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
(Dis)proving correctness with respect to formal specification

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

**Verified**
(Dis)proving correctness with respect to formal specification

Formal methods

Model checking
  - Automated, systematic, exhaustive
  - Symbolic representation using logical formulas

```c
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
- Undecidable problem in general
- Many applications in industry

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

Challenges

- Complexity and scalability
Software Verification

Challenges

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

- state-space explosion

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

Challenges

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

Our contributions

- More precise abstractions with Craig interpolation
- Novel application of abstraction for detecting long counterexamples
- Efficient parallelization with information exchange
Verification in Industry

Verification in Industry

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

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
Our Layered View of Software Verification

cooperative layer

verification layer

foundational layer
Our Layered View of Software Verification

cooperative layer

verification layer

foundational layer
Foundations

- **Satisfiability Modulo Theories (SMT)**
  - Theory of linear arithmetic (reals, integers)
    
    $$x \geq 0 \land (y = x + 1 \lor y = x) \land y < 0$$
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, ...
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
  - \texttt{OpenSMT}, \texttt{Z3}, \texttt{cvc5}, \texttt{Yices}, \texttt{MathSAT}, \texttt{SMTInterpol}, \texttt{Princess}, \ldots

- 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 $A \land B \rightarrow \bot$. Formula $I$ is an interpolant for $A, B$ if

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

Definition (Craig ’57)

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

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

$I_1$
Craig interpolation

Definition (Craig ’57)

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

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

\begin{figure}
\centering
\includegraphics[width=\textwidth]{diagram.png}
\end{figure}
Our Layered View of Software Verification

- Cooperative layer
- Verification layer
- Foundational layer
Checking feasibility of a program path

```c
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

```c
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

```c
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
Checking feasibility of a program path

```c
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

\[ j = 0 \land \neg(i > j) \]
\[ \land r = j \land \neg(r > 0) \]
Checking feasibility of a program path

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

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

\[
\begin{align*}
    j &= 0 \land j > i \\
    \land r &= i \land \neg (r > 0)
\end{align*}
\]

UNSAT

\[
\begin{align*}
    j &= 0 \land \neg (i > j) \\
    \land r &= j \land \neg (r > 0)
\end{align*}
\]

SAT

\[
\begin{align*}
    i &= -1, j = 0, r = 0
\end{align*}
\]
Model Checking

Transition systems

\begin{align*}
\text{x} &= 0; \\
\text{y} &= 0; \\
\text{while} (*) \{ \\
&\quad \text{x} = \text{x} + \text{y}; \\
&\quad \text{y} = \text{y} + 1; \\
\} \\
\text{assert} (\text{x} \geq 0); \\
\text{Init} \equiv x = 0 \wedge y = 0 \\
\text{Tr} \equiv x' = x + y \wedge y' = y + 1 \\
\text{Bad} \equiv \neg (x \geq 0)
\end{align*}
Model Checking

Transition systems

\begin{align*}
  x &= 0; \\
y &= 0; \\
  \text{while} (*) \{ & \quad Init \equiv x = 0 \land y = 0 \\
  & \quad \quad \quad x = x + y; \\
  & \quad \quad \quad y = y + 1; \\
  \} & \quad Tr \equiv x' = x + y \land y' = y + 1 \\
  \text{assert}(x \geq 0); & \quad Bad \equiv \neg(x \geq 0)
\end{align*}

