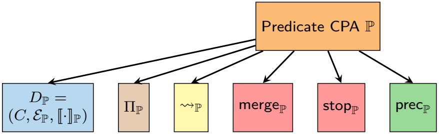


Using the CPA Framework

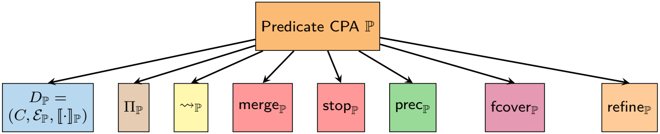
- CPA Algorithm is a configurable reachability analysis for arbitrary abstract domains
- Provide Predicate CPA for our predicate-based abstract domain
- Reuse other CPAs
- Built further algorithms on top that make use of reachability analysis



Predicate CPA



Predicate CPA



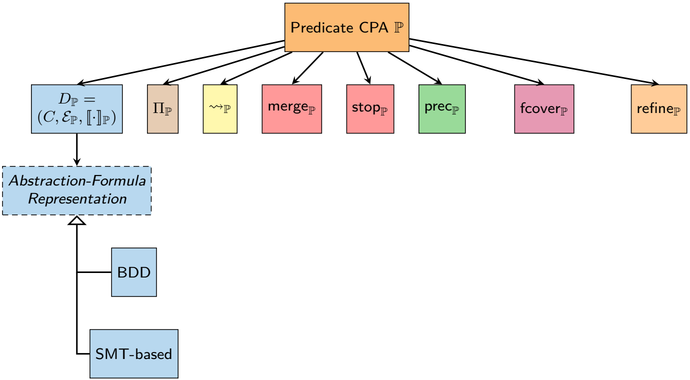
Predicate CPA: Abstract Domain

- Abstract state: (ψ, φ)
  - tuple of abstraction formula ψ and path formula φ (for ABE)
  - 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: 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

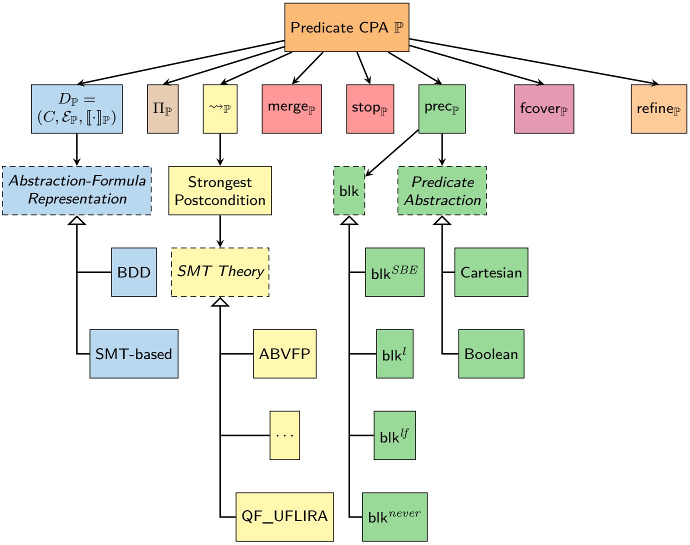
Predicate CPA: CPA Operators

- 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 (ψ', true)

Predicate CPA



Predicate CPA: Refinement

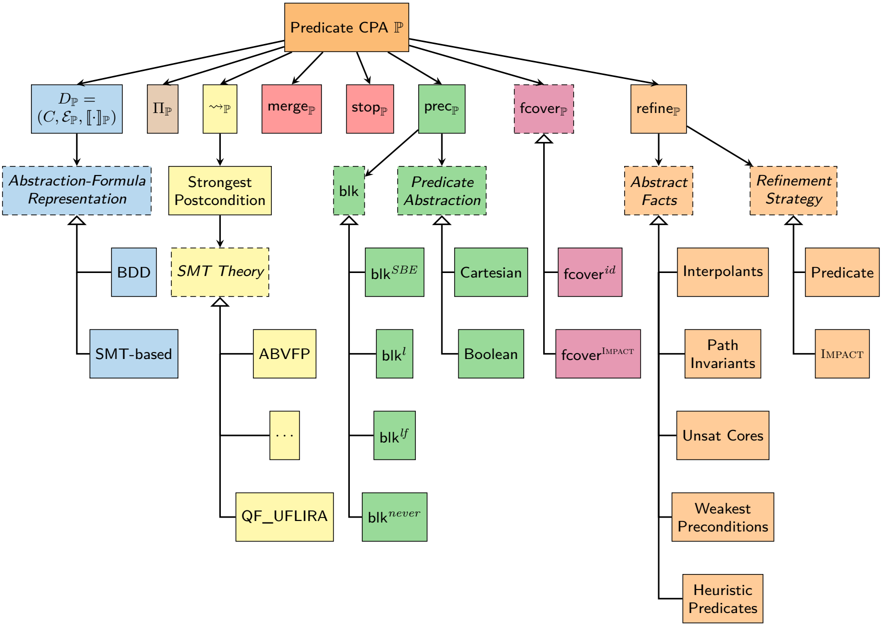
Four steps:

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
  - [CAV'97, POPL'02, J. ACM'03, POPL'04]
  - 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

