Towards Practical Predicate Analysis

Philipp Wendler

2017-11-20
Why Automatic Software Verification?

- Software is critical in today’s world
- Software has bugs
- Software is too complex for humans to find all bugs
Why Automatic Software Verification?

- Software is critical in today’s world
- Software has bugs
- Software is too complex for humans to find all bugs

Solution “Model Checking”:

Program

Specification

TRUE

FALSE
Why Automatic Software Verification?

- Software is critical in today’s world
- Software has bugs
- Software is too complex for humans to find all bugs

Solution “Model Checking”:

Program

Specification

TRUE

UNKNOWN [Turing37]

FALSE
Many approaches share common idea:

▶ Convert parts of program into formulas of first-order logic (satisfiability modulo theories, SMT)
  Example: $x > 0 \land y < 10$

▶ Query solver about satisfiability

We call SMT-based approaches *predicate analyses*.

▶ Leverage power of modern solvers for model checking
▶ Used in practice for verification of software
Existing Predicate Analyses

- Predicate Abstraction
- **IMPACT**
- Bounded Model Checking
- *k*-Induction
- Property-Directed Reachability
- ...

Philipp Wendler
Existing Predicate Analyses and Tools

- Predicate Abstraction
  \((\text{Blast}, \text{Slam}, \ldots)\)

- **IMPACT**
  \((\text{Impact}, \text{Wolverine}, \ldots)\)

- Bounded Model Checking
  \((\text{CBMC}, \text{ESBMC}, \ldots)\)

- \(k\)-Induction
  \((\text{ESBMC}, 2\text{LS}, \ldots)\)

- Property-Directed Reachability
  \((\text{SeaHorn}, \text{VVT}, \ldots)\)

- \(\ldots\)
Problems with State of the Art

Approaches exist in isolation

▶ Hard to see differences and commonalities
▶ Hard to understand key concepts and advantages
▶ Slows down research
Problems with State of the Art

Approaches exist in isolation
▶ Hard to see differences and commonalities
▶ Hard to understand key concepts and advantages
▶ Slows down research

Approaches implemented in different tools
▶ ... of varying quality (academic prototypes)
▶ Experimental comparison of approaches difficult
▶ Slows down research and adoption in practice
Question at a Recent Workshop

Which do you think is better, i.e., solves more tasks?

- $k$-Induction
- Predicate Abstraction

Depending on configuration, either (A) or (B) can be true!

Technical details (e.g., choice of SMT theory) influence evaluation of algorithms.
Question at a Recent Workshop

Which do you think is better, i.e., solves more tasks?

(A) $k$-Induction solves 29% more tasks

(B) Predicate Abstraction solves 3% more tasks

Depending on configuration, either (A) or (B) can be true!

Technical details (e.g., choice of SMT theory) influence evaluation of algorithms.
Question at a Recent Workshop

Which do you think is better, i.e., solves more tasks?

(A) $k$-Induction solves 29\% more tasks

(B) Predicate Abstraction solves 3\% more tasks

Depending on configuration, either (A) or (B) can be true!

Technical details (e.g., choice of SMT theory) influence evaluation of algorithms
Goals

1. Provide a unifying framework for predicate analyses
2. Understand differences and key concepts of approaches
3. Determine potential of extensions and combinations
4. Provide solid platform for experimental research
Approach

- Understand, and, if necessary, re-formulate algorithms
- Design a configurable framework for predicate analyses (as configurable program analysis)
- Express existing algorithms using the common framework
  - Predicate abstraction
  - IMPACT
  - Bounded model checking
  - $k$-Induction
- Implement framework (in CPAchecker)
Using CPA framework:

- Reuse existing CPA algorithm for state-space exploration
- CPA algorithm uses operators provided by CPA(s)
- Predicate CPA provides predicate-based operators
- Algorithms for additional features implemented on top of CPA algorithm
Predicate CPA

\[ D_P = (C, \mathcal{E}_P, \mathcal{E}_P) \]

\[ \Pi_P \]

\[ \lnot_P \]

merge_P

stop_P

prec_P

fcover_P

refine_P

Philipp Wendler 10 / 24
Predicate CPA

- $D_P = (C, E_P, \cdot_P)$
- $\Pi_P$
- $\approx_P$
- `merge_P`
- `stop_P`
- `prec_P`
- `fcover_P`
- `refine_P`

Region Representation

- BDD
- SMT-based
Predicate CPA

\[ D_P = (C, \varepsilon_P, \llbracket \cdot \rrbracket_P) \]

Strongest Postcondition

Region Representation

BDD

SMT Theory

Arrays+ Lin. Arith.

Bitvectors+ Quantifiers

Predicate CPA \( \Pi_P \)

merge\( \_P \)

stop\( \_P \)

prec\( \_P \)

fcover\( \_P \)

refine\( \_P \)

fcover\( \_P \)

Impact

refine\( \_P \)

Path Invariants

Unsat Cores

Weakest Preconditions

Heuristic Predicates

Refinement Strategy

Abstract Facts

Interpolants

Predicate Impact

Predicate CPA
Predicate CPA

\[ D_{P} = (C, \xi_{P}, [.]_{P}) \]

\[ \Pi_{P} \]

\[ \rightsquigarrow_{P} \]

\[ \text{merge}_{P} \]

\[ \text{stop}_{P} \]

\[ \text{prec}_{P} \]

\[ \text{fcover}_{P} \]

\[ \text{refine}_{P} \]

\[ \text{Region Representation} \]

\[ \text{Strongest Postcondition} \]

\[ \text{blk} \]

\[ \text{Predicate Abstraction} \]

\[ \text{BDD} \]

\[ \text{SMT Theory} \]

\[ \text{SMT-based} \]

\[ \text{Arrays+Lin. Arith.} \]

\[ \ldots \]

\[ \text{Bitvectors+Quantifiers} \]

\[ \text{blk}^{SBE} \]

\[ \text{Cartesian} \]

\[ \text{Boolean} \]

\[ \text{blk}^{l} \]

\[ \text{blk}^{lf} \]

\[ \text{blk}^{never} \]
Predicate CPA

\[ D_P = (C, \varepsilon_P, [\cdot]_P) \]

\[ \Pi_P \]

\[ \leadsto_P \]

\[ \text{merge}_P \]

\[ \text{stop}_P \]

\[ \text{prec}_P \]

\[ \text{fcover}_P \]

\[ \text{refine}_P \]

\[ \text{Region Representation} \]

\[ \text{Strongest Postcondition} \]

\[ \text{blk} \]

\[ \text{Predicate Abstraction} \]

\[ \text{SMT Theory} \]

\[ \text{SMT-based} \]

\[ \text{Arrays+ Lin. Arith.} \]

\[ \text{Bitvectors+ Quantifiers} \]

\[ \text{BDD} \]

\[ \text{Cartesian} \]

\[ \text{Boolean} \]

\[ \text{fcover}^{id} \]

\[ \text{fcover}^{\text{IMPACT}} \]

\[ \text{blk}^{\text{SBE}} \]

\[ \text{blk}^{l} \]

\[ \text{blk}^{lf} \]

\[ \text{blk}^{never} \]

Strongest Postcondition

Region Representation

Predicate CPA

SMT Theory

SMT-based

Arrays+ Lin. Arith.

Bitvectors+ Quantifiers

BDD

Cartesian

Boolean

fcover^{id}

fcover^{\text{IMPACT}}

\[ \text{blk} \]
Predicate CPA

$D_P = (C, \varepsilon_P, [\cdot]_P)$

Region Representation

Strongest Postcondition

Predicate CPA $\Pi_P$

Predicate Abstraction

Refinement Strategy

Abstract Facts

Predicate

Interpolants

Path Invariants

Unsat Cores

Weakest Preconditions

Heuristic Predicates

SMT-based

SMT Theory

Arrays+ Lin. Arith.

