Effective Automated Software Verification: A Multilayered Approach

Martin Blicha

Advisors: Natasha Sharygina and Jan Kofroň

Università della Svizzera italiana, Lugano, Switzerland

Charles University, Prague, Czech Republic

March 21, 2023

Software Verification

Introduction

- (Dis)proving correctness with respect to formal specification

```
x = 0;
y = 0;
while(*) {
  x = x + y;
  y = y + 1;
}
assert(x >= 0);
```

Software Verification

Introduction

- (Dis)proving correctness with respect to formal specification
- Formal methods
- Model checking
  - Automated, systematic, exhaustive
  - Symbolic representation using logical formulas

```
x = 0;
y = 0;
while(*) {
  x = x + y;
  y = y + 1;
}
assert(x >= 0);
```

- (Dis)proving correctness with respect to formal specification
- Formal methods
- Model checking
  - Automated, systematic, exhaustive
  - Symbolic representation using logical formulas
- Undecidable problem in general
- Many applications in industry

```
x = 0;
y = 0;
while(*) {
  x = x + y;
  y = y + 1;
}
assert(x >= 0);
```

Software Verification

Challenges

- Complexity and scalability

Software Verification

Challenges

state-space explosion

- Complexity and scalability
- Symbolic representation
- Abstraction
  - What is the right abstraction?
  - How to compute it automatically and efficiently?

Software Verification

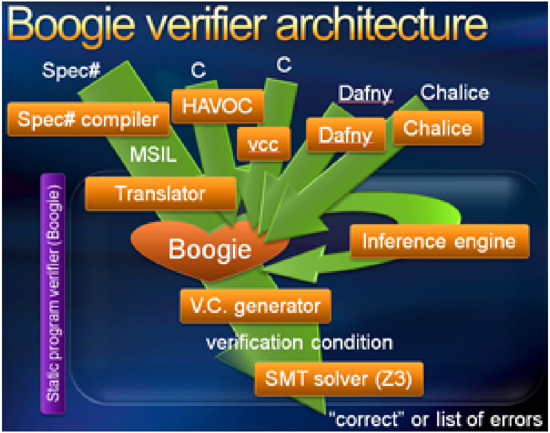
Challenges

- Complexity and scalability
- Symbolic representation
- Abstraction
  - What is the right abstraction?
  - How to compute it automatically and efficiently?

state-space explosion

- Our contributions
  - More precise abstractions with Craig interpolation
  - Novel application of abstraction for detecting long counterexamples
  - Efficient parallelization with information exchange

Verification in Industry



https://www.microsoft.com/en-us/research/project/boogie-an-intermediate-verification-language/

Verification in Industry

CLOUD AND SYSTEMS

How AWS's Automated Reasoning Group helps make AWS and other Amazon products more secure

Amazon scientists are on the cutting edge of using math-based logic to provide better network security, access management, and greater reliability.

By Douglas Gantenbein

June 24, 2020

https://www.amazon.science/latest-news/

how-awss-automated-reasoning-group-helps-make-aws-and-other-amazon-products-more-secure

Verification in Industry

Docs » SMTChecker and Formal Verification

Edit on GitHub

SMTChecker and Formal Verification

Using formal verification it is possible to perform an automated mathematical proof that your source code fulfills a certain formal specification. The specification is still formal (just as the source code), but usually much simpler.

Note that formal verification itself can only help you understand the difference between what you did (the specification) and how you did it (the actual implementation). You still need to check whether the specification is what you wanted and that you did not miss any unintended effects of it.

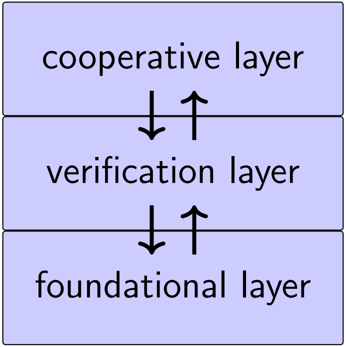
Solidity implements a formal verification approach based on SMT and Horn solving. The SMTChecker module automatically tries to prove that the code satisfies the specification given by `require` and `assert` statements. That is, it considers `require` statements as assumptions and tries to prove that the conditions inside `assert` statements are always true. If an assertion failure is found, a counterexample may be given to the user showing how the assertion can be violated. If no warning is given by the SMTChecker for a property, it means that the property is safe.

https://docs.soliditylang.org/en/v0.8.3/smtchecker.html

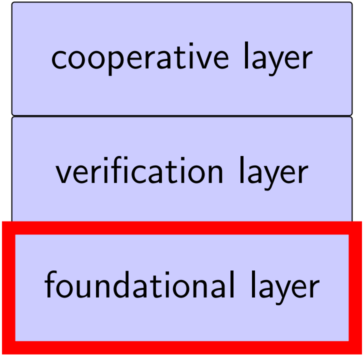
Our Layered View of Software Verification

software verification

Our Layered View of Software Verification



Our Layered View of Software Verification



Foundations

- Satisfiability Modulo Theories (SMT)
  - Theory of linear arithmetic (reals, integers)

$$x \geq 0 \wedge (y = x + 1 \vee y = x) \wedge y < 0$$

Foundations

- Satisfiability Modulo Theories (SMT)
  - Theory of linear arithmetic (reals, integers)

$$x \geq 0 \wedge (y = x + 1 \vee y = x) \wedge y < 0$$

- SMT solvers
  - OpenSMT, Z3, cvc5, Yices, MathSAT, SMTInterpol, Princess, ...

Foundations

- Satisfiability Modulo Theories (SMT)
  - Theory of linear arithmetic (reals, integers)

$$x \geq 0 \land (y = x + 1 \lor y = x) \land y < 0$$

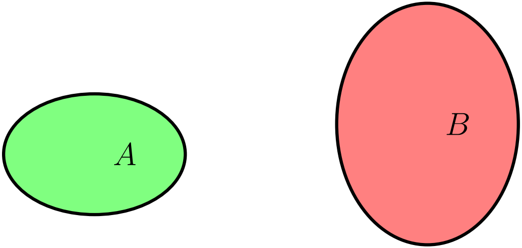
- SMT solvers
  - OpenSMT, Z3, cvc5, Yices, MathSAT, SMTInterpol, Princess, ...
- Craig interpolation
  - Source of abstraction
  - Efficient computation from proofs of unsatisfiability by SMT solvers

Craig interpolation

Definition (Craig '57)

Let A, B be formulas such that $A \wedge B \rightarrow \bot$. Formula I is an interpolant for A, B if

- $A \rightarrow I$,
- $I \wedge B \rightarrow \bot$, and
- I uses only shared free variables of A and B.

