

Predicate Analysis in CPAchecker

Matthias Dangl and Philipp Wendler

LMU Munich, Germany

2017-09-05



Based on:

Dirk Beyer, Matthias Dangl, Philipp Wendler:

A Unifying View on SMT-Based Software Verification

to appear in JAR,

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

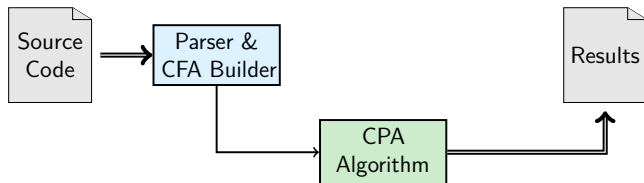
Base: Configurable Program Analysis

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, ...

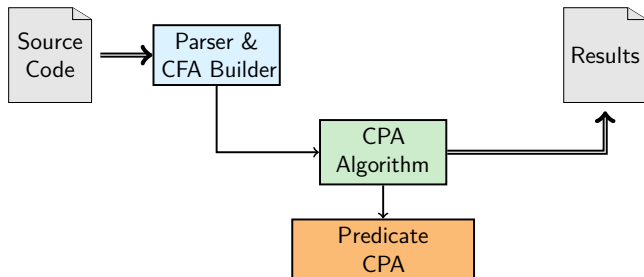
Using the CPA Framework

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



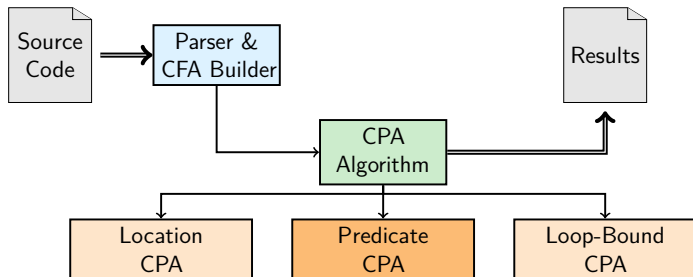
Using the CPA Framework

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



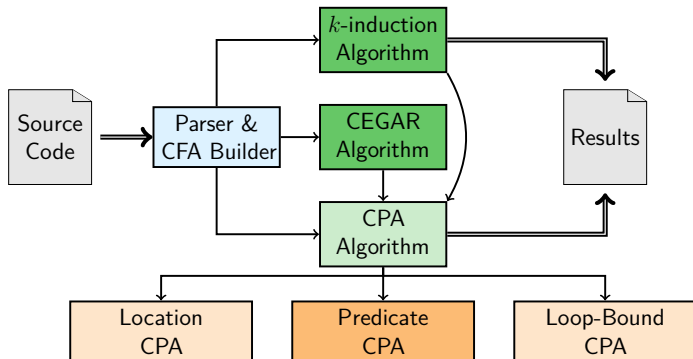
Using the CPA Framework

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

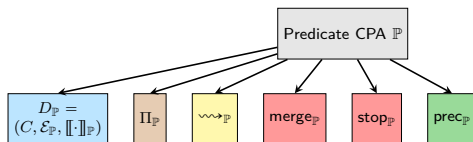


Using the CPA Framework

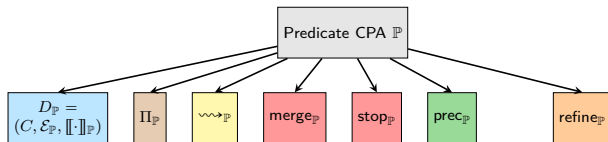
- ▶ CPA Algorithm is a configurable reachability analysis for arbitrary abstract domains
- ▶ Provide Predicate CPA for our predicate-based abstract domain
- ▶ Reuse other CPAs
- ▶ Built further algorithms on top that make use of reachability analysis