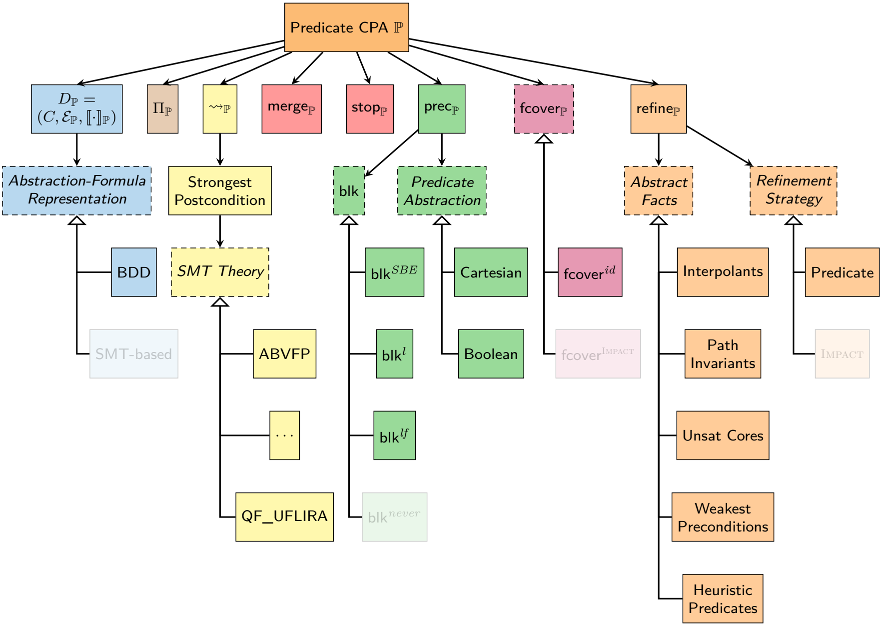
Expressing Predicate Abstraction

- 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:

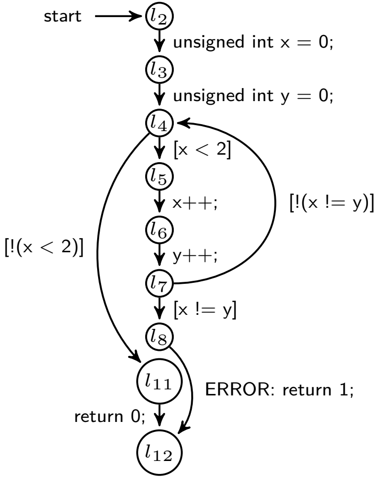
```
while true do
    run CPA Algorithm
    if target state found then
        call refine
        if target state reachable then
            return false
    else
        return true
```

Predicate CPA

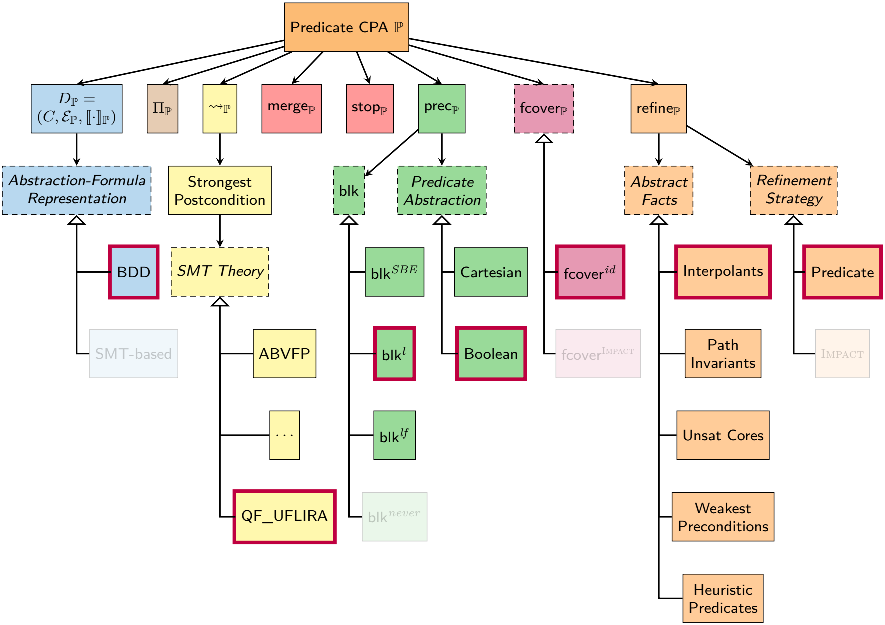


Example Program

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

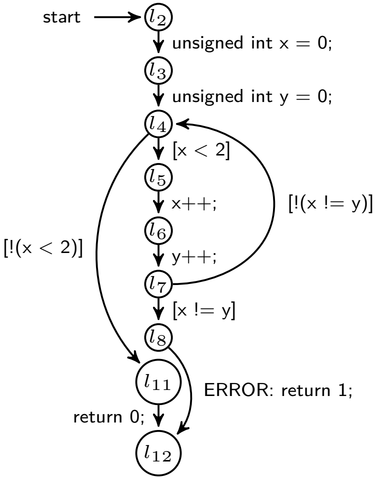


Predicate CPA



Predicate Abstraction: Example

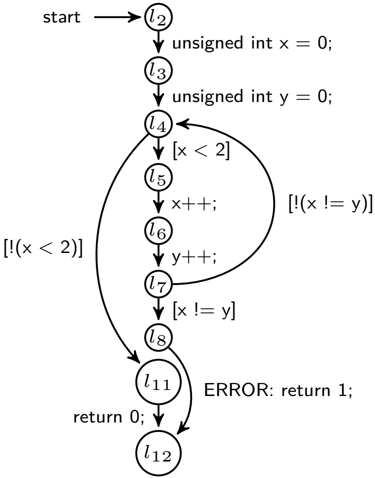
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

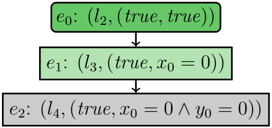


e_0: $(l_2, (true, true))$

Predicate Abstraction: Example

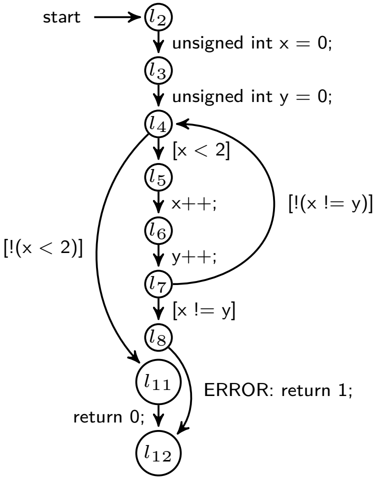
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