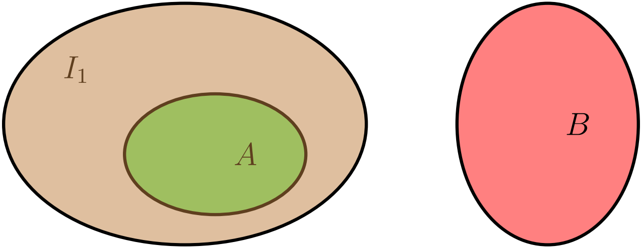


Craig interpolation

Definition (Craig '57)

Let A, B be formulas such that $A \wedge B \rightarrow \bot$. Formula I is an interpolant for A, B if

- $A \rightarrow I$,
- $I \wedge B \rightarrow \bot$, and
- I uses only shared free variables of A and B.

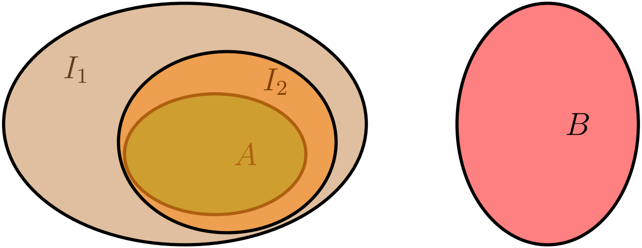


Craig interpolation

Definition (Craig '57)

Let A, B be formulas such that $A \wedge B \rightarrow \bot$. Formula I is an interpolant for A, B if

- $A \rightarrow I$,
- $I \wedge B \rightarrow \bot$, and
- I uses only shared free variables of A and B.



Our Layered View of Software Verification

cooperative layer

verification layer

foundational layer

Checking feasibility of a program path

```
int max(int i, int j) {
  if (i > j)
    return i;
  else
    return j;
}

int main() {
  int r = max(random(),0);
  assert(r > 0);
}
```

Checking feasibility of a program path

```
int max(int i, int j) {
  if (i > j)
    return i;
  else
    return j;
}

int main() {
  int r = max(random(),0);
  assert(r > 0);
}
```

$$j = 0 \land i > j$$
$$\land\, r = i \land \neg(r > 0)$$

Checking feasibility of a program path

```
int max(int i, int j) {
  if (i > j)
    return i;
  else
    return j;
}

int main() {
  int r = max(random(),0);
  assert(r > 0);
}
```

$$j = 0 \wedge i > j$$
$$\wedge\, r = i \wedge \neg(r > 0)$$

UNSAT

Checking feasibility of a program path

```
int max(int i, int j) {
  if (i > j)
    return i;
  else
    return j;
}

int main() {
  int r = max(random(),0);
  assert(r > 0);
}
```

$$j = 0 \wedge i > j$$
$$\wedge\, r = i \wedge \neg(r > 0)$$

UNSAT

$$j = 0 \wedge \neg(i > j)$$
$$\wedge\, r = j \wedge \neg(r > 0)$$

Checking feasibility of a program path

```
int max(int i, int j) {
  if (i > j)
    return i;
  else
    return j;
}

int main() {
  int r = max(random(),0);
  assert(r > 0);
}
```

$$j = 0 \land i > j$$
$$\land\, r = i \land \neg(r > 0)$$

UNSAT

$$j = 0 \land \neg(i > j)$$
$$\land\, r = j \land \neg(r > 0)$$

SAT

$$i = -1, j = 0, r = 0$$

Model Checking

Transition systems

```
x = 0;
y = 0;
while(*) {
    x = x + y;
    y = y + 1;
}
assert(x >= 0);
```

$$Init \equiv x = 0 \land y = 0$$

$$Tr \equiv x' = x + y \land y' = y + 1$$

$$Bad \equiv \neg(x \geq 0)$$

Model Checking

Transition systems

```
x = 0;
y = 0;
while(*) {
    x = x + y;
    y = y + 1;
}
assert(x >= 0);
```

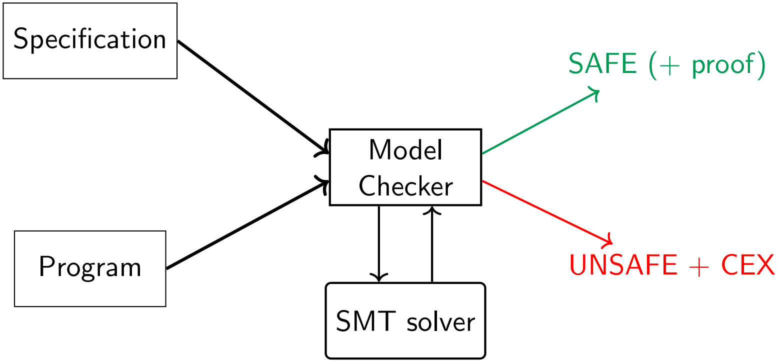
$$Init \equiv x = 0 \land y = 0$$

$$Tr \equiv x' = x + y \land y' = y + 1$$

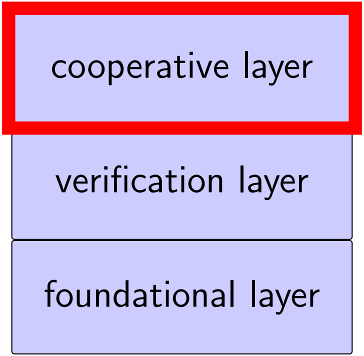
$$Bad \equiv \neg(x \geq 0)$$

Complex control flow, functions $\implies$ Constrained Horn Clauses (CHC)

SMT-based Model Checking

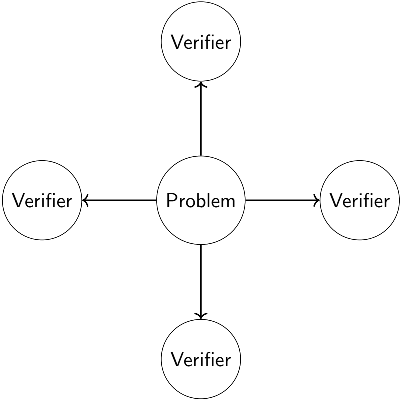


Our Layered View of Software Verification



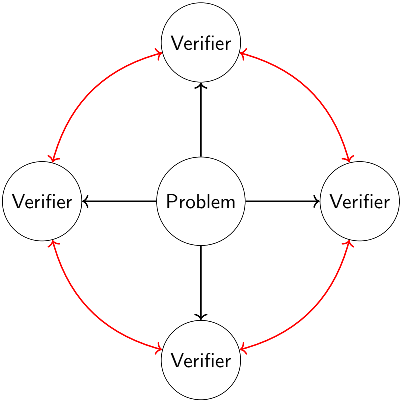
Multi-agent scenario

- Multiple verifiers solving the same problem



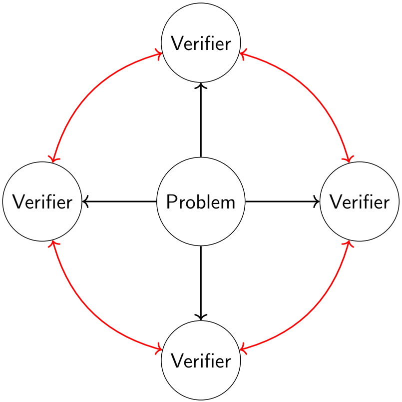
Multi-agent scenario

- Multiple verifiers solving the same problem
- Cooperation with information exchange