Predicate CPA



Predicate CPA



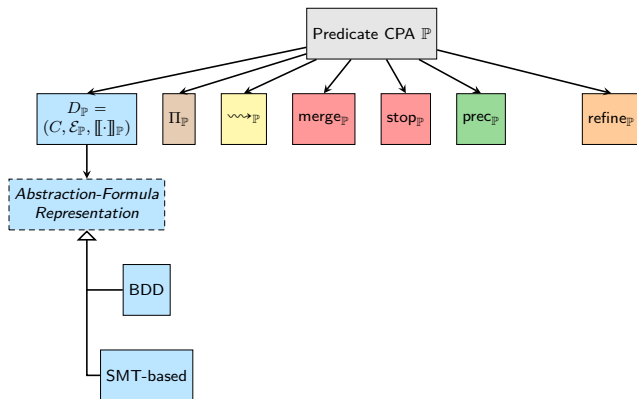
Predicate CPA: Abstract Domain

- ▶ 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

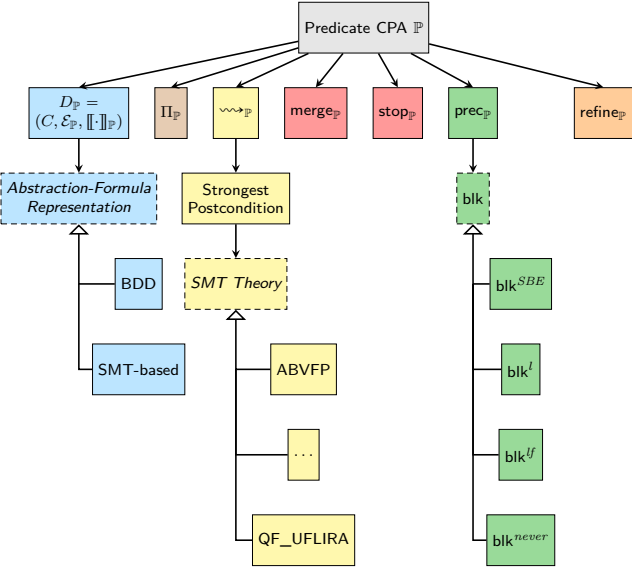
Predicate CPA: CPA Operators

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

Predicate CPA

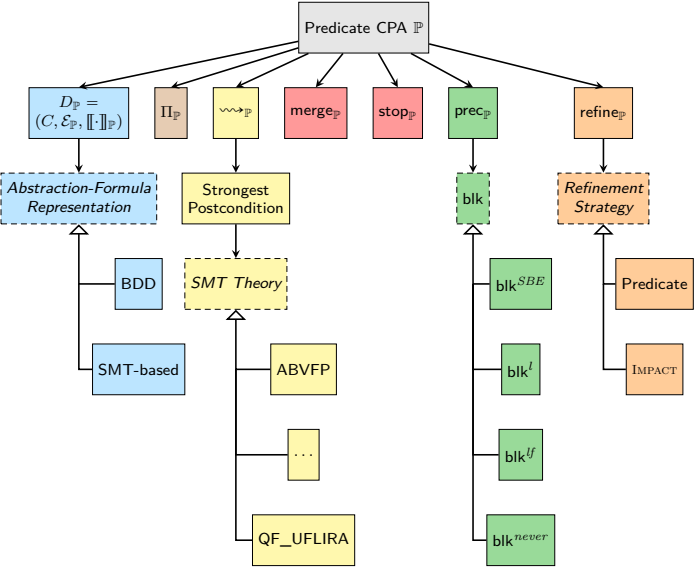


Predicate CPA: Refinement

Four steps:

1. Reconstruct ARG path to abstract error state
2. Check feasibility of path
3. Compute interpolants along path
4. Refine abstract model
 - add predicates to precision, cut ARGor
 - 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

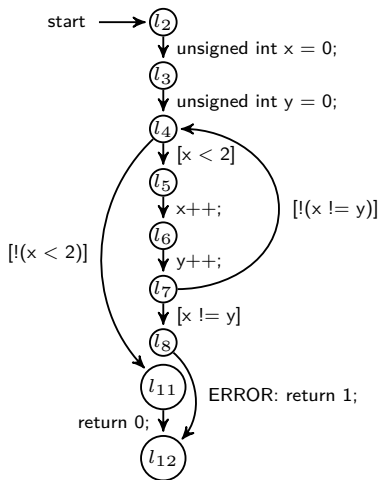
- ▶ Abstraction Formulas: BDDs
- ▶ Block Size (blk): e.g. blk^{SBE} or blk^l or blk^{lf}
- ▶ Refinement Strategy: add predicates to precision, cut ARG

Use CEGAR Algorithm:

- 1: **while** *true* **do**
- 2: run CPA Algorithm
- 3: **if** target state found **then**
- 4: call refine
- 5: **if** target state reachable **then**
- 6: **return** *false*
- 7: **else**
- 8: **return** *true*

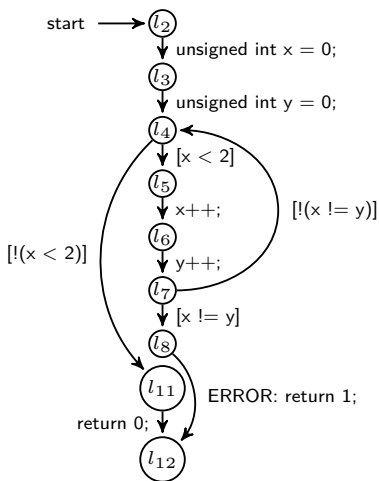
Example Program

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



Predicate Abstraction: Example

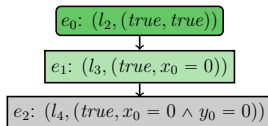
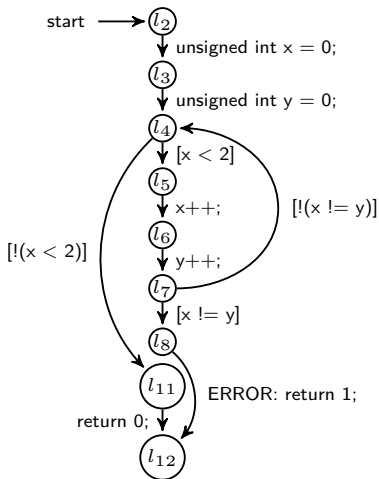
with blk^l , $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{\text{false}\}$