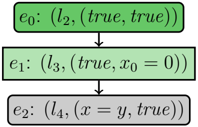




Predicate Abstraction: Example

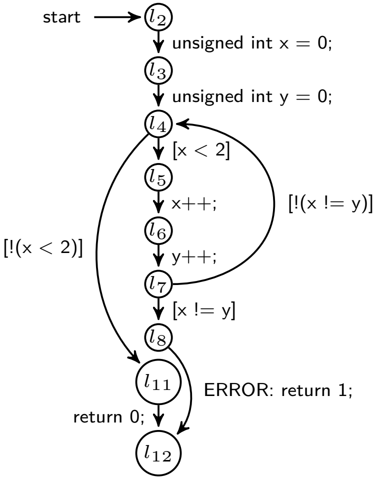
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

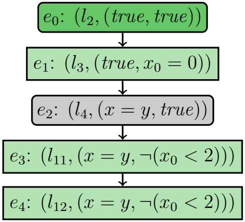




Predicate Abstraction: Example

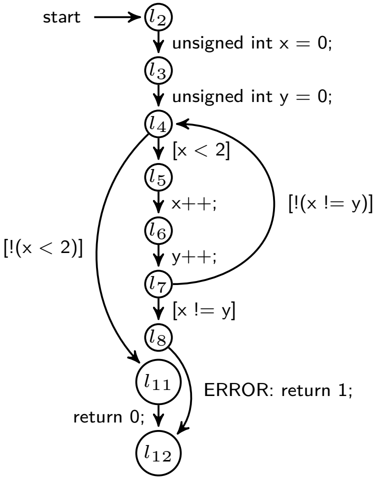
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

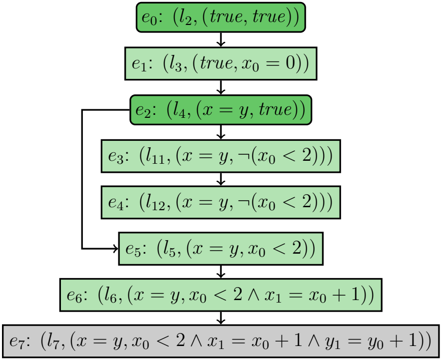




Predicate Abstraction: Example

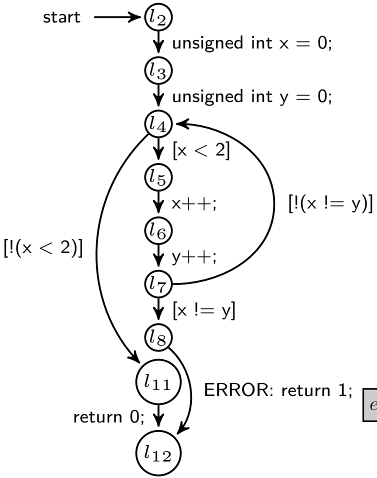
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$

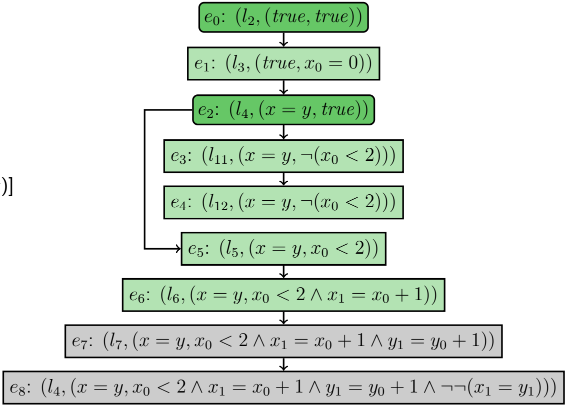




Predicate Abstraction: Example

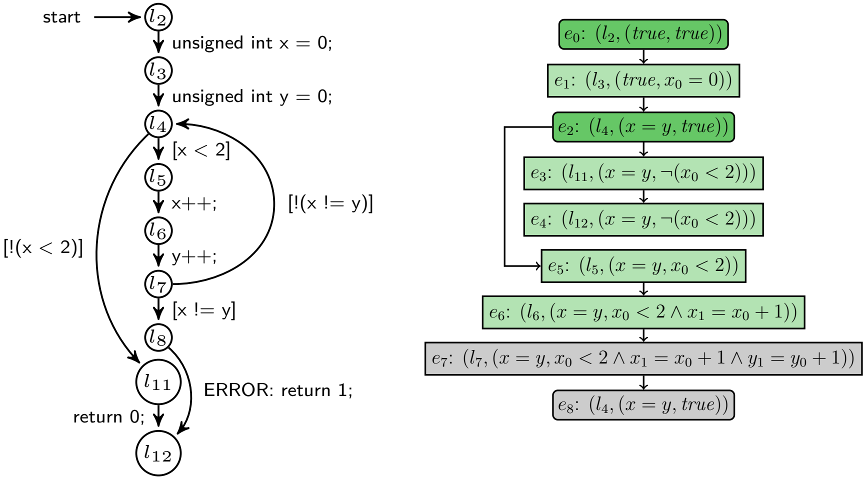
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$