Multi-agent scenario

- Multiple verifiers solving the same problem
- Cooperation with information exchange
- Diverse behaviour



Our contributions

	concepts	implementation
cooperative layer	IcE/FiRE	SMTS + SALLY
verification layer	Transition Power Abstraction	GOLEM
foundational layer	Decomposed Farkas interpolants	OPENSMT

Our contributions

	concepts	implementation
cooperative layer	IcE/FiRE	SMTS + SALLY
verification layer	Transition Power Abstraction	GOLEM
foundational layer	Decomposed Farkas interpolants	OPENSMT

Decomposed Farkas interpolants

Motivation

- Divergence of interpolation-based model-checking algorithm

Decomposed Farkas interpolants

Motivation

- Divergence of interpolation-based model-checking algorithm
- Existing approaches
  - Interpolation abstractions (templates) [RS13]
  - Interpolation with Conflict Resolution [SJ18]
  - Global guidance [VKCSG20]

Decomposed Farkas interpolants

Motivation

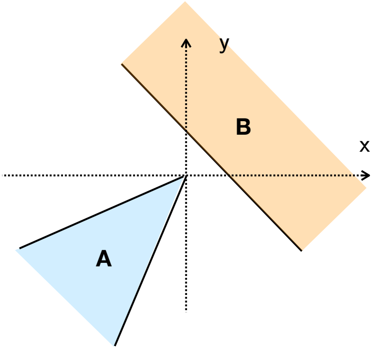
- Divergence of interpolation-based model-checking algorithm
- Existing approaches
  - Interpolation abstractions (templates) [RS13]
  - Interpolation with Conflict Resolution [SJ18]
  - Global guidance [VKCSG20]
- Ours: Generalize existing interpolation procedure based on Farkas lemma

Farkas interpolant

- Unsatisfiable system of linear inequalities, partitioned into A and B
- Farkas coefficients
  - Coefficients of the linear combination witnessing unsatisfiability
- Farkas interpolant
  - Linear combination restricted to the A-part of the linear system
  - Always a single inequality

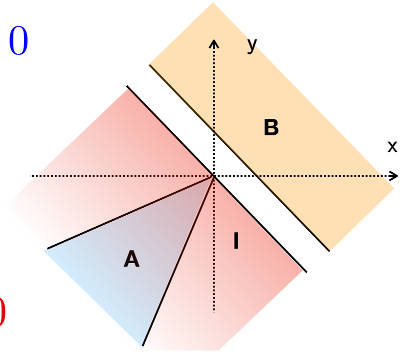
$$2x - y \leq 0$$
$$-x + 2y \leq 0$$

$$x + y \geq 1$$



$$2x - y \leq 0$$
$$-x + 2y \leq 0$$

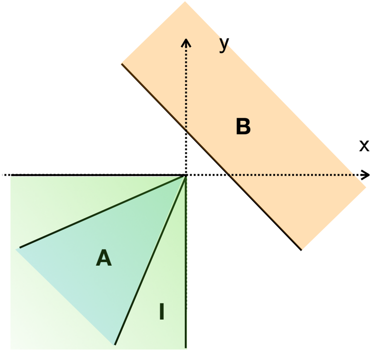
$$x + y \geq 1$$



$$x + y \leq 0$$

$$2x - y \leq 0$$
$$-x + 2y \leq 0$$

$$x + y \geq 1$$



$$x \leq 0$$
$$y \leq 0$$

Decomposed Farkas interpolants

- Decomposition of vector of Farkas coefficients into linearly independent components

input : matrix M, vector $\mathbf{v}$ such that $\mathbf{v} \in ker(M)$ and $\mathbf{v} > \mathbf{0}$

output: $\{\mathbf{w_1}, \ldots, \mathbf{w_n}\}$, a decomposition of $\mathbf{v}$, such that $\mathbf{w_i} \in ker(M), \mathbf{w_i} \geq \mathbf{0}$ and $\sum \mathbf{w_i} = \mathbf{v}$

$M \leftarrow \texttt{RREF(}M\texttt{)}$

$n \leftarrow \texttt{Nullity(}M\texttt{)}$

if $n = 1$ **then return** $\{\mathbf{v}\}$

$(\mathbf{b_1}, \ldots, \mathbf{b_n}) \leftarrow \texttt{KernelBasis(}M\texttt{)}$

$(\alpha_1, \ldots, \alpha_n) \leftarrow \texttt{Coordinates(}\mathbf{v}, (\mathbf{b_1}, \ldots, \mathbf{b_n})\texttt{)}$

assert $\alpha_k > 0$ for each $k = 1, \ldots, n$

while $\exists i, j$ *such that* $\mathbf{b_i}_j < 0$ **do**

 $C \leftarrow 1 + \frac{-\mathbf{b_i}_j \alpha_i}{\mathbf{v}_j}$

 $\mathbf{b_i} \leftarrow \mathbf{b_i} + \frac{-\mathbf{b_i}_j}{\mathbf{v}_j} \mathbf{v}$

 $(\alpha_1, \ldots, \alpha_n) \leftarrow (\frac{\alpha_1}{C}, \ldots, \frac{\alpha_n}{C})$

 assert $\alpha_k > 0$ for each $k = 1, \ldots, n$

 assert $\mathbf{v} = \sum_{k=1}^{n} \alpha_k \mathbf{b_k}$

end

assert $\mathbf{b_k} \geq \mathbf{0}$ for each $k = 1, \ldots, n$

return $\{\alpha_1 \mathbf{b_1}, \ldots, \alpha_n \mathbf{b_n}\}$

Decomposed Farkas interpolants

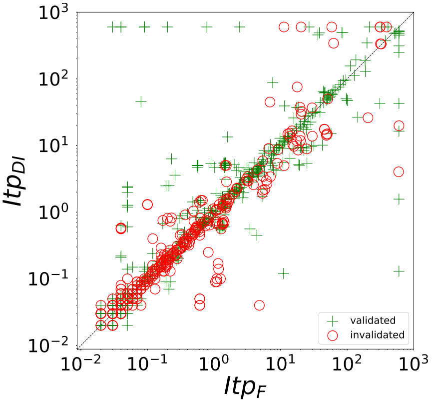
- Decomposition of vector of Farkas coefficients into linearly independent components
  - Conjunction of inequalities
  - Logically stronger than Farkas interpolant

Decomposed Farkas interpolants

- Decomposition of vector of Farkas coefficients into linearly independent components
  - Conjunction of inequalities
  - Logically stronger than Farkas interpolant
- Useful in interpolation-based model checking
  - More precise abstraction