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

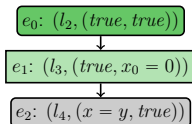
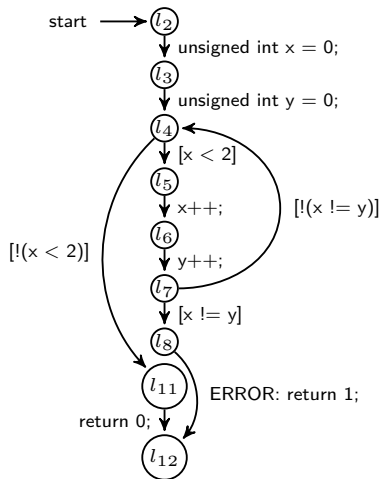
Predicate Abstraction: Example

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



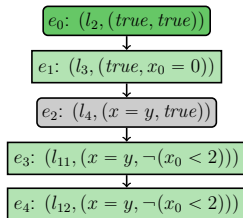
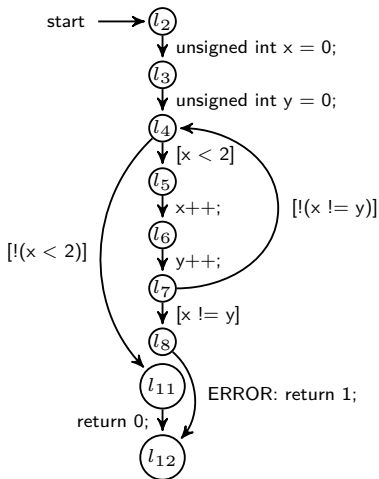
Predicate Abstraction: Example

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



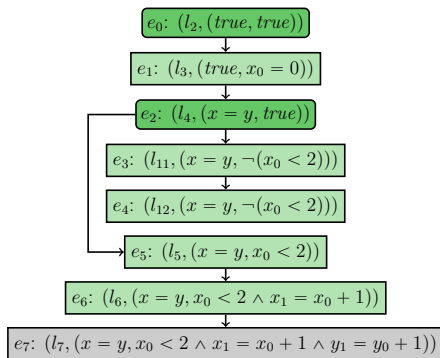
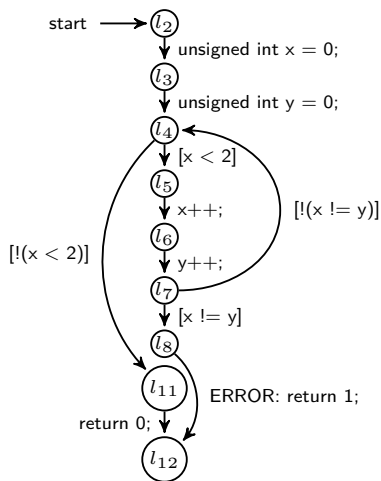
Predicate Abstraction: Example

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



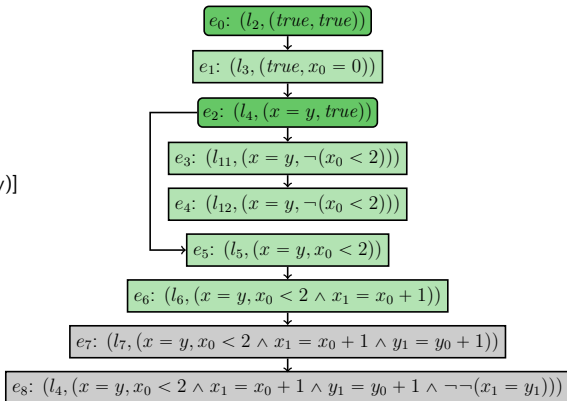
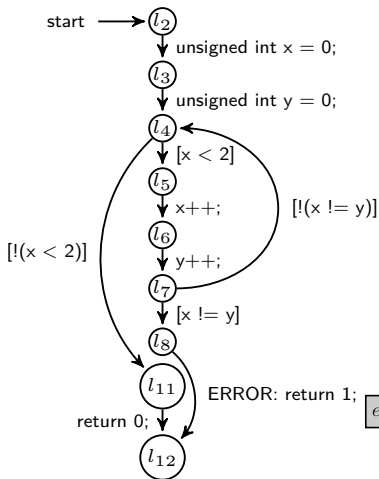
Predicate Abstraction: Example

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



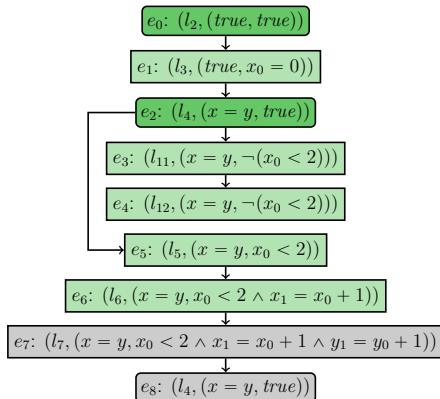
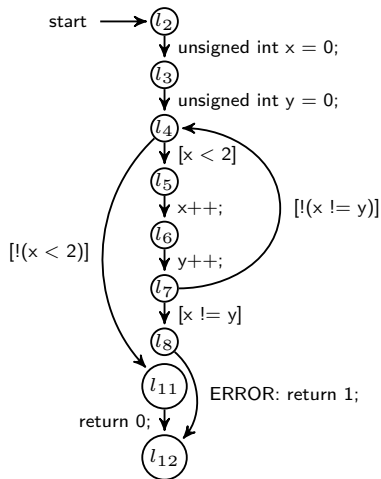
Predicate Abstraction: Example