Complex control flow, functions \implies Constrained Horn Clauses (CHC)
SMT-based Model Checking

- Specification
- Program
- Model Checker
- SMT solver

SAFE (+ proof)
UNSAFE + CEX
Our Layered View of Software Verification

- cooperative layer
- verification layer
- foundational layer
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

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

**concepts** vs. **implementation**
Our contributions

cooperative layer
verification layer
foundational layer

IcE/FiRE
Transition Power Abstraction
Decomposed Farkas interpolants
SMTS + Sally
Golem
OpenSMT

concepts
implementation
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
\[2x - y \leq 0\]
\[-x + 2y \leq 0\]
\[x + y \geq 1\]
\[
\begin{align*}
2x - y &\leq 0 \\
-x + 2y &\leq 0 \\
x + y &\geq 1 \\
\end{align*}
\]
\[ 2x - y \leq 0 \]
\[ -x + 2y \leq 0 \]
\[ x + y \geq 1 \]
\[ x \leq 0 \]
\[ y \leq 0 \]
Decomposed Farkas interpolants

- Decomposition of vector of Farkas coefficients into linearly independent components
input: matrix $M$, vector $v$ such that $v \in \ker(M)$ and $v > 0$

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

1. $M \leftarrow \text{RREF}(M)$
2. $n \leftarrow \text{Nullity}(M)$
3. if $n = 1$ then return $\{v\}$
4. $(b_1, \ldots, b_n) \leftarrow \text{KernelBasis}(M)$
5. $(\alpha_1, \ldots, \alpha_n) \leftarrow \text{Coordinates}(v, (b_1, \ldots, b_n))$
6. assert $\alpha_k > 0$ for each $k = 1, \ldots, n$
7. while $\exists i, j$ such that $b_{i,j} < 0$ do
   8. $C \leftarrow 1 + \frac{-b_{i,j}\alpha_i}{v_j}$
   9. $b_i \leftarrow b_i + \frac{-b_{i,j}}{v_j} v$
   10. $(\alpha_1, \ldots, \alpha_n) \leftarrow (\frac{\alpha_1}{C}, \ldots, \frac{\alpha_n}{C})$
   11. assert $\alpha_k > 0$ for each $k = 1, \ldots, n$
   12. assert $v = \sum_{k=1}^{n} \alpha_k b_k$
8. end
14. assert $b_k \geq 0$ for each $k = 1, \ldots, n$
15. return $\{\alpha_1 b_1, \ldots, \alpha_n b_n\}$
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
  → portfolio
Our contributions

cooperative layer
- IcE/FiRE

verification layer
- Transition
  - Power Abstraction

foundational layer
- Decomposed Farkas interpolants
- SMTS + Sally
- Golem
- OpenSMT

concepts

implementation
Transition Power Abstraction

Motivation

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

![Graph showing runtime vs N for different tools: Eldarica, IC3-IA, Z3-BMC, Z3-Spacer. The x-axis represents N, the y-axis represents runtime in seconds.]
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

\[ TPA^{\leq 0}, TPA^{\leq 1}, \ldots, TPA^{\leq n}, \ldots \]

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

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

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

2: $n = 0$

3: while $true$

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

5: return UNSAFE

6: $n = n + 1$
Transition Power Abstraction

Main algorithm

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

CheckSafety($Init, Tr, Bad$)
1: $TPA^{\leq 0} = Id \lor Tr$
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 $Tr$
IsReachable procedure

IsReachable(\textit{Source}, \textit{Target}, n)

1: \texttt{res} = \texttt{Sat}?\left[\textit{Source}(x) \land TPA \leq n(x, x') \land TPA \leq n(x', x'') \land \textit{Target}(x'')\right]

2. if \texttt{res} == \texttt{UNSAT}

3. \texttt{I} = \text{Itpl}(TPA \leq n(x, x') \land TPA \leq n(x', x''), \textit{Source}(x) \land \textit{Target}(x''))

4. \text{TPA} \leq n + 1 = \text{TPA} \leq n + 1 \land \texttt{I}

5. return false

6. else // \texttt{res} == \texttt{SAT}

7. if \texttt{n} == 0

8. return true

9. \texttt{Intermediate} = \text{ExtractIntermediate}()

10. if not IsReachable(\textit{Source}, \textit{Intermediate}, n−1)

11. goto 1

12. if not IsReachable(\textit{Intermediate}, \textit{Target}, n−1)

13. goto 1

14. return true

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

IsReachable procedure

IsReachable(Source, Target, n)