Decomposed Farkas Interpolants
Evaluation

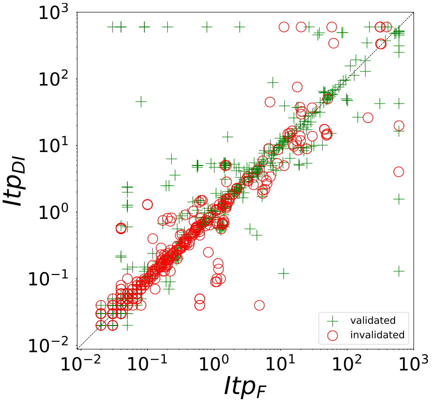
- Implemented in OpenSMT
- Evaluation using model checker Sally (PD-KIND)
  - 1105 benchmarks
  - Farkas interpolants vs decomposed Farkas interpolants



Decomposed Farkas Interpolants

Evaluation

- Implemented in OpenSMT
- Evaluation using model checker Sally (PD-KIND)
  - 1105 benchmarks
  - Farkas interpolants vs decomposed Farkas interpolants
- Neither strictly better $\implies$ portfolio



Our contributions

	concepts	implementation
cooperative layer	IcE/FiRE	SMTS + SALLY
verification layer	**Transition Power Abstraction**	GOLEM
foundational layer	Decomposed Farkas interpolants	OPENSMT

Transition Power Abstraction

Motivation

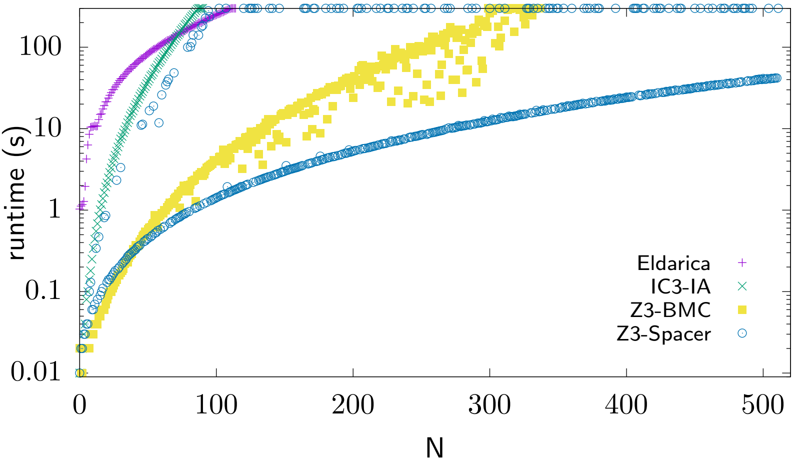
Transition Power Abstraction

Motivation

```
x = 0;
y = N;
while(x < 2N)
{
  x = x + 1;
  if(x > N)
    y = y + 1;
}
assert(y != 2N);
```

Transition Power Abstraction

Motivation



Transition Power Abstraction

Motivation

Problem: Slow progress in search for longer counterexamples

Our solution: Novel use of abstraction for summarization of multiple steps

Transition Power Abstraction

Motivation

Problem: Slow progress in search for longer counterexamples

Our solution: Novel use of abstraction for summarization of multiple steps

Automatically with Craig interpolation

Transition Power Abstraction

Concept

Transition Power Abstraction (TPA) sequence
$TPA^{\leq 0}, TPA^{\leq 1}, \ldots, TPA^{\leq n}, \ldots$

$TPA^{\leq n}(x, x')$

Transition Power Abstraction

Concept

Transition Power Abstraction (TPA) sequence
$TPA^{\leq 0}, TPA^{\leq 1}, \dots, TPA^{\leq n}, \dots$

$TPA^{\leq n}(x, x')$

- overapproximates reachability up to 2^n steps of Tr
  - $Tr^i \subseteq TPA^{\leq n}$ for $0 \leq i \leq 2^n$

Transition Power Abstraction

Concept

Transition Power Abstraction (TPA) sequence
$TPA^{\leq 0}, TPA^{\leq 1}, \ldots, TPA^{\leq n}, \ldots$

$TPA^{\leq n}(x, x')$

- overapproximates reachability up to 2^n steps of Tr
  - $Tr^i \subseteq TPA^{\leq n}$ for $0 \leq i \leq 2^n$
- quantifier-free (only 2 copies of state variables)

Transition Power Abstraction

Concept

Transition Power Abstraction (TPA) sequence
$TPA^{\leq 0}, TPA^{\leq 1}, \ldots, TPA^{\leq n}, \ldots$

$TPA^{\leq n}(x, x')$

- overapproximates reachability up to 2^n steps of Tr
  - $Tr^i \subseteq TPA^{\leq n}$ for $0 \leq i \leq 2^n$
- quantifier-free (only 2 copies of state variables)

- Construction and refinement of the sequence intertwined with bounded reachability checks

Transition Power Abstraction

Main algorithm

Global: $TPA^{\leq 0}, TPA^{\leq 1}, \ldots$ lazy-initialized to $true$

```
CheckSafety(Init, Tr, Bad)
    TPA^{≤0} = Id ∨ Tr
    n = 0
    while true
        if IsReachable(Init, Bad, n)
            return UNSAFE
        n = n + 1
```

Transition Power Abstraction

Main algorithm

Global: $TPA^{\leq 0}, TPA^{\leq 1}, \ldots$ lazy-initialized to $true$

`CheckSafety`$(Init, Tr, Bad)$

1: $TPA^{\leq 0} = Id \vee Tr$

2: $n = 0$

3: **while** $true$

4: **if** `IsReachable`$(Init, Bad, n)$

5: **return UNSAFE**

6: $n = n + 1$

up to 2^{n+1} steps of Tr

Transition Power Abstraction

IsReachable procedure

`IsReachable`$(Source, Target, n)$

1: $res = Sat?[Source(x) \wedge TPA^{\leq n}(x, x') \wedge TPA^{\leq n}(x', x'') \wedge Target(x'')]$

2: **if** $res == UNSAT$

3: $\ldots(x) \wedge Target(x''))$

4:

5: **return false**

6: **else** $//\ res == SAT$

7: **if** $n == 0$

8: **return true**

9: $Intermediate =$ `ExtractIntermediate()`

10: **if not** `IsReachable`$(Source, Intermediate, n-1)$

11: goto 1

12: **if not** `IsReachable`$(Intermediate, Target, n-1)$

13: goto 1

14: **return true**

Is $Target$ reachable from $Source$ in $\leq 2^{n+1}$ steps

Transition Power Abstraction

IsReachable procedure

```
IsReachable(Source, Target, n)
1:  res = Sat?[Source(x) ∧ TPA^{≤n}(x, x') ∧ TPA^{≤n}(x', x'') ∧ Target(x'')]
2:  if res == UNSAT
3:      I = Itp(TPA^{≤n}(x, x') ... ce(x) ∧ Target(x''))
4:      TPA^{≤n+1} = TPA^{≤n+ ...
5:      return false
6:  else // res == SAT
7:      if n == 0
8:          return true
9:      Intermediate = ExtractIntermediate()
10:     if not IsReachable(Source, Intermediate, n−1)
11:         goto 1
12:     if not IsReachable(Intermediate, Target, n−1)
13:         goto 1
14:     return true
```