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



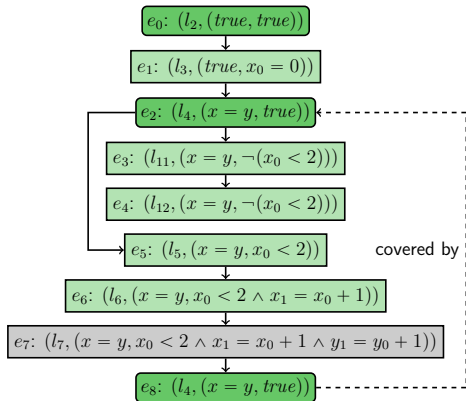
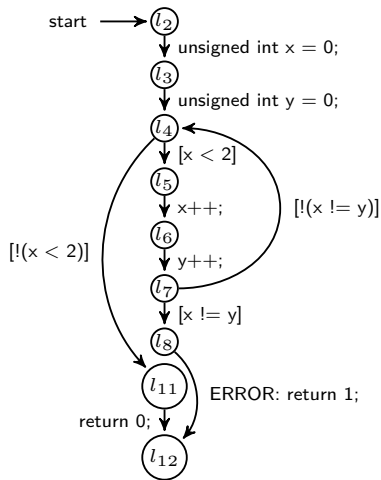
Predicate Abstraction: Example

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



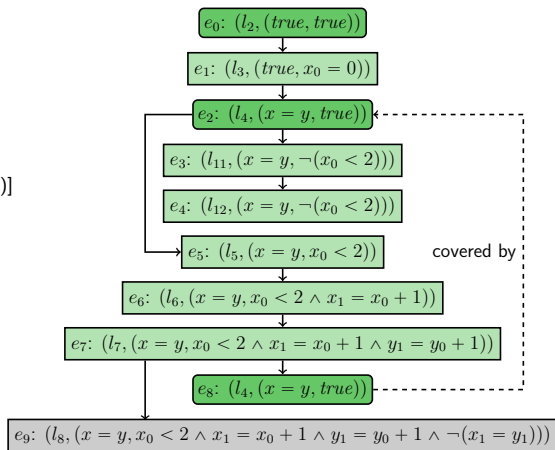
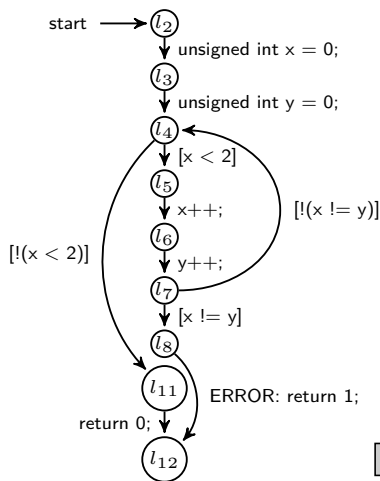
Predicate Abstraction: Example

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



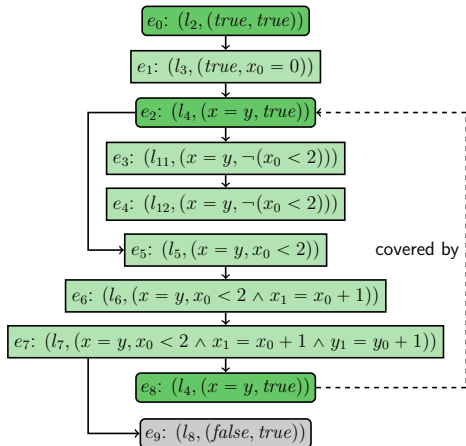
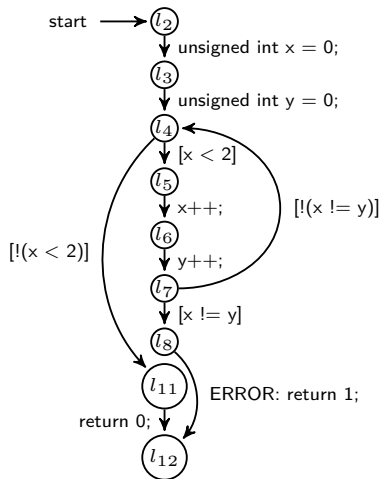
Predicate Abstraction: Example

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



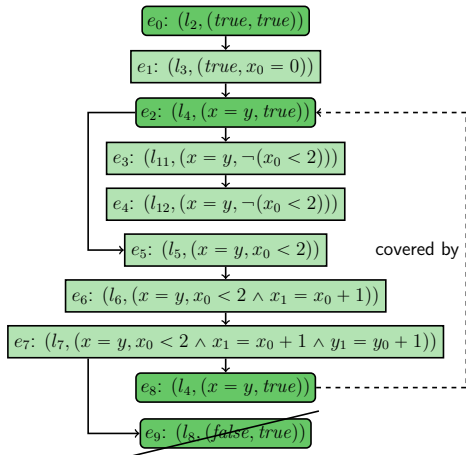
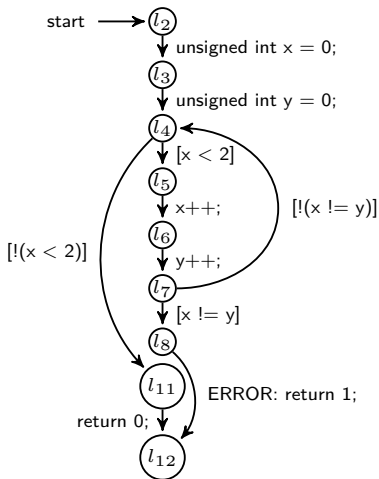
Predicate Abstraction: Example