Bitvectors+ Quantifiers

SBE

Blk

Cartesian

Block

Boolean

F cover

f cover $^{id}$

f cover $^{IMPACT}$

f cover $^{fcover}$

Predicates

Predicate CPA

Refinement Strategy

Abstract Facts

Interpolants

Path Invariants

Unsat Cores

Weakest Preconditions

Heuristic Predicates

SMT-based

SMT Theory

Arrays+ Lin. Arith.

Bitvectors+ Quantifiers

SBE

Blk

Cartesian

Block

Boolean

F cover

f cover $^{id}$

f cover $^{IMPACT}$

f cover $^{fcover}$

Predicates
Predicate Abstraction

- [CAV’97, POPL’02, J. ACM’03, POPL’04]
- Abstract-interpretation technique
- Abstract domain constructed from a set of predicates $\pi$
- Use CEGAR to add predicates to $\pi$ (refinement)
- Derive new predicates using Craig interpolation
Predicate CPA for Predicate Abstraction

$D_P = (C, \varepsilon_P, [\cdot]_P)$

$\Pi_P \leadsto \Pi_P$  
$\Pi_P$  
$\text{merge}_P$  
$\text{stop}_P$  
$\text{prec}_P$  
$\text{fcover}_P$  
$\text{refine}_P$  

Region Representation

Strongest Postcondition

Predicate Abstraction

Abstract Facts

Refinement Strategy

$\text{fcover}^{id}$

$\text{fcover}^{IMPACT}$

Interpolants

Path Invariants

Unsat Cores

Weakest Preconditions

Heuristic Predicates

Arrays+ Lin. Arith.

Bitvectors+ Quantifiers

BDD

SMT Theory

SMT-based

Cartesian

Boolean

$\text{blk}^{SBE}$

$\text{blk}^l$

$\text{blk}^{lf}$

$\text{blk}^{never}$

Philipp Wendler 12 / 24
Predicate CPA for Predicate Abstraction

\[ D_P = (C, \varepsilon_P, \llbracket \cdot \rrbracket_P) \]

\[ \Pi_P \]

\[ \sim_P \]

merge\(_P\)

stop\(_P\)

prec\(_P\)

fcover\(_P\)

refine\(_P\)

Region Representation

Strongest Postcondition

Predicate Abstraction

Refinement Strategy

Abstract Facts

Interpolants

Predicate

SMT Theory

BDD

SMT-based

Arrays+ Lin. Arith.

Bitvectors+ Quantifiers

blk

blk\(^{SBE}\)

blk\(^{l}\)

blk\(^{lf}\)

blk\(^{never}\)

Cartesian

Boolean

fcover\(^{id}\)

fcover\(^{IMPACT}\)

Path Invariants

Unsat Cores

Weakest Preconditions

Heuristic Predicates

Weakest Preconditions

Heuristic Predicates

Path Invariants

Unsat Cores

Weakest Preconditions

Heuristic Predicates
Example Program

```c
int main() {
    int x = 0;
    x = x + 1;
    if (*) {
        x = x + 1;
        return 0;
    } else {
        if (x <= 0) {
            ERROR:
            return 1;
        } else {
            return 0;
        }
    }
}
```
Predicate Abstraction: Example

with initial precision: $\pi = \{\}$

Diagram:

- Start at $l_2$
- $l_2 \rightarrow l_3$
  - $\text{int } x = 0$
- $l_3 \rightarrow l_4$
  - $x = x + 1$
- $l_4 \rightarrow l_5$
  - $x = x + 1$
- $l_5 \rightarrow l_6$
  - $\text{return 0}$
- $l_6 \rightarrow l_7$
  - $\text{ERROR}$
- $l_5 \leftarrow l_8$
  - $[x \leq 0]$
- $l_8 \rightarrow l_9$
  - $[x > 0]$
- $l_9 \rightarrow l_{10}$
  - $\text{return 0}$
- $l_{10} \rightarrow l_{13}$
  - $l_{12} \rightarrow l_{13}$
  - $l_{12} \rightarrow l_{13}$
Predicate Abstraction: Example