Abstract path exists?
(SMT query)

Transition Power Abstraction

IsReachable procedure

$\texttt{IsReachable}(Source, Target, n)$

$res = Sat?[Source(x) \wedge TPA^{\leq n}(x, x') \wedge TPA^{\leq n}(x', x'') \wedge Target(x'')]$

if $res == UNSAT$

  $I = Itp(TPA^{\leq n}(x, x') \wedge TPA^{\leq n}(x', x''), Source(x) \wedge Target(x''))$

  $TPA^{\leq n+1} = TPA^{\leq n+1} \wedge I$

  return false

else $// \ res == SAT$

  if $n == 0$

    return true

  $Intermediate = \texttt{ExtractIntermediate()}$

  if not $\texttt{IsReachable}(Source, Intermediate, n{-}1)$

    goto 1

  if not $\texttt{IsReachable}(Intermediate, Target, n{-}1)$

    goto 1

  return true

No abstract path $\implies$ unreachable
New/better abstraction with interpolation

Transition Power Abstraction

IsReachable procedure

`IsReachable`$(Source, Target, n)$

1: $res = Sat?[Source(x) \wedge TPA^{\leq n}(x, x') \wedge TPA^{\leq n}(x', x'') \wedge Target(x'')]$

2: **if** $res == UNSAT$

3: $\quad I = Itp(TPA^{\leq n}(x, x') \wedge TPA^{\leq n}(x', x''), Source(x) \wedge Target(x''))$

4: $\quad TPA^{\leq n+1} = TPA^{\leq n+1} \wedge I$

5: $\quad$ **return false**

6: **else** $// \ res == SAT$

7: $\quad$ **if** $n == 0$

8: $\quad\quad$ **return true**

9: $\quad Intermediate =$ `ExtractIntermediate()`

10: $\quad$ **if not** `IsReachable`$(Source, Intermediate, n-1)$

11: $\quad\quad$ goto 1

12: $\quad$ **if not** `IsReachable`$(Intermediate, Target, n-1)$

13: $\quad\quad$ goto 1

14: $\quad$ **return true**

Recursive refinement

Transition Power Abstraction

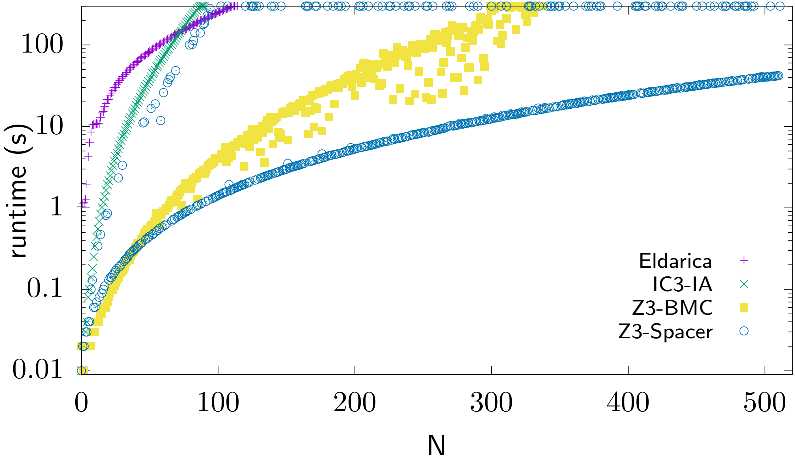
IsReachable procedure

`IsReachable`$(Source, Target, n)$

$res = Sat?[Source(x) \wedge TPA^{\leq n}(x, x') \wedge TPA^{\leq n}(x', x'') \wedge Target(x'')]$
if $res == UNSAT$
    $I = Itp(TPA^{\leq n}(x, x') \wedge TPA^{\leq n}(x', x''), Source(x) \wedge Target(x''))$
    $TPA^{\leq n+1} = TPA^{\leq n+1} \wedge I$
    return false
else $// \; res == SAT$
    if $n == 0$
        return true
    $Intermediate =$ `ExtractIntermediate()`
    if not `IsReachable`$(Source, Intermediate, n{-}1)$
        goto 1
    if not `IsReachable`$(Intermediate, Target, n{-}1)$
        goto 1
    return true

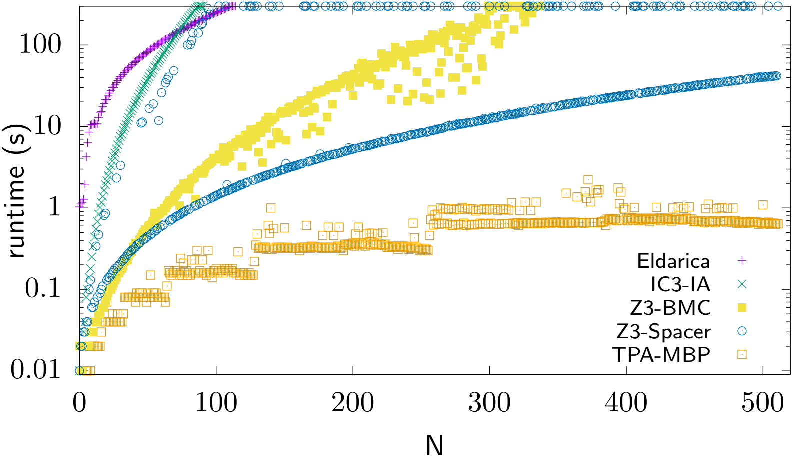
Transition Power Abstraction

Experiments



Transition Power Abstraction

Experiments



Transition Power Abstraction

Proving Safety

- Elements of TPA as candidates for safe inductive transition invariants

Transition Power Abstraction

Proving Safety

- Elements of TPA as candidates for safe inductive transition invariants

- SPLIT-TPA
  - Split the abstraction into $<$ and $=$ parts
  - More candidates for transition invariants
  - Enables k-inductive reasoning

Transition Power Abstraction

Experiments

- TPA and SPLIT-TPA implemented in our Horn solver GOLEM
- 54 benchmarks representing challenging multi-phase loops (from CHC-COMP)
- safe and unsafe version
- timeout 5 minutes

Transition Power Abstraction

Experiments

Benchmark suite	split-TPA	TPA	Z3Spacer	GSpacer	Eldarica
multi-phase unsafe	37 (3)	35 (2)	20 (0)	17 (0)	17 (0)
multi-phase safe	19 (7)	12 (0)	6 (0)	24 (3)	26 (4)

Solved (unique) instances.