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



Predicate Abstraction: Example

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



- ▶ 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

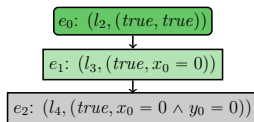
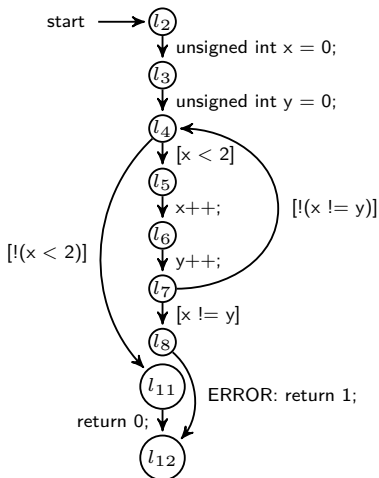
- ▶ Abstraction Formulas: SMT-based
- ▶ Block Size (blk): blk^{SBE} or other (new!)
- ▶ Refinement Strategy:
conjoin interpolants to abstract states,
recheck coverage relation

Furthermore:

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

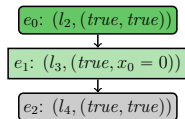
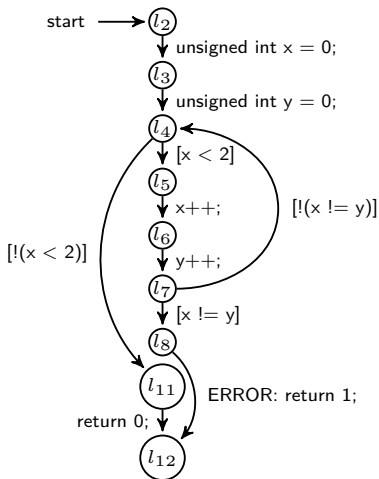
IMPACT: Example

with blk^l



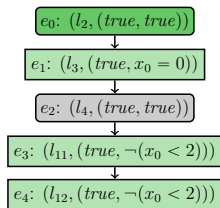
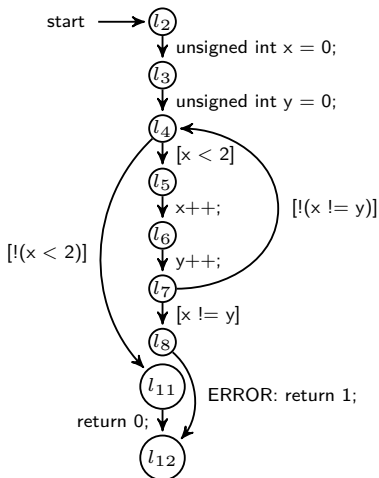
IMPACT: Example

with blk^l



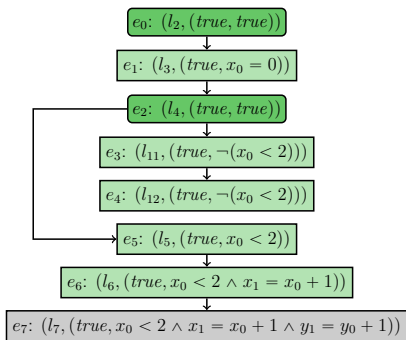
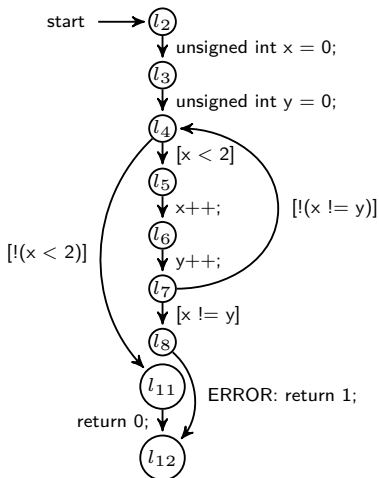
IMPACT: Example

with blk^l



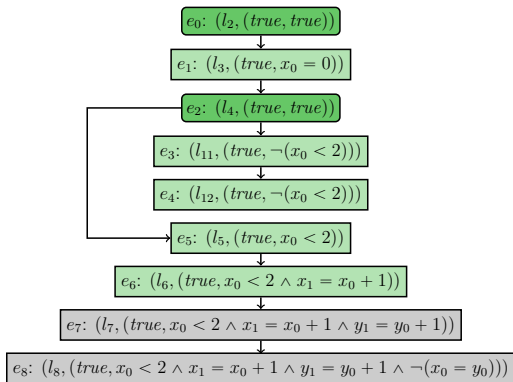
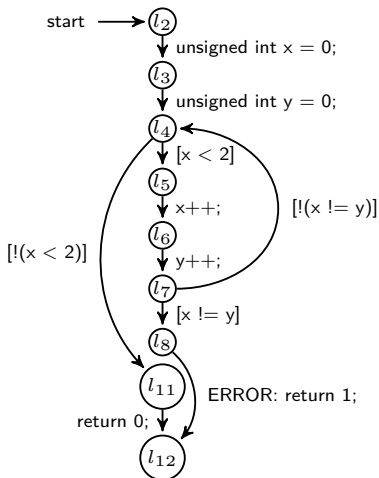
IMPACT: Example