with initial precision: $\pi = \{\}$

```
int x = 0;
x = x + 1;
x = x + 1;
return 0;
```

```
[x <= 0] ERROR
[x > 0] return 0;
```
Predicate Abstraction: Example

with initial precision: \( \pi = \{ \} \)

\[
\begin{align*}
\text{int } x & = 0; \\
x & = x + 1; \\
x & = x + 1; \\
\text{return 0;}
\end{align*}
\]

\[
\begin{align*}
[l_2, \text{true}] \\
[l_3, \text{true}] \\
[l_4, \text{true}]
\end{align*}
\]
Predicate Abstraction: Example

with initial precision: $\pi = \{}$
Predicate Abstraction: Example

with initial precision: $\pi = \emptyset$

```
int x = 0;
x = x + 1;
x = x + 1;
return 0;
[x <= 0] ERROR
[x > 0] return 0;
```

Diagram:

- Start at $l_2$
- $l_3$: $x = x + 1$
- $l_4$: $x = x + 1$ (branching)
- $l_5$: $x = x + 1$ (branching)
- $l_6$: return 0
- $l_7$: ERROR
- $l_8$: $x > 0$
- $l_9$: [x <= 0]
- $l_{10}$: return 0
- $l_{12}$: return 0
- $l_{13}$: return 0
- $l_2, true$
- $l_3, true$
- $l_4, true$
- $l_5, true$
- $l_6, true$
- $l_8, true$
Predicate Abstraction: Example

with initial precision: $\pi = \{\}$

```
int x = 0;
x = x + 1;
x = x + 1;
return 0;
```

$\{x \leq 0\}$: ERROR

$\{x > 0\}$: return 0;
Predicate Abstraction: Example

with initial precision: $\pi = \{\}$

```
int x = 0;
x = x + 1;
x = x + 1;
return 0;
```

$[x \leq 0]$ ERROR $[x > 0]$ return 0;
Predicate Abstraction: Example

with initial precision: $\pi = \{\}$

\[\begin{align*}
\text{start} & \rightarrow l_2 \\
& \quad \text{int } x = 0; \\
& \quad l_3 \\
& \quad x = x + 1; \\
& \quad l_4 \\
& \quad l_5, x = x + 1; \\
& \quad l_6, x <= 0; \\
& \quad l_7, \text{ERROR} \\
& \quad l_8, x > 0; \\
& \quad l_9, \text{return 0;} \\
& \quad l_{10} \\
& \quad l_{11} \\
& \quad l_{12, \text{true}} \\
& \quad l_{13, \text{true}} \\
& \quad l_9, \text{return 0;} \\
& \quad l_7, \text{true} \\
& \quad l_6, \text{true} \\
& \quad l_5, \text{true} \\
& \quad l_4, \text{true} \\
& \quad l_3, \text{true} \\
& \quad l_2, \text{true}
\end{align*}\]
Predicate Abstraction: Example

with initial precision: $\pi = \{\}$
1. Convert path $< l_2, \ldots, l_{10} >$ into formula:

$$x_1 = 0 \land x_2 = x_1 + 1 \land x_2 \leq 0$$
Predicate Abstraction: Example

Refinement

1. Convert path $< l_2, \ldots, l_{10} >$ into formula:
   
   $$x_1 = 0 \land x_2 = x_1 + 1 \land x_2 \leq 0$$

2. Check for satisfiability: unsat!

---

**Diagram:**

- Start at $l_2$
- $\text{int } x = 0$
- $x = x + 1$
- $x = x + 1$
- $x \leq 0$ (ERROR)
- $x > 0$
- $\text{return 0}$
Predicate Abstraction: Example