1: res = Sat?[Source(x) \land TPA^{\leq n}(x, x') \land TPA^{\leq n}(x', x'') \land Target(x'')] 
2: if res == UNSAT 
3: I = Itp(TPA^{\leq n}(x, x') \land P_e(x) \land Target(x'')) 
4: TPA^{\leq n+1} = TPA^{\leq n+1} 
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

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') ∧ TPA≤n(x', x''), Source(x) ∧ Target(x''))
4: TPA≤n+1 = TPA≤n+1 ∧ I
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

No abstract path ➞ unreachable
New/better abstraction with interpolation
Transition Power Abstraction

IsReachable procedure

\begin{algorithm}
\textbf{IsReachable}(Source, Target, n)
\begin{algorithmic}[1]
\STATE \texttt{res} = Sat?\[Source(x) \land TPA^\leq_n(x, x') \land TPA^\leq_n(x', x'') \land Target(x'')\]
\IF {\texttt{res} == UNSAT}
\STATE \texttt{I} = Itp(TPA^\leq_n(x, x') \land TPA^\leq_n(x', x''), Source(x) \land Target(x''))
\STATE TPA^\leq_{n+1} = TPA^\leq_{n+1} \land I
\STATE \textbf{return} \texttt{false}
\ELSIF {\texttt{res} == SAT}
\IF {n == 0}
\STATE \textbf{return true}
\ENDIF
\STATE \texttt{Intermediate} = ExtractIntermediate()
\IF {\textbf{not} \texttt{IsReachable}(Source, Intermediate, n - 1)}
\STATE goto 1
\ENDIF
\IF {\textbf{not} \texttt{IsReachable}(Intermediate, Target, n - 1)}
\STATE goto 1
\ENDIF
\STATE \textbf{return true}
\end{algorithmic}
\end{algorithm}

Recursive refinement
Transition Power Abstraction

IsReachable procedure

\textbf{IsReachable}(Source, Target, n)
1: \quad res = Sat?[Source(x) \land TPA^{\leq n}(x, x') \land TPA^{\leq n}(x', x'') \land Target(x'')]
2: \quad \textbf{if} \ res == UNSAT
3: \quad I = Itp(TPA^{\leq n}(x, x') \land TPA^{\leq n}(x', x''), Source(x) \land Target(x''))
4: \quad TPA^{\leq n+1} = TPA^{\leq n+1} \land I
5: \quad \textbf{return false}
6: \quad \textbf{else} // res == SAT
7: \quad \textbf{if} \ n == 0
8: \quad \quad \textbf{return true}
9: \quad Intermediate = ExtractIntermediate()
10: \quad \textbf{if not} IsReachable(Source, Intermediate, n−1)
11: \quad \quad goto 1
12: \quad \textbf{if not} IsReachable(Intermediate, Target, n−1)
13: \quad \quad goto 1
14: \quad \textbf{return true}
Transition Power Abstraction

Experiments

![Graph showing runtime vs. N for different tools: Eldarica, IC3-IA, Z3-BMC, Z3-Spacer. The x-axis represents N, and the y-axis represents runtime in seconds. The graph illustrates the performance comparison of these tools across different values of N.]
Transition Power Abstraction

Experiments

![Graph showing the runtime comparison between different tools: Eldarica, IC3-IA, Z3-BMC, Z3-Spacer, TPA-MBP. The x-axis represents the value of N, and the y-axis represents the runtime in seconds. The graph compares the performance of these tools as N increases.]
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 $\text{SPLIT-TPA}$ implemented in our Horn solver $\text{GOLEM}$
- 54 benchmarks representing challenging multi-phase loops (from CHC-COMP)
- safe and unsafe version
- timeout 5 minutes
## Transition Power Abstraction

### Experiments

<table>
<thead>
<tr>
<th>Benchmark suite</th>
<th><strong>SPLIT-TPA</strong></th>
<th>TPA</th>
<th><strong>Z3Spacer</strong></th>
<th><strong>GSpacer</strong></th>
<th><strong>Eldarica</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>multi-phase unsafe</td>
<td>37 (3)</td>
<td>35 (2)</td>
<td>20 (0)</td>
<td>17 (0)</td>
<td>17 (0)</td>
</tr>
<tr>
<td>multi-phase safe</td>
<td>19 (7)</td>
<td>12 (0)</td>
<td>6 (0)</td>
<td>24 (3)</td>
<td>26 (4)</td>
</tr>
</tbody>
</table>

Solved (unique) instances.
Our contributions

- cooperative layer
  - IcE/FiRE
- verification layer
  - Transition
    - Power Abstraction
- foundational layer
  - Decomposed Farkas interpolants

- SMTS + Sally
- Golem
- OpenSMT

concepts  implementation
IcE/FiRE cooperative framework

Multi-agent setting

- Abstract framework
  - Reachability analysis
  - Inductive reasoning

![Diagram showing the IcE/FiRE cooperative framework with bounded invariants for induction and reachability engines.