with blk^l



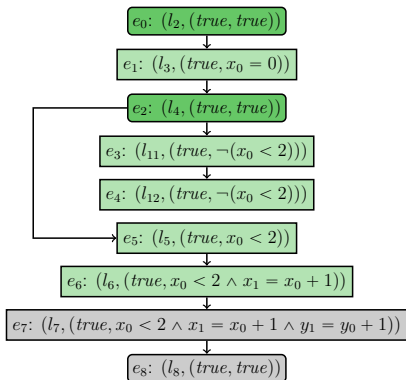
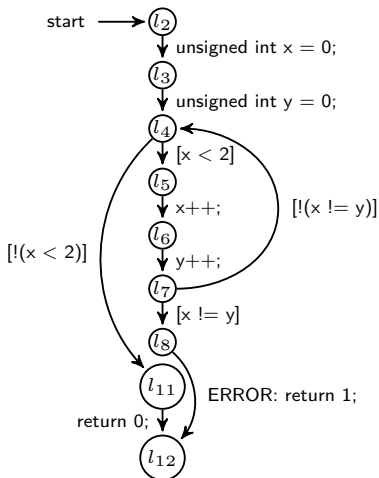
IMPACT: Example

with blk^l



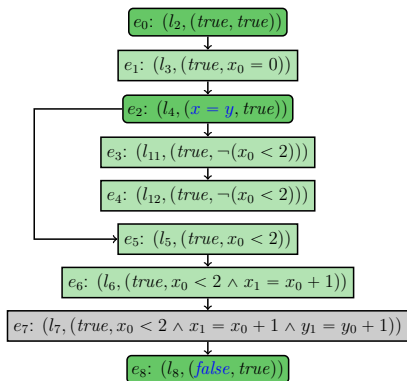
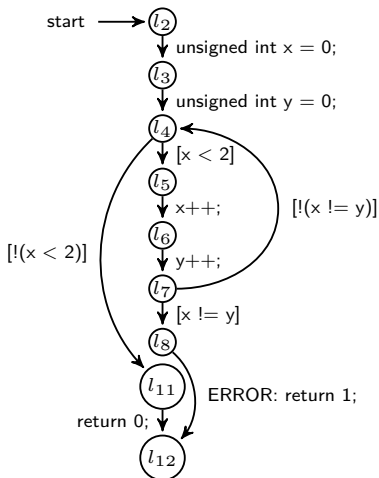
IMPACT: Example

with blk^l



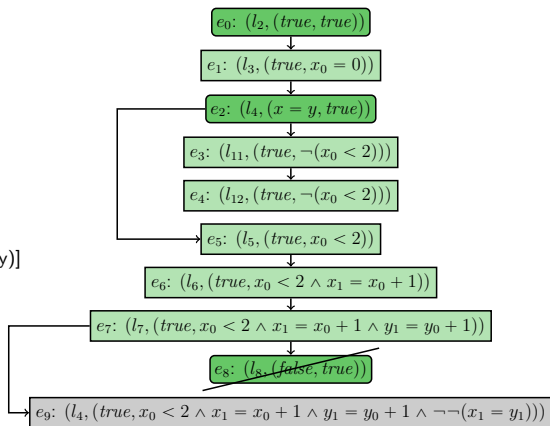
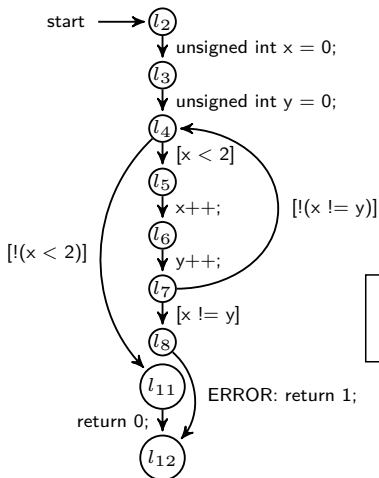
IMPACT: Example

with blk^l



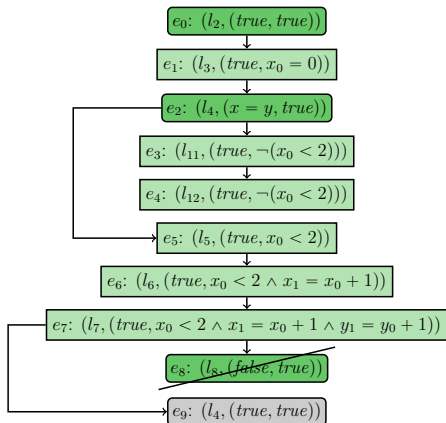
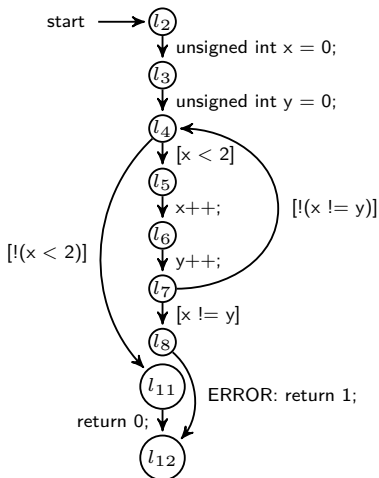
IMPACT: Example