**Refinement**

1. Convert path \(< l_2, \ldots, l_{10} >\) into formula:
   \[ x_1 = 0 \land x_2 = x_1 + 1 \land x_2 \leq 0 \]

2. Check for satisfiability: unsat!

3. Let solver compute interpolants:

   \[ x_1 = 0 \]

   \[ x_1 \geq 0 \]

   \[ x_2 = x_1 + 1 \]

   \[ x_2 > 0 \]

   \[ x_2 \leq 0 \]
Predicate Abstraction: Example

Refinement

1. Convert path $< l_2, \ldots, l_{10} >$ into formula:
   $$ x_1 = 0 \land x_2 = x_1 + 1 \land x_2 \leq 0 $$

2. Check for satisfiability: unsat!

3. Let solver compute interpolants:
   $$ x_1 = 0 
   \begin{align*}
   x_1 & \geq 0 \\
   x_2 & = x_1 + 1 \\
   x_2 & > 0 \\
   x_2 & \leq 0
   \end{align*} $$

4. Add interpolants to precision:
   $$ \pi = \{ x \geq 0, x > 0 \} $$
Predicate Abstraction: Example

with precision after refinement: \( \pi = \{ x \geq 0, x > 0 \} \)
Predicate Abstraction: Example

with precision after refinement: $\pi = \{x \geq 0, x > 0\}$
Predicate Abstraction: Example

with precision after refinement: \( \pi = \{ x \geq 0, x > 0 \} \)
Predicate Abstraction: Example

with precision after refinement: \( \pi = \{x \geq 0, x > 0\} \)

\[
\begin{align*}
\text{start} & \rightarrow l_2 \\
l_2 & \rightarrow \text{int } x = 0; \\
l_3 & \rightarrow x = x + 1; \\
l_4 & \rightarrow x = x + 1; \\
l_5 & \rightarrow x = x + 1; \quad [x \leq 0] \\
l_6 & \rightarrow \text{return 0;} \\
l_7 & \rightarrow \text{ERROR} \\
l_8 & \rightarrow [x > 0] \\
l_9 & \rightarrow \text{return 0;} \\
l_{10} & \rightarrow l_5, x \geq 0 \land x > 0 \\
l_{12} & \rightarrow l_8, x \geq 0 \land x > 0 \\
l_{13} & \rightarrow l_8, x \geq 0 \land x > 0 \\
l_2, \text{true} & \rightarrow l_3, x \geq 0 \\
l_3, x \geq 0 & \rightarrow l_4, x \geq 0 \land x > 0 \\
l_4, x \geq 0 \land x > 0 & \rightarrow l_5, x \geq 0 \land x > 0 \\
l_5, x \geq 0 \land x > 0 & \rightarrow l_8, x \geq 0 \land x > 0 \\
\end{align*}
\]
Predicate Abstraction: Example

with precision after refinement: \( \pi = \{ x \geq 0, x > 0 \} \)

\[\text{start} \rightarrow l_2 \]

\[\text{int } x = 0; \]

\[l_3 \]

\[x = x + 1; \]

\[l_4 \]

\[l_5 \]

\[x = x + 1; \]

\[l_6 \]

\[x \leq 0 \]

\[\text{return 0; } \]

\[l_7 \]

\[x > 0 \]

\[\text{ERROR} \]

\[l_8 \]

\[\text{return 0;} \]

\[l_9 \]

\[l_{10} \]

\[l_{12} \]

\[l_{13} \]
Predicate Abstraction: Example

with precision after refinement: \( \pi = \{ x \geq 0, x > 0 \} \)
Predicate Abstraction: Example

with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

```java
int x = 0;
x = x + 1;
x = x + 1;
return 0;
```
Predicate Abstraction: Example

with precision after refinement: \( \pi = \{ x \geq 0, x > 0 \} \)
Predicate Abstraction: Example

with precision after refinement: \( \pi = \{ x \geq 0, x > 0 \} \)
Predicate Abstraction: Example

with precision after refinement: \( \pi = \{ x \geq 0, x > 0 \} \)
Predicate Abstraction: Example

with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

\begin{align*}
\text{int } x &= 0; \\
x &= x + 1; \\
\end{align*}

\begin{align*}
\text{return 0; } & \quad \text{ERROR} \\
\end{align*}
Impact

- “Lazy Abstraction with Interpolants” [CAV’06]
- Abstraction is derived dynamically/lazily
- Solution to avoiding expensive abstraction computations
- Compute fixed point over three operations
  - Expand
  - Refine
  - Cover
Predicate CPA for \textbf{IMPACT}

\[ D_P = (C, \mathcal{E}_P, [])_P \]