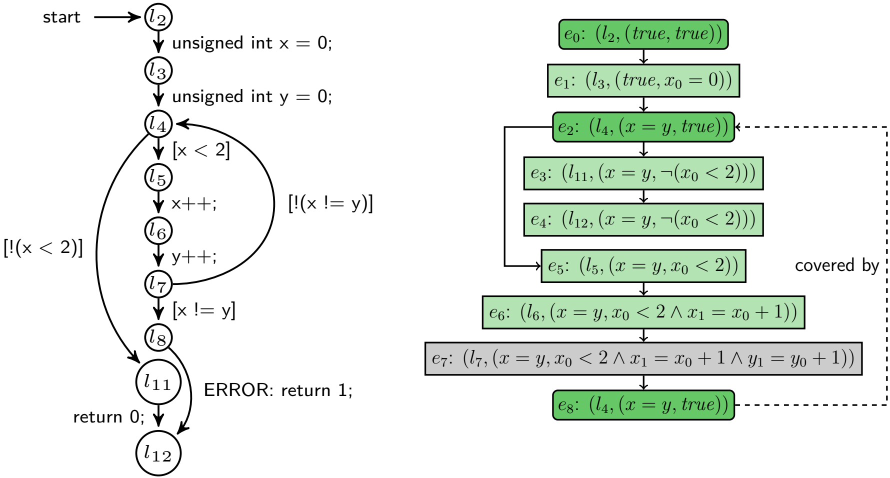
Predicate Abstraction: Example

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



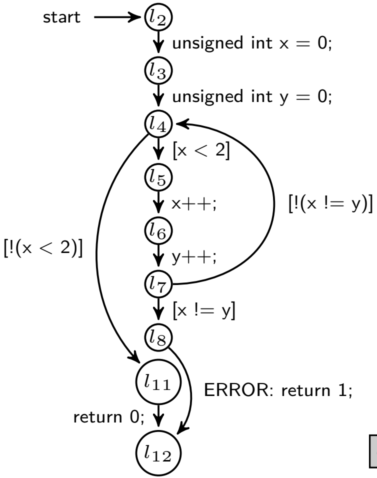
Predicate Abstraction: Example

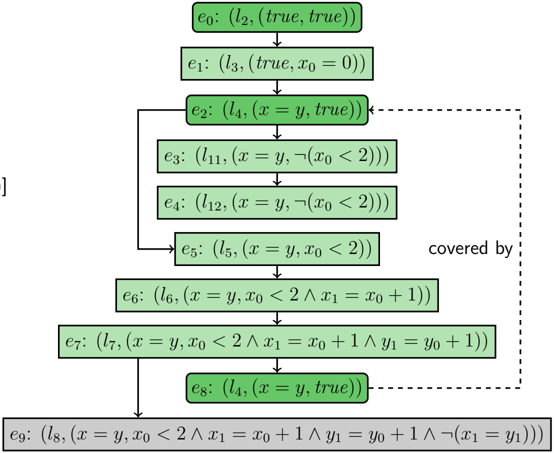
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$



Predicate Abstraction: Example

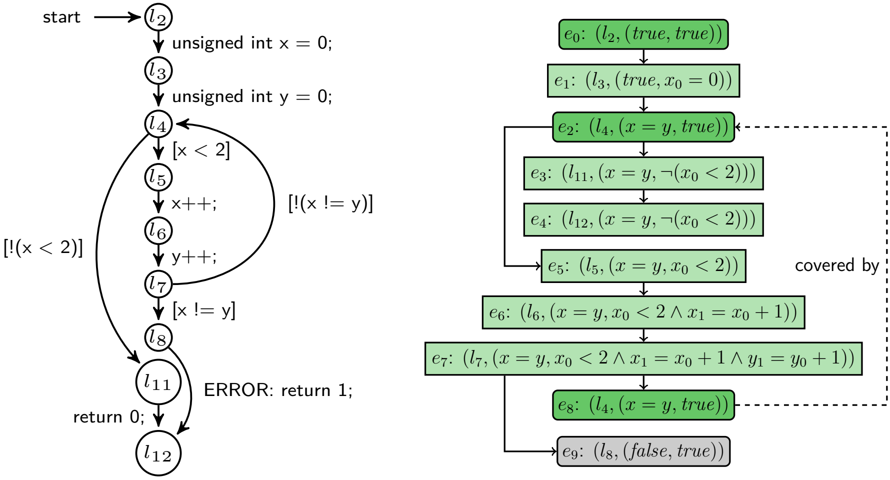
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$





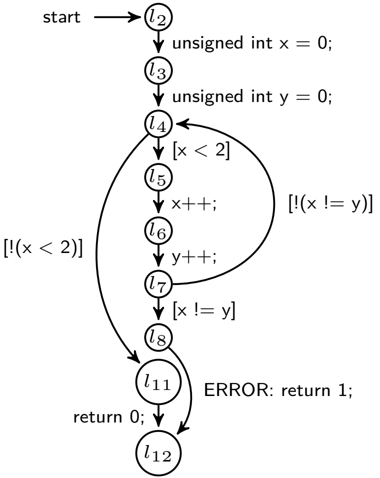
Predicate Abstraction: Example

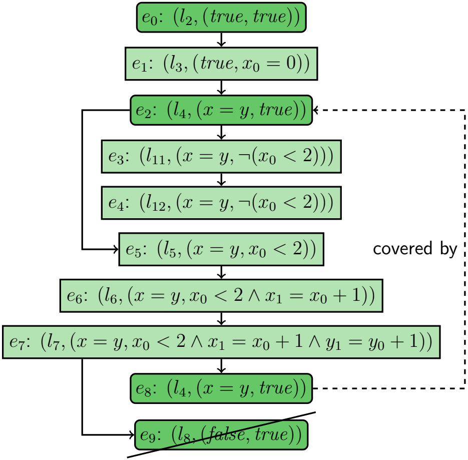
with blk^l, $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$



Predicate Abstraction: Example

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





IMPACT

- IMPACT
  - "Lazy Abstraction with Interpolants" [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