with blk^l



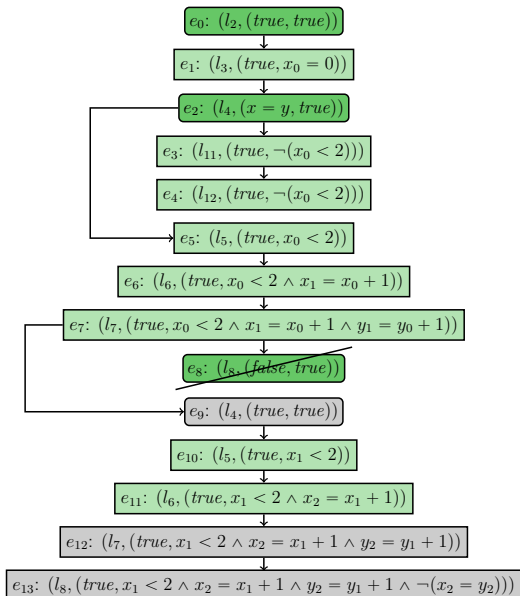
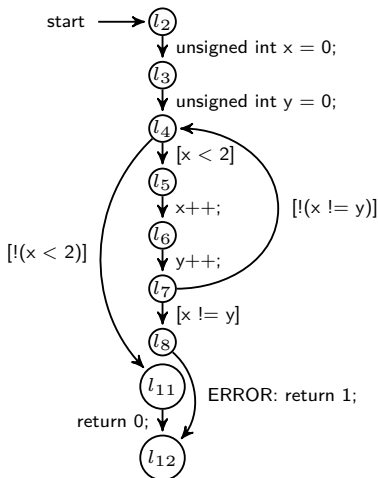
IMPACT: Example

with blk^l



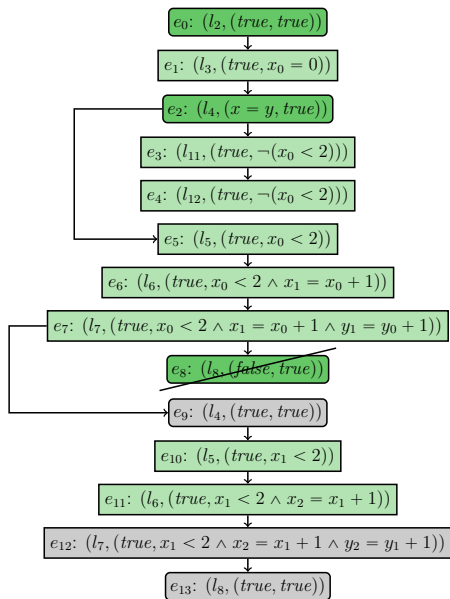
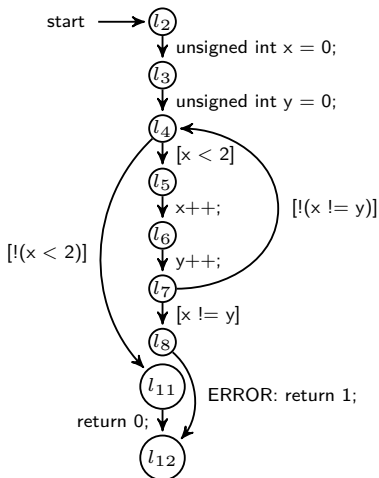
IMPACT: Example

with blk^l



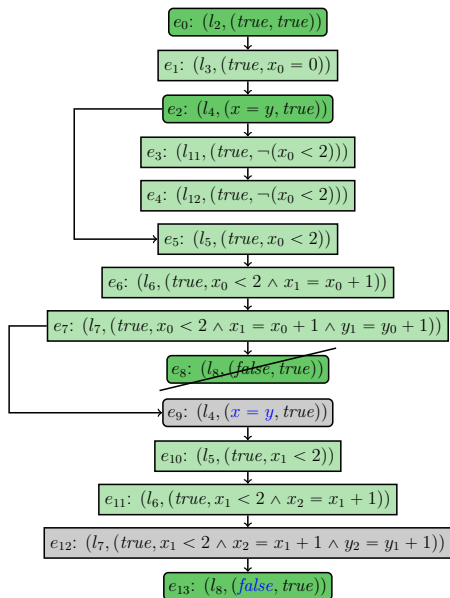
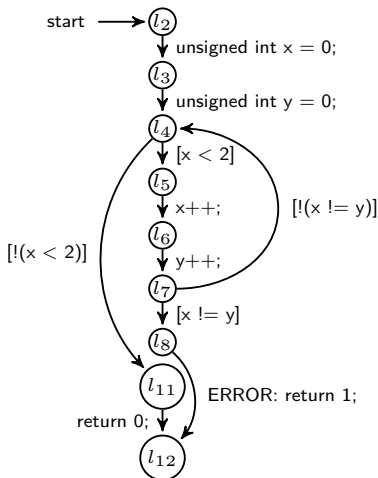
IMPACT: Example