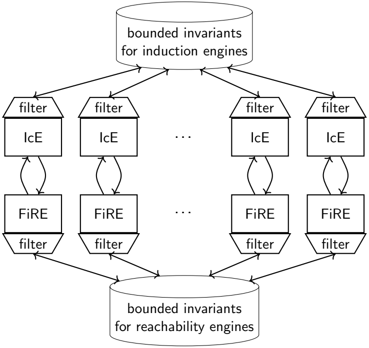
Our contributions

	concepts	implementation
cooperative layer	IcE/FiRE	SMTS + SALLY
verification layer	Transition Power Abstraction	GOLEM
foundational layer	Decomposed Farkas interpolants	OPENSMT

IcE/FiRE cooperative framework

Multi-agent setting

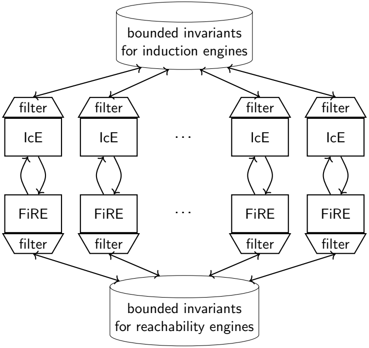
- Abstract framework
  - Reachability analysis
  - Inductive reasoning



IcE/FiRE cooperative framework

Multi-agent setting

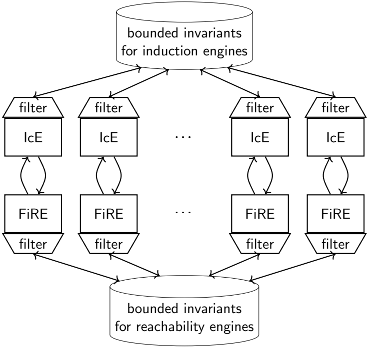
- Abstract framework
  - Reachability analysis
  - Inductive reasoning
- Generalization of PD-KIND [JD16]
  - Cooperation through information exchange



IcE/FiRE cooperative framework

Multi-agent setting

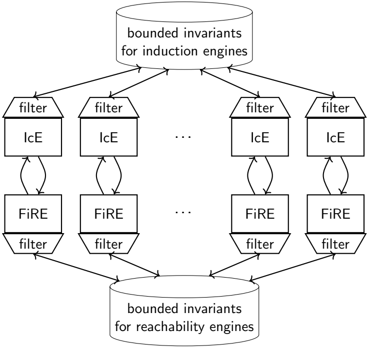
- Abstract framework
  - Reachability analysis
  - Inductive reasoning
- Generalization of PD-KIND [JD16]
  - Cooperation through information exchange
- Parallel PD-KIND



IcE/FiRE cooperative framework

Multi-agent setting

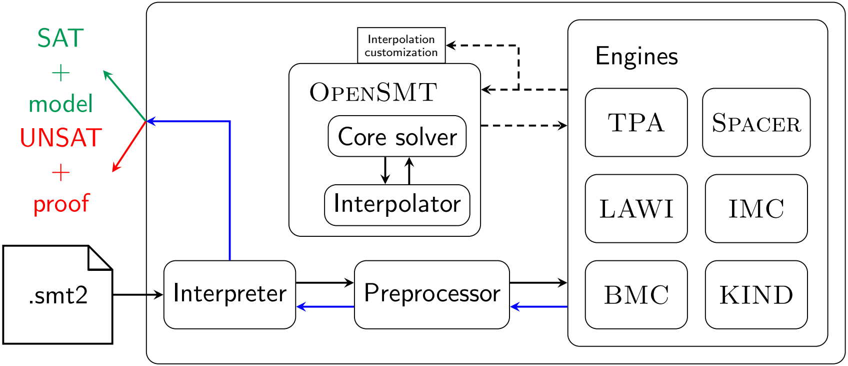
- Abstract framework
  - Reachability analysis
  - Inductive reasoning
- Generalization of PD-KIND [JD16]
  - Cooperation through information exchange
- Parallel PD-KIND
- 4-fold speed-up with 9 instances
  - Information exchange
  - Diverse interpolants



Our contributions

cooperative layer	IcE/FiRE	SMTS + SALLY
verification layer	Transition Power Abstraction	$\boxed{\text{GOLEM}}$
foundational layer	Decomposed Farkas interpolants	OPENSMT
	concepts	implementation

Golem



High-level architecture of our CHC solver Golem

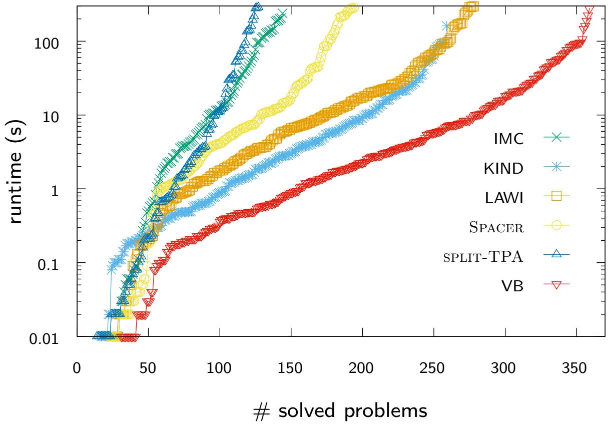
Golem

Evaluation

- CHC-COMP '21
  - 2nd place in LRA-TS track
  - 2nd (3rd) place in LIA-Lin track
- CHC-COMP '22
  - 1st (2nd) place in LRA-TS track
  - 1st (2nd) place in LIA-Lin track
  - 1st (2nd) place in LIA-Nonlin track

Golem

Evaluation



Performance of Golem's engines on SAT benchmarks from LRA-TS track

Summary

- Foundational layer: decomposed Farkas interpolants
  - Logically stronger interpolants $\implies$ more precise abstractions

Summary

- Foundational layer: decomposed Farkas interpolants
  - Logically stronger interpolants $\implies$ more precise abstractions
- Verification layer: TPA and GOLEM
  - Order of magnitude improvement in length of counterexamples detected
  - Automatic discovery of safe transition invariants
  - New CHC solver competitive with state-of-the-art

Summary

- Foundational layer: decomposed Farkas interpolants
  - Logically stronger interpolants $\implies$ more precise abstractions
- Verification layer: TPA and GOLEM
  - Order of magnitude improvement in length of counterexamples detected
  - Automatic discovery of safe transition invariants
  - New CHC solver competitive with state-of-the-art
- Cooperative layer: IcE/FiRE and parallel PD-KIND
  - Cooperation through information exchange
  - 4-fold speed-up with 9 instances in parallel
  - Significant impact of diverse interpolants

Future Work

- TPA
  - hardware model checking
  - liveness properties
  - nonlinear CHCs
- Golem
  - more engines and generalization of existing engines
  - better preprocessing
  - support for more SMT theories
- Parallelization of Golem (in the style of IcE/FiRE)

Publications

Main

- **Blicha, M.**, Hyvärinen, A. E. J., Kofroň, J. and Sharygina, N. [2019]. Decomposing Farkas interpolants, TACAS.
- **Blicha, M.**, Hyvärinen, A. E. J., Marescotti, M. and Sharygina, N. [2020]. A cooperative parallelization approach for property-directed k-induction, VMCAI.
- **Blicha, M.**, Hyvärinen, A. E. J., Kofroň, J. and Sharygina, N. [2022]. Using linear algebra in decomposition of Farkas interpolants, STTT.
- **Blicha, M.**, Fedyukovich, G., Hyvärinen, A. E. J. and Sharygina, N. [2022]. Transition power abstractions for deep counterexample detection, TACAS.
- **Blicha, M.**, Fedyukovich, G., Hyvärinen, A. E. J. and Sharygina, N. [2022]. Split transition power abstractions for unbounded safety, FMCAD.