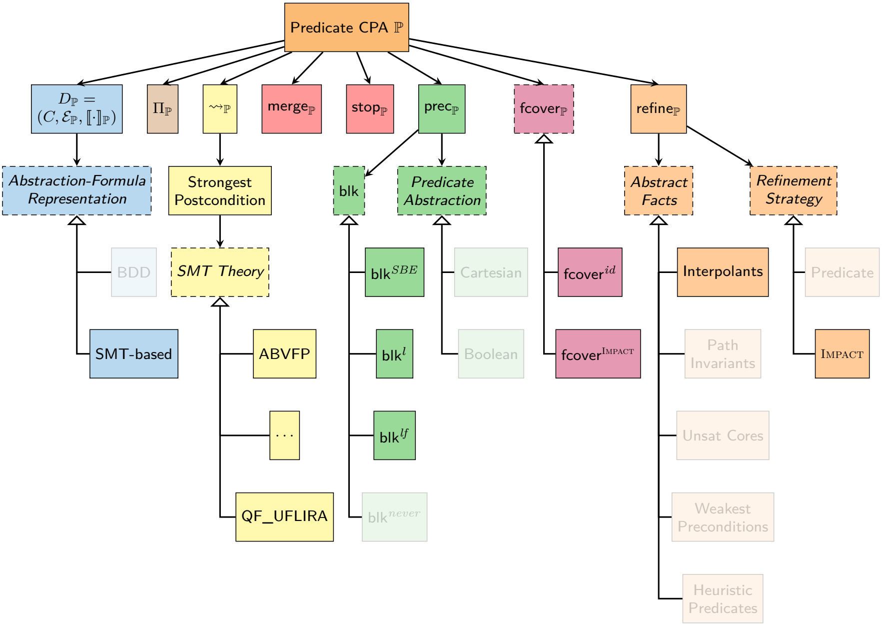
Expressing IMPACT

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

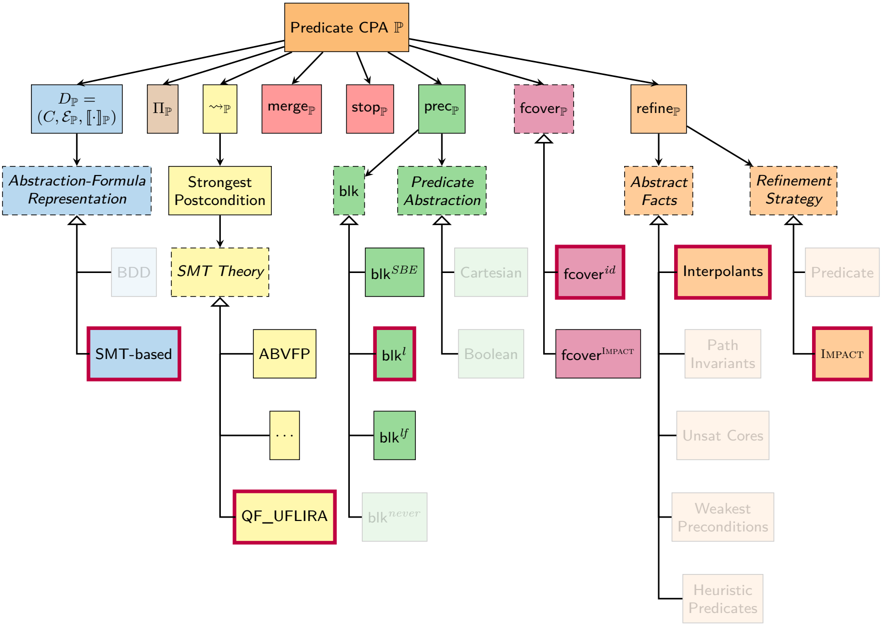
Furthermore:

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

Predicate CPA

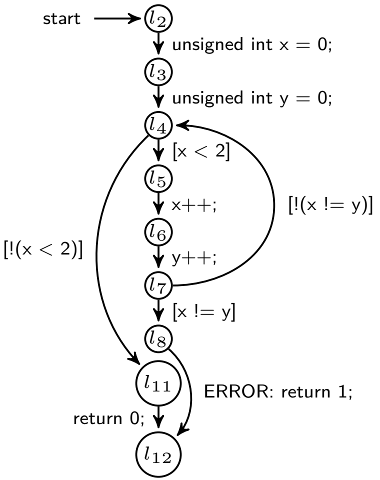


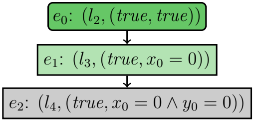
Predicate CPA



IMPACT: Example

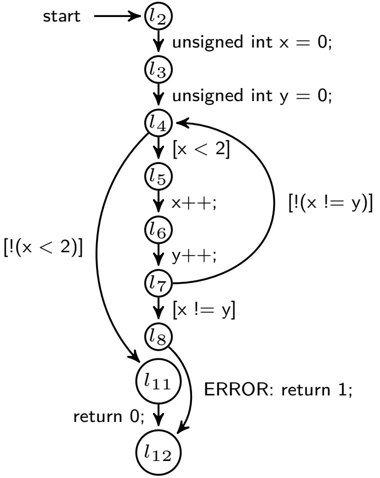
with blk^l

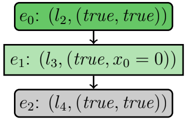




IMPACT: Example

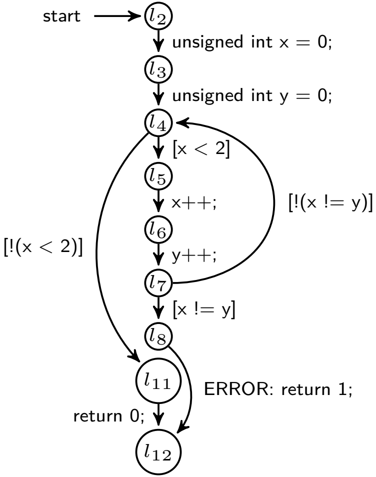
with blk^l

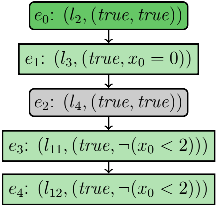




IMPACT: Example

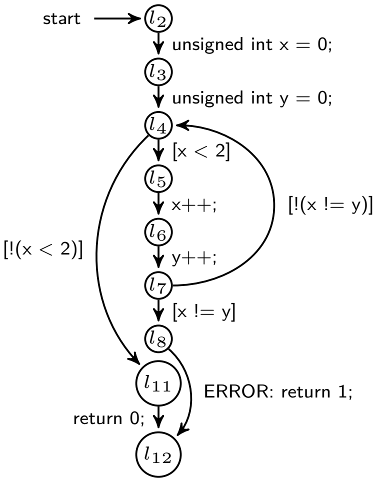
with blk^l

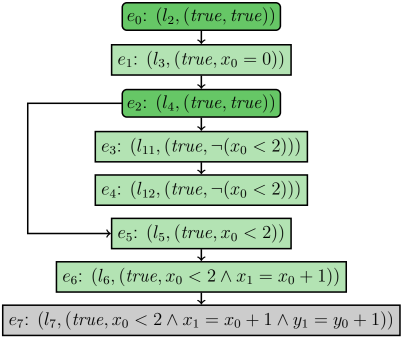




IMPACT: Example

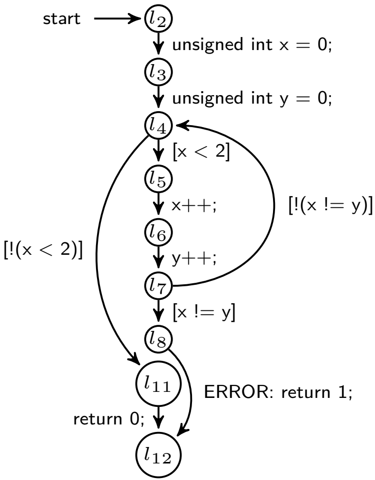
with blk^l

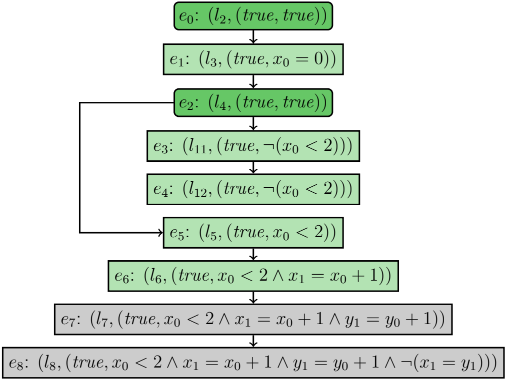




IMPACT: Example

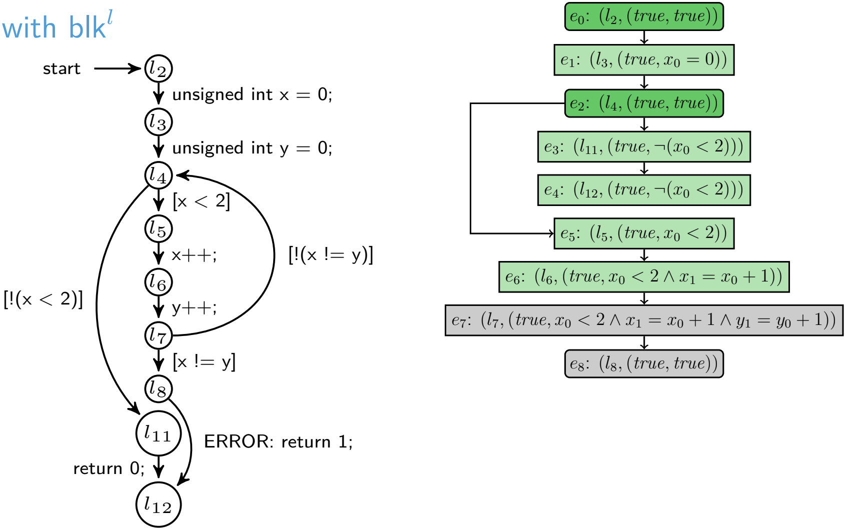
with blk^l





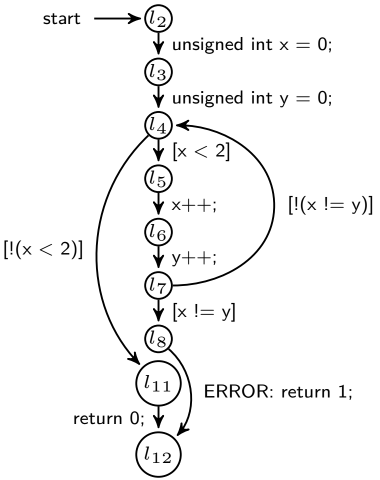
Impact: Example

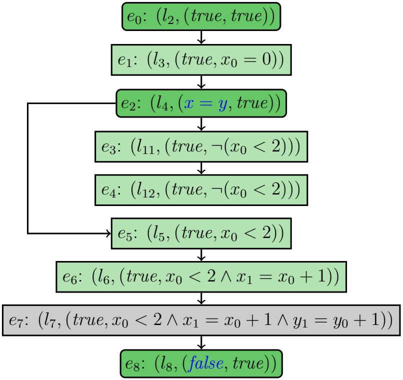
with blk^l



IMPACT: Example

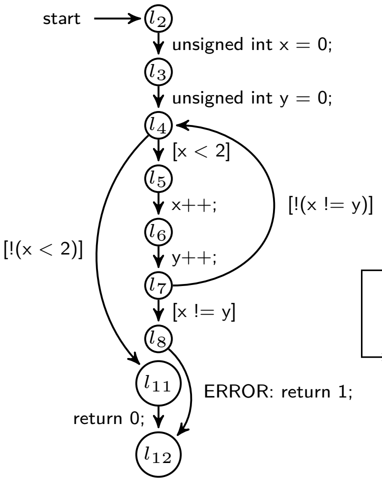