- **Region Representation**
- **Strongest Postcondition**
- **Predicate Abstraction**
- **Abstract Facts**
- **Refinement Strategy**

- **SMT Theory**
  - BDD
  - SMT-based
  - Arrays + Lin. Arith.
  - Bitvectors + Quantifiers
  - \ldots
- **Predicat Abstraction**
  - \( \text{blk}^{SBE} \)
  - \( \text{blk}^l \)
  - \( \text{blk}^n_{\text{ever}} \)
- **Refine**
  - \( \text{fcover}^{id} \)
  - \( \text{fcover}^\text{IMPACT} \)
- **Interpolants**
- **Path Invariants**
- **Unsat Cores**
- **Weakest Preconditions**
- **Heuristic Predicates**

**Heuristic**

**Abstract**

**Facts**

**Path**

**Invariants**

**Unsat**

**Cores**

**Weakest**

**Preconditions**

**Predicate**

**IMPACT**

**Predicates**

**Impact**

**Abstract**

**Facts**

**Path**

**Invariants**

**Unsat**

**Cores**

**Weakest**

**Preconditions**

**Predicate**

**IMPACT**

**Predicates**
**Impact:** Example

with precision $\pi = \{\}$

```
int x = 0;
x = x + 1;
x = x + 1;
return 0;
```

Refinement of path $\langle l_2, \ldots, l_{10} \rangle$ using interpolation.

$l_2, true$
**Impact:** Example

with precision $\pi = \{\}$

```
int x = 0;
x = x + 1;
int x = 0;
x = x + 1;
```

Refinement of path $<l_2, ..., l_{10}>$ using interpolation
**Impact:** Example

with precision $\pi = \{\}$

start $\rightarrow l_2$

$\text{int } x = 0;$

$l_2$ $\rightarrow l_3$

$x = x + 1;$

$l_3$ $\rightarrow l_4$

$x = x + 1;$

$l_4$ $\rightarrow l_5 \rightarrow l_6 \rightarrow l_7$

$\text{return } 0;$

$x = x + 1;$

$l_5$ $\rightarrow l_9$ $\rightarrow l_{10}$$\rightarrow l_{13}$

ERROR

$[x \leq 0]$

$l_6$ $\rightarrow l_9$$\rightarrow l_{12}$

$[x > 0]$

$l_8$ $\rightarrow l_{12}$

$\text{return } 0;$

$l_7$$\rightarrow l_{10}$

$l_2, true$

$l_3, true$

$l_4, true$
**Impact:** Example

with precision $\pi = \{\}$

\[
\begin{align*}
    \text{int } x & = 0; \\
    x & = x + 1; \\
\end{align*}
\]

---

Refinement of path $\langle l_2, \ldots, l_{10} \rangle$ using interpolation

---

Philipp Wendler

19 / 24
**Impact:** Example

with precision $\pi = \{\}$

```
int x = 0;
x = x + 1;
```

Refinement of path $\langle l_2, \ldots, l_9 \rangle$ using interpolation:
**Impact:** Example

with precision $\pi = \{\}$
**Impact**: Example

with precision $\pi = \{\}$

\[
\begin{align*}
\text{start} & \rightarrow l_2 \\
l_2 & \rightarrow \text{int } x = 0; \\
l_3 & \rightarrow x = x + 1; \\
l_4 & \rightarrow \text{[x <= 0]} \\
l_5 & \rightarrow x = x + 1; \\
l_6 & \rightarrow \text{return 0; } \\
l_7 & \rightarrow \text{ERROR} \\
l_8 & \rightarrow \text{[x > 0]} \\
l_9 & \rightarrow \text{return 0; } \\
l_{10} & \rightarrow \text{return 0; } \\
l_{12} & \rightarrow \text{return 0; }
\end{align*}
\]
**Impact:** Example

with precision $\pi = \{\}$
**Impact:** Example

with precision $\pi = \emptyset$

Refinement of path $< l_2, \ldots, l_{10} >$ using interpolation
**Impact:** Example

with precision $\pi = \{ \}$

Refinement of path $< l_2, \ldots, l_{10} >$ using interpolation
**Impact:** Example

with precision $\pi = \{\}$

```
int x = 0;
x = x + 1;
```

**Refinement of path $<l_2, ..., l_{10}>$ using interpolation**

```
l_2, true

l_3, x \geq 0

l_4, x > 0

l_5, true

l_6, true

l_7, true

ERROR

l_10, false

l_12, true

l_8, x > 0

l_9, false

l_13, true
```
**Impact:** Example

with precision $\pi = \emptyset$

```
int x = 0;
x = x + 1;
```

Refinement of path $\langle l_2, \ldots, l_{10} \rangle$ using interpolation.
Insights

差异的特征

差异的特征 

显式差异通过框架配置选项

我们知道只有这些差异是相关的！

谓词抽象需要创建更一般化的抽象模型

IMPACT更懒惰，但这也可能导致许多细化

→ 强制覆盖被提议作为优化

新特征的组合可能

例：IMPACT与可调整块编码减少大量细化
Insights

- **Difference between predicate abstraction and IMPACT:**
  - Differences explicit via configuration options of framework
  - 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 proposed as optimization

- **New combinations of features possible**
  - Example: IMPACT with adjustable-block encoding reduces high number of refinements
Evaluation: Usefulness of Framework

- 4 existing approaches successfully integrated
- Ongoing projects for integration of further approaches
- Interesting insights learned about these approaches
- High configurability allows new combinations and hybrid approaches
Evaluation: Usefulness of Framework

- 4 existing approaches successfully integrated
- Ongoing projects for integration of further approaches
- Interesting insights learned about these approaches
- High configurability allows new combinations and hybrid approaches
- Already used as base for successful research projects by other researchers, e.g.
  - Block-abstraction memoization [ICFEM’12]
  - Refinement selection [SPIN’15]
  - Local policy iteration [VMCAI’16]
  - ...
Evaluation: Usefulness of Implementation

- Used in other research projects:
  - Conditional model checking [FSE’12]
  - Verifying recursive programs [SAS’14]
  - Verification witnesses [FSE’15, FSE’16]
Evaluation: Usefulness of Implementation

- Used in other research projects:
  - Conditional model checking [FSE’12]
  - Verifying recursive programs [SAS’14]
  - Verification witnesses [FSE’15, FSE’16]

- Enables experimental studies:
  - SMT-based software model checking:
    An experimental comparison of four algorithms [VSTTE’16]
  - Comparison of SMT solvers and theories
    - 120 different configurations on 5,594 programs
    - Important insights on how SMT solvers and theories can influence benchmark results and skew conclusion

Both studies not possible before!
Evaluation: Comparison with State of the Art

▶ State of the art visible in Intl. Competition on Software Verification (SV-COMP)

▶ Implementation won 4 medals in first year (SV-COMP’12)

▶ Contributed to 40 more medals

▶ Awarded Gödel medal by Kurt Gödel Society
Conclusion

1. Provide a unifying framework for predicate analyses

2. Understand differences and key concepts of approaches

3. Determine potential of extensions and combinations

4. Provide solid platform for experimental research
Conclusion

1. Provide a unifying framework for predicate analyses
   - Formally defined and used for 4 different analyses:
     - accepted for Journal of Automated Reasoning (JAR)
     - Adopted by others for even more analyses

2. Understand differences and key concepts of approaches

3. Determine potential of extensions and combinations

4. Provide solid platform for experimental research
Conclusion

1. Provide a unifying framework for predicate analyses
   