Publications

Other

- Asadi, S., **Blicha, M.**, Fedyukovich G., Hyvärinen, A. E. J., Even-Mendoza K., Sharygina N. and Chockler H. [2018]. Function Summarization Modulo Theories, LPAR.
- Marescotti, M., **Blicha, M.**, Hyvärinen, A. E. J., Asadi, S. and Sharygina, N. [2018]. Computing Exact Worst-Case Gas Consumption for Smart Contracts. ISoLA.
- Asadi, S., **Blicha, M.**, Hyvärinen, A. E. J., Fedyukovich, G. and Sharygina, N. [2020]. Farkas-Based Tree Interpolation, SAS.
- Asadi, S., **Blicha, M.**, Hyvärinen, A. E. J., Fedyukovich, G. and Sharygina, N. [2020]. Incremental Verification by SMT-based Summary Repair, FMCAD.
- Otoni, R., **Blicha, M.**, Eugster, P., Hyvärinen, A. E. J. and Sharygina, N. [2021]. Theory-Specific Proof Steps Witnessing Correctness of SMT Executions, DAC.
- **Blicha, M.**, Kofroň, J. and Tatarko, W. [2022]. Summarization of Branching Loops, SAC.
- Alt, L., **Blicha, M.**, Hyvärinen, A. E. J. and Sharygina, N. [2022]. SolCMC: Solidity Compiler's Model Checker, CAV.

Our contributions

cooperative layer	IcE/FiRE	SMTS + SALLY
verification layer	Transition Power Abstraction	GOLEM
foundational layer	Decomposed Farkas interpolants	OPENSMT
	concepts	implementation

References I

Dejan Jovanović and Bruno Dutertre.
Property-directed k-induction.
In *2016 Formal Methods in Computer-Aided Design (FMCAD)*, pages 85–92, Oct 2016.

Matteo Marescotti, Antti E. J. Hyvärinen, and Natasha Sharygina.
SMTS: distributed, visualized constraint solving.
In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, 2018.

Philipp Rümmer and Pavle Subotic.
Exploring interpolants.
In *Proc. FMCAD 2013*, pages 69–76. IEEE, 2013.

Tanja Schindler and Dejan Jovanović.
Selfless interpolation for infinite-state model checking.
In Isil Dillig and Jens Palsberg, editors, *VMCAI 2018*, pages 495–515, Cham, 2018. Springer.

Hari Govind Vediramana Krishnan, YuTing Chen, Sharon Shoham, and Arie Gurfinkel.
Global guidance for local generalization in model checking.
In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 101–125, Cham, 2020. Springer International Publishing.

Backup slides

Correction

- A bug in the experimental extension of TPA beyond transition systems
- Too eager to prove safety

Correction

- A bug in the experimental extension of TPA beyond transition systems
- Too eager to prove safety

Golem					
TPA	split-TPA	LAWI	Spacer	Z3-Spacer	Eldarica
~~31~~ 11	~~44~~ 22	12	18	16	36

Table 1: Number of solved benchmarks from `extra-small-lia` subcategory

Constrained Horn Clauses

- Purely logical intermediate language for verification tasks
- Fragment of first-order logic
- $\varphi \land B_1 \land B_2 \land \ldots \land B_n \implies H$
  - φ is a constraint (interpreted over theory $\mathcal{T}$)
  - $B_1, \ldots, B_n$ are uninterpreted predicates
  - H is uninterpreted predcate or $false$
- System of CHCs is
  - SAT, if $\exists$ interpretation of predicates that makes all clauses valid
  - UNSAT, otherwise

Decomposed Farkas interpolants

Farkas interpolant

$$
\begin{aligned}
x_1 &\leq 0 \\
x_2 - x_1 &\leq 0 \\
x_3 - x_1 &\leq 0 \\
-x_2 - x_3 &\leq -1
\end{aligned}
$$

Decomposed Farkas interpolants

Farkas interpolant

$$x_1 \leq 0$$
$$x_2 - x_1 \leq 0$$
$$x_3 - x_1 \leq 0$$
$$-x_2 - x_3 \leq -1$$

$$\cfrac{\cfrac{\cfrac{x_2 - x_1 \leq 0 \quad x_1 \leq 0}{x_2 \leq 0} \qquad \cfrac{x_3 - x_1 \leq 0 \quad x_1 \leq 0}{x_3 \leq 0}}{x_2 + x_3 \leq 0} \qquad -x_2 - x_3 \leq -1}{0 \leq -1}$$

Decomposed Farkas interpolants

Farkas interpolant

$$
\begin{aligned}
x_1 &\leq 0\\
x_2 - x_1 &\leq 0\\
x_3 - x_1 &\leq 0\\
-x_2 - x_3 &\leq -1
\end{aligned}
$$

$$
\dfrac{\dfrac{\dfrac{x_2 - x_1 \leq 0 \quad x_1 \leq 0}{x_2 \leq 0} \qquad \dfrac{x_3 - x_1 \leq 0 \quad x_1 \leq 0}{x_3 \leq 0}}{x_2 + x_3 \leq 0} \qquad -x_2 - x_3 \leq -1}{0 \leq -1}
$$

$$
\begin{array}{lr}
2\times & x_1 \leq 0\\
1\times & x_2 - x_1 \leq 0\\
1\times & x_3 - x_1 \leq 0\\
1\times & -x_2 - x_3 \leq -1\\
\hline\hline
 & 0 \leq -1
\end{array}
$$

Decomposed Farkas interpolants

Farkas interpolant

$$\begin{aligned} x_1 &\le 0 \\ x_2 - x_1 &\le 0 \\ x_3 - x_1 &\le 0 \\ -x_2 - x_3 &\le -1 \end{aligned}$$

$$\dfrac{\dfrac{\dfrac{x_2 - x_1 \le 0 \quad x_1 \le 0}{x_2 \le 0} \qquad \dfrac{x_3 - x_1 \le 0 \quad x_1 \le 0}{x_3 \le 0}}{x_2 + x_3 \le 0} \qquad -x_2 - x_3 \le -1}{0 \le -1}$$

$$\begin{array}{lr} 2\times & x_1 \le 0 \\ 1\times & x_2 - x_1 \le 0 \\ 1\times & x_3 - x_1 \le 0 \\ 1\times & -x_2 - x_3 \le -1 \\ \hline\hline & 0 \le -1 \end{array} \qquad \begin{array}{lr} 2\times & x_1 \le 0 \\ 1\times & x_2 - x_1 \le 0 \\ 1\times & x_3 - x_1 \le 0 \\ & \\ \hline\hline & x_2 + x_3 \le 0 \end{array}$$