- IcE filter
- FiRE filter
- Parallel PD-KIND
  - 4-fold speed-up with 9 instances

Martin Blicha (USI+CUNI) Effective Automated Software Verification Mar 21, '23 33
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

![Diagram showing the cooperative framework with IcE and FiRE filters and bounded invariants for induction and reachability engines.](image-url)
Our contributions

- cooperative layer
- verification layer
- foundational layer

**Implementations:**
- **IcE/FiRE**
- **Transition**
- **Power Abstraction**
- **Decomposed Farkas interpolants**

**Concepts:**
- **SMTS + SALLY**
- **GOLEM**
- **OPENSMRT**

**Implementation:**
- Concepts
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
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

Verification layer: TPA and Golem
- Order of magnitude improvement in length of counterexamples
- 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
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.**, 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., Hyvärinen, A. E. J., Fedyukovich, G. and Sharygina, N. [2020]. Farkas-Based Tree Interpolation, SAS.
- Blicha, M., Kofroň, J. and Tatarko, W. [2022]. Summarization of Branching Loops, SAC.
Our contributions

- Cooperative layer
- Verification layer
- Foundational layer

- IcE/FiRE
- Transition Power Abstraction
- Decomposed Farkas interpolants
- SMTS + Sally
- GoLEM
- OpenSMT

Concepts

Implementation
Dejan Jovanović and Bruno Dutertre.
Property-directed k-induction.

Matteo Marescotti, Antti E. J. Hyvärinen, and Natasha Sharygina.
SMTS: distributed, visualized constraint solving.

Philipp Rümmer and Pavle Subotic.
Exploring interpolants.

Tanja Schindler and Dejan Jovanović.
Selfless interpolation for infinite-state model checking.

Hari Govind Vediramana Krishnan, YuTing Chen, Sharon Shoham, and Arie Gurfinkel.
Global guidance for local generalization in model checking.
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

<table>
<thead>
<tr>
<th>GOLEM</th>
<th>TPA</th>
<th>SPLIT-TPA</th>
<th>LAWI</th>
<th>SPACER</th>
<th>Z3-SPACER</th>
<th>ELDARICA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31</td>
<td>11</td>
<td>44</td>
<td>22</td>
<td>12</td>
<td>18</td>
</tr>
</tbody>
</table>

*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

\( \varphi \land B_1 \land B_2 \land \ldots \land B_n \implies H \)
  - \( \varphi \) is a constraint (interpreted over theory \( T \))
  - \( B_1, \ldots, B_n \) are uninterpreted predicates
  - \( H \) is uninterpreted predicate 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{align*}
x_1 & \leq 0 \\
x_2 - x_1 & \leq 0 \\
x_3 - x_1 & \leq 0 \\
x_2 - x_3 & \leq -1
\end{align*}
Decomposed Farkas interpolants

Farkas interpolant

\[
\begin{align*}
x_1 & \leq 0 \\
x_2 - x_1 & \leq 0 \\
x_3 - x_1 & \leq 0 \\
-x_2 - x_3 & \leq -1
\end{align*}
\]
Decomposed Farkas interpolants

Farkas interpolant

\[ \begin{align*}
  x_1 &\leq 0 \\
  x_2 - x_1 &\leq 0 \\
  x_3 - x_1 &\leq 0 \\
  -x_2 - x_3 &\leq -1
\end{align*} \]

\[ \begin{align*}
  &2 \times x_1 \leq 0 \\
  &1 \times x_2 - x_1 \leq 0 \\
  &1 \times x_3 - x_1 \leq 0 \\
  &1 \times -x_2 - x_3 \leq -1
\end{align*} \]

\[ 0 \leq -1 \]
Decomposed Farkas interpolants

Farkas interpolant

\[
\begin{align*}
x_1 &\leq 0 \\
x_2 - x_1 &\leq 0 \\
x_3 - x_1 &\leq 0 \\
-x_2 - x_3 &\leq -1
\end{align*}