with blk^l

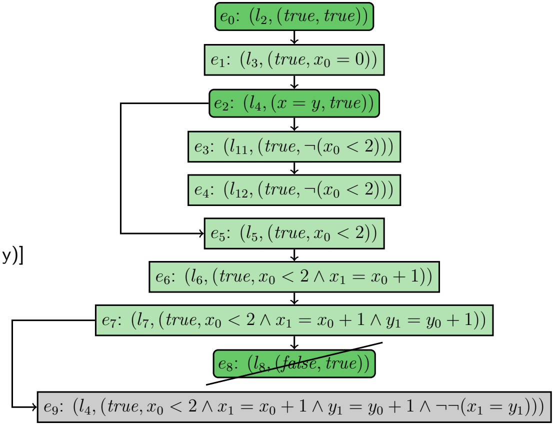




Impact: Example

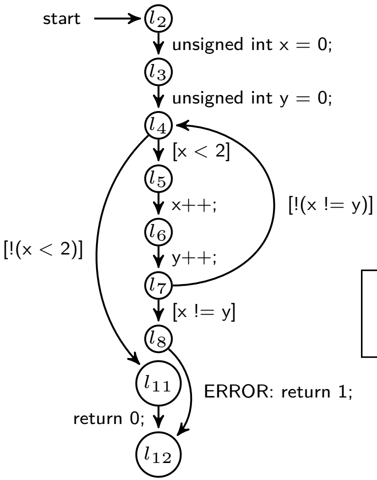
with blk^l

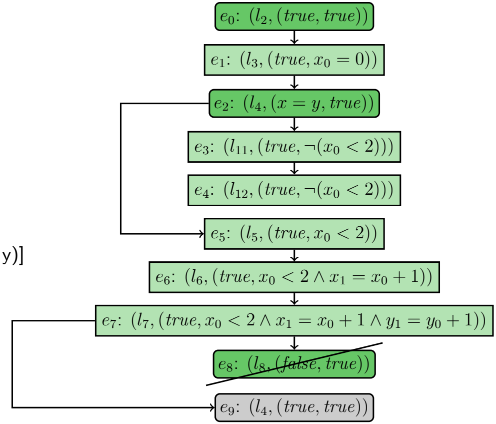




IMPACT: Example

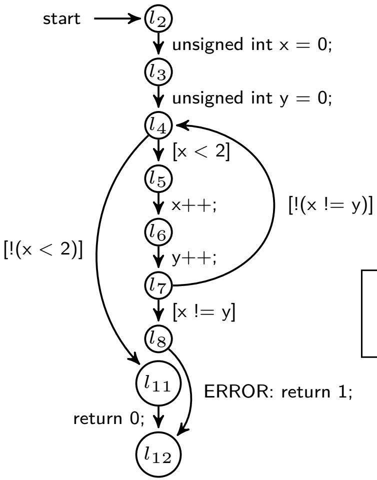
with blk^l

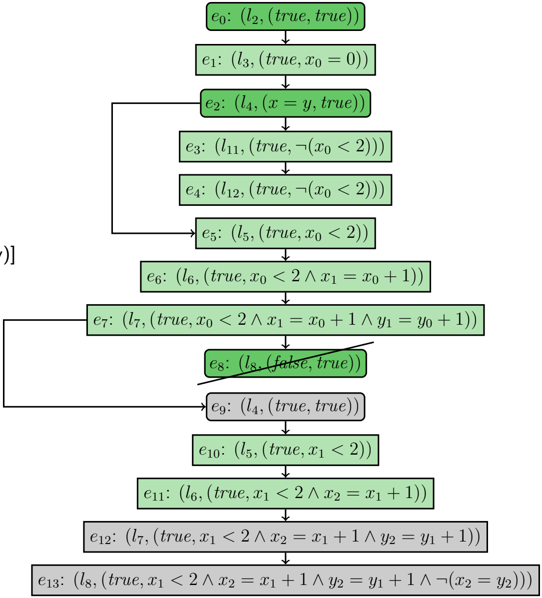




IMPACT: Example

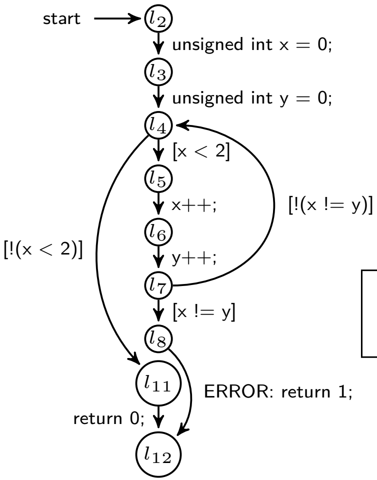
with blk^l

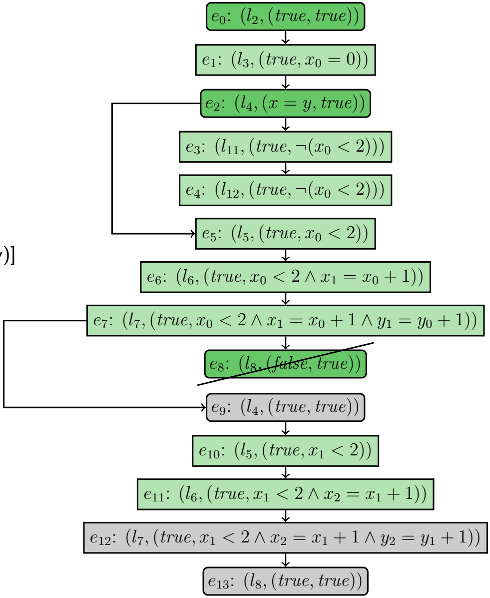




Impact: Example

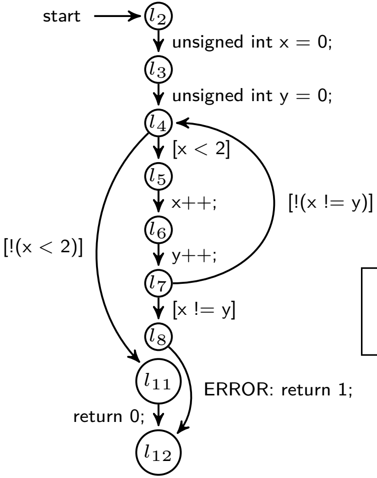
with blk^l

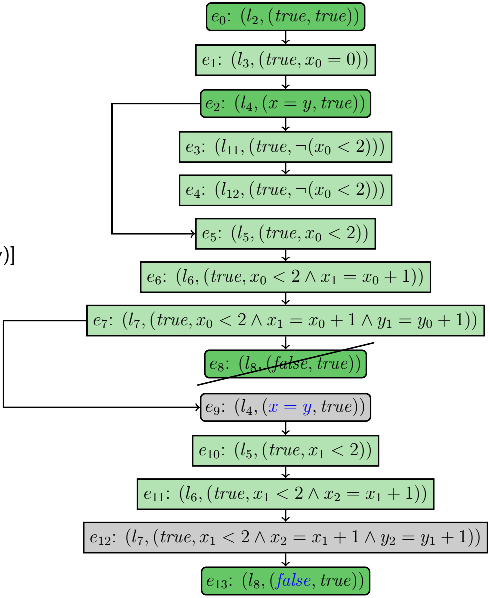




IMPACT: Example

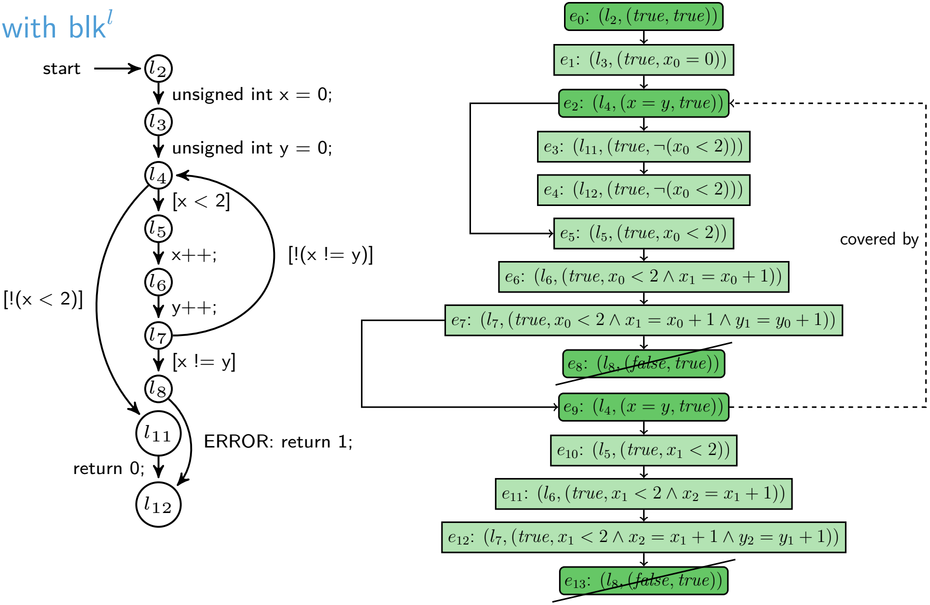
with blk^l





IMPACT: Example

with blk^l



Bounded Model Checking

- Bounded Model Checking:
  - Biere, Cimatti, Clarke, Zhu: [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

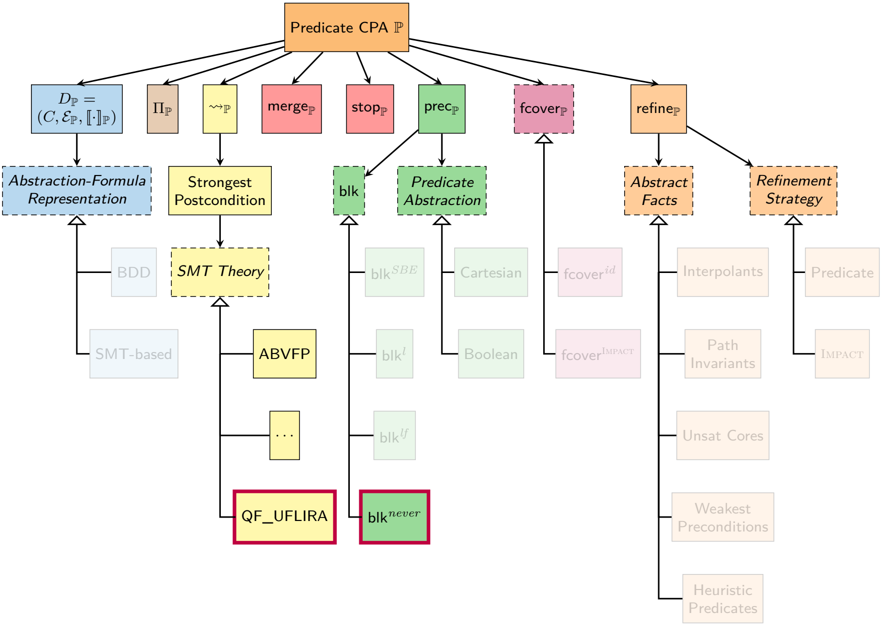
- Block Size (blk): blk^{never}

Furthermore:

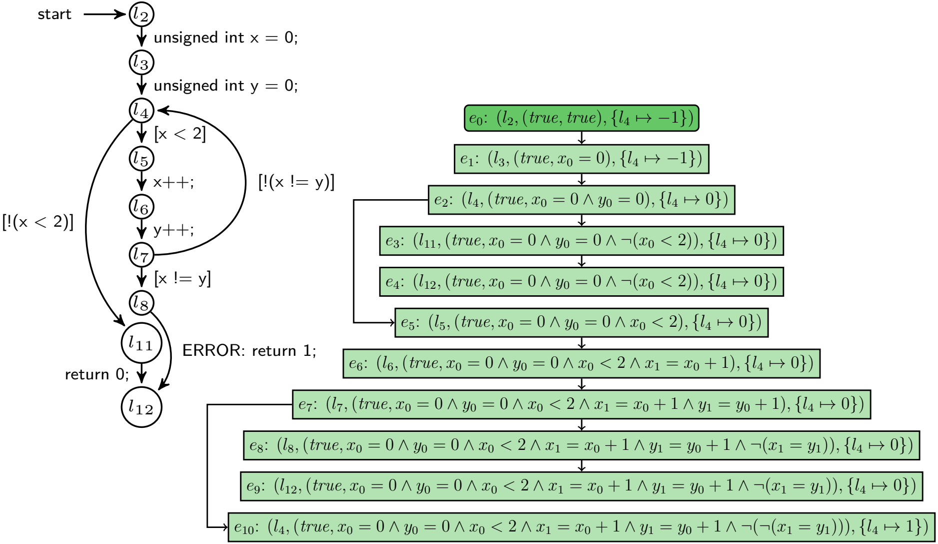
- Add CPA for bounding state space (e.g., loop bounds)
- Choices for abstraction formulas and refinement irrelevant because block end never encountered
- Use Algorithm for iterative BMC:

1: $k = 1$
2: **while** !finished **do**
3: run CPA Algorithm
4: check feasibility of each abstract error state
5: $k{+}{+}$

Predicate CPA



Bounded Model Checking: Example with $k = 1$



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: [PDMC'11]

k-Induction with Auxiliary Invariants

Induction:

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

Invariant generation:

```
prec = <weak>
invariants = ∅
while !finished do
    invariants = GenInv(prec)
    prec = RefinePrec(prec)
```

k-Induction: Example

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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