Decomposed Farkas interpolants

Farkas interpolant

$$\dfrac{\dfrac{\dfrac{x_2 - x_1 \leq 0 \quad x_1 \leq 0}{x_2 \leq 0} \qquad \dfrac{x_3 - x_1 \leq 0 \quad x_1 \leq 0}{x_3 \leq 0}}{x_2 + x_3 \leq 0} \qquad -x_2 - x_3 \leq -1}{0 \leq -1}$$

$x_1 \leq 0$

$x_2 - x_1 \leq 0$

$x_3 - x_1 \leq 0$

$-x_2 - x_3 \leq -1$

Farkas coefficients

$$\begin{array}{lr} \boxed{2\times} & x_1 \leq 0 \\ \boxed{1\times} & x_2 - x_1 \leq 0 \\ \boxed{1\times} & x_3 - x_1 \leq 0 \\ \boxed{1\times} & -x_2 - x_3 \leq -1 \\ \hline\hline & 0 \leq -1 \end{array} \qquad \begin{array}{lr} 2\times & x_1 \leq 0 \\ 1\times & x_2 - x_1 \leq 0 \\ 1\times & x_3 - x_1 \leq 0 \\ & \text{Farkas interpolant} \\ \hline\hline & \boxed{x_2 + x_3 \leq 0} \end{array}$$

Decomposed Farkas interpolants

Decomposing Farkas interpolant

$$
\begin{array}{rr}
2\times & x_1 \leq 0 \\
1\times & x_2 - x_1 \leq 0 \\
1\times & x_3 - x_1 \leq 0 \\
\hline\hline
 & x_2 + x_3 \leq 0
\end{array}
$$

Decomposed Farkas interpolants

Decomposing Farkas interpolant

$$
\begin{array}{rr}
2\times & x_1 \leq 0 \\
1\times & x_2 - x_1 \leq 0 \\
1\times & x_3 - x_1 \leq 0 \\
\hline\hline
 & x_2 + x_3 \leq 0
\end{array}
$$

$$
\begin{array}{rr}
1\times & x_1 \leq 0 \\
1\times & x_2 - x_1 \leq 0 \\
0\times & x_3 - x_1 \leq 0 \\
\hline\hline
 & x_2 \leq 0
\end{array}
\qquad\qquad
\begin{array}{rr}
1\times & x_1 \leq 0 \\
0\times & x_2 - x_1 \leq 0 \\
1\times & x_3 - x_1 \leq 0 \\
\hline\hline
 & x_3 \leq 0
\end{array}
$$

Decomposed Farkas interpolants

Decomposing Farkas interpolant

$$\begin{array}{rr} 2\times & x_1 \leq 0 \\ 1\times & x_2 - x_1 \leq 0 \\ 1\times & x_3 - x_1 \leq 0 \\ \hline\hline & x_2 + x_3 \leq 0 \end{array}$$

$$\begin{array}{rr} 1\times & x_1 \leq 0 \\ 1\times & x_2 - x_1 \leq 0 \\ 0\times & x_3 - x_1 \leq 0 \\ \hline\hline & x_2 \leq 0 \end{array} \qquad \begin{array}{rr} 1\times & x_1 \leq 0 \\ 0\times & x_2 - x_1 \leq 0 \\ 1\times & x_3 - x_1 \leq 0 \\ \hline\hline & x_3 \leq 0 \end{array}$$

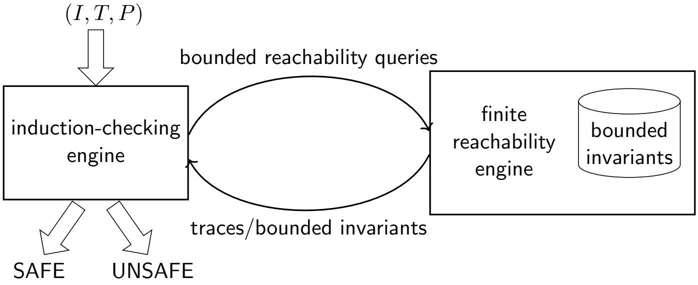
$$\boxed{x_2 \leq 0 \wedge x_3 \leq 0}$$

Decomposed interpolant

IcE/FiRE cooperative framework

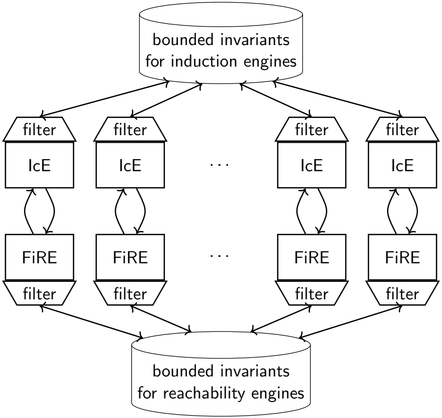
IcE/FiRE cooperative framework

- Abstract framework for model checking with reachability analysis and inductive reasoning
- Explicitly aiming at multi-agent cooperation with information exchange



IcE/FiRE cooperative framework

Multi-agent setting



IcE/FiRE cooperative framework

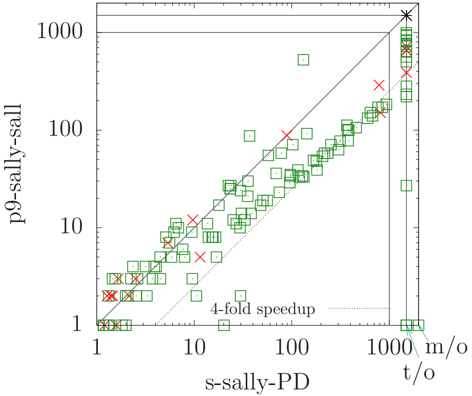
Parallel PD-KIND

- Implementation of parallel PD-KIND
  - Using Sally and SMTS [JD16, MHS18]

IcE/FiRE cooperative framework

Parallel PD-KIND

- Implementation of parallel PD-KIND
  - Using SALLY and SMTS [JD16, MHS18]
- 4-fold speed-up with 9 instances
  - Information exchange both between induction engines and reachability engines



IcE/FiRE cooperative framework

Parallel PD-KIND

- Implementation of parallel PD-KIND
  - Using SALLY and SMTS [JD16, MHS18]
- 4-fold speed-up with 9 instances
  - Information exchange both between induction engines and reachability engines
- Diverse interpolants important
  - $1 + 1 > 6$