with blk^l



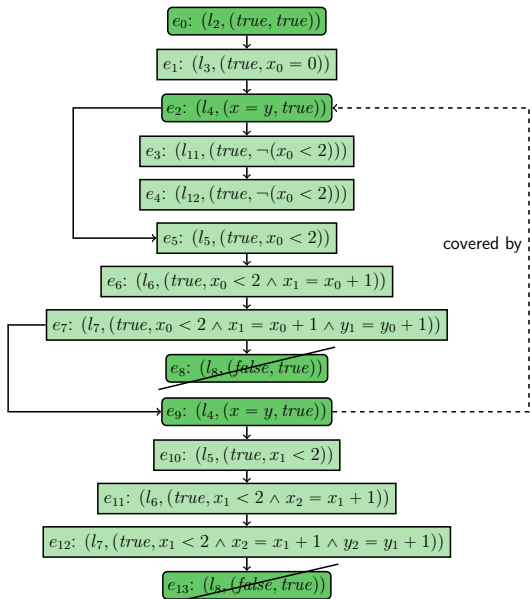
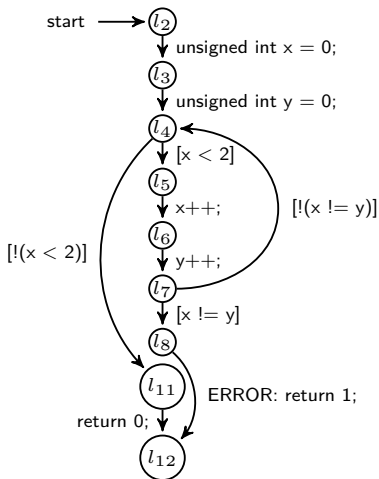
IMPACT: Example

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:

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

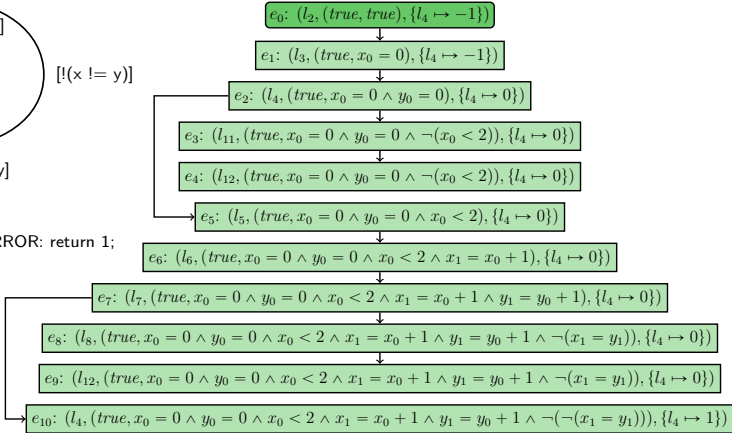
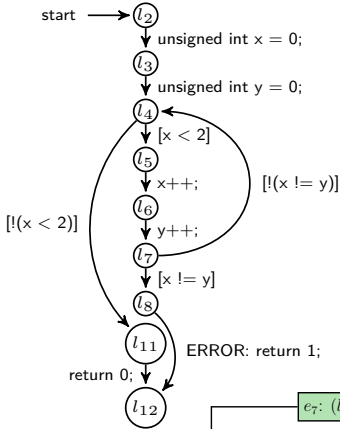
Expressing BMC

- ▶ 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: $k++$

Bounded Model Checking: Example with $k = 1$



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 $Z3$ as expected:
BV $<$ sound LIRA $<$ unsound LIRA

Point of View: Theories and Encodings

- ▶ Unsound linear encoding always the easiest (as expected)
- ▶ Correctness as expected:
 $BV > \text{sound LIRA} > \text{unsound LIRA}$
- ▶ Effectivity for $Z3$ as expected:
 $BV < \text{sound LIRA} < \text{unsound LIRA}$
- ▶ Effectivity for $MATHSAT5$:
 $\text{sound LIRA} < BV \approx \text{unsound LIRA}$
(but BV needs more CPU time)

⇒ $MATHSAT5$ is really good with bitvectors.

Point of View: Theories and Encodings

- ▶ Unsound linear encoding always the easiest (as expected)
 - ▶ Correctness as expected:
 $BV > \text{sound LIRA} > \text{unsound LIRA}$
 - ▶ Effectivity for Z3 as expected:
 $BV < \text{sound LIRA} < \text{unsound LIRA}$
 - ▶ Effectivity for MATHSAT5:
 $\text{sound LIRA} < BV \approx \text{unsound LIRA}$
(but BV needs more CPU time)
 - ▶ Effectivity for SMTINTERPOL:
 $\text{sound LIRA} \ll \text{unsound LIRA}$
- ⇒ MATHSAT5 is really good with bitvectors.

Point of View: Theories and Encodings

- ▶ Unsound linear encoding always the easiest (as expected)
 - ▶ Correctness as expected:
 $BV > \text{sound LIRA} > \text{unsound LIRA}$
 - ▶ Effectivity for Z3 as expected:
 $BV < \text{sound LIRA} < \text{unsound LIRA}$
 - ▶ Effectivity for MATHSAT5:
 $\text{sound LIRA} < BV \approx \text{unsound LIRA}$
(but BV needs more CPU time)
 - ▶ Effectivity for SMTINTERPOL:
 $\text{sound LIRA} \ll \text{unsound LIRA}$
- ⇒ MATHSAT5 is really good with bitvectors.
- ⇒ 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