\]

Farkas coefficients

\[
\begin{align*}
2 \times & \quad x_1 \leq 0 \\
1 \times & \quad x_2 - x_1 \leq 0 \\
1 \times & \quad x_3 - x_1 \leq 0 \\
1 \times & \quad -x_2 - x_3 \leq -1
\end{align*}
\]

\[
\begin{align*}
2 \times & \quad x_1 \leq 0 \\
1 \times & \quad x_2 - x_1 \leq 0 \\
1 \times & \quad x_3 - x_1 \leq 0 \\
1 \times & \quad -x_2 - x_3 \leq -1
\end{align*}
\]

\[
\begin{align*}
0 \leq -1
\end{align*}
\]

\[
\begin{align*}
x_2 + x_3 &\leq 0
\end{align*}
\]
Decomposed Farkas interpolants

Farkas interpolant

\[ x_1 \leq 0 \]
\[ x_2 - x_1 \leq 0 \]
\[ x_3 - x_1 \leq 0 \]
\[ -x_2 - x_3 \leq -1 \]

Farkas coefficients

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 \times</td>
<td>( x_1 \leq 0 )</td>
</tr>
<tr>
<td>1 \times</td>
<td>( x_2 - x_1 \leq 0 )</td>
</tr>
<tr>
<td>1 \times</td>
<td>( x_3 - x_1 \leq 0 )</td>
</tr>
<tr>
<td>1 \times</td>
<td>( -x_2 - x_3 \leq -1 )</td>
</tr>
</tbody>
</table>

\[ 0 \leq -1 \]

Farkas interpolant

\[ x_2 + x_3 \leq 0 \]
Decomposed Farkas interpolants

Decomposing Farkas interpolant

\[\begin{align*}
2 \times & \quad x_1 \leq 0 \\
1 \times & \quad x_2 - x_1 \leq 0 \\
1 \times & \quad x_3 - x_1 \leq 0 \\
\hline
& \quad x_2 + x_3 \leq 0
\end{align*}\]
Decomposed Farkas interpolants

Decomposing Farkas interpolant

\[
\begin{align*}
2 \times & \quad x_1 \leq 0 \\
1 \times & \quad x_2 - x_1 \leq 0 \\
1 \times & \quad x_3 - x_1 \leq 0 \\
\hline
1 \times & \quad x_2 + x_3 \leq 0
\end{align*}
\]

\[
\begin{align*}
1 \times & \quad x_1 \leq 0 \\
1 \times & \quad x_2 - x_1 \leq 0 \\
0 \times & \quad x_3 - x_1 \leq 0 \\
\hline
1 \times & \quad x_2 \leq 0
\end{align*}
\]

\[
\begin{align*}
1 \times & \quad x_1 \leq 0 \\
0 \times & \quad x_2 - x_1 \leq 0 \\
1 \times & \quad x_3 - x_1 \leq 0 \\
\hline
1 \times & \quad x_3 \leq 0
\end{align*}
\]
Decomposed Farkas interpolants

Decomposing Farkas interpolant

\[
\begin{align*}
2 \times x_1 & \leq 0 \\
1 \times x_2 - x_1 & \leq 0 \\
1 \times x_3 - x_1 & \leq 0 \\
\hline
x_2 + x_3 & \leq 0
\end{align*}
\]

\[
\begin{align*}
1 \times x_1 & \leq 0 \\
1 \times x_2 - x_1 & \leq 0 \\
0 \times x_3 - x_1 & \leq 0 \\
\hline
x_2 & \leq 0 \\
1 \times x_3 - x_1 & \leq 0 \\
\hline
x_3 & \leq 0
\end{align*}
\]

\[
x_2 \leq 0 \land x_3 \leq 0
\]

Decomposed interpolant
IcE/FiRE cooperative framework

Abstract framework for model checking with reachability analysis and inductive reasoning

Explicitly aiming at multi-agent cooperation with information exchange

Induction-checking engine

Finite reachability engine

Bounded invariants

Bounded reachability queries

Traces/bounded invariants

\((I, T, P)\)

SAFE

UNSAFE

Martin Blichá (USI+CUNI)
IcE/FiRE cooperative framework

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

\[(I, T, P)\]

**Diagram:**
- Induction-checking engine
- Bounded reachability queries
- Traces/bounded invariants
- SAFE
- UNSAFE
- Finite reachability engine
- Bounded invariants

Martin Blicha (USI+CUNI)
Effective Automated Software Verification
Mar 21, '23
IcE/FiRE cooperative framework

Multi-agent setting

bounded invariants for induction engines

bounded invariants for reachability engines

filter

IcE

FiRE

filter

filter

filter

filter

filter

filter
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

$1 + 1 > 6$
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
  - $1 + 1 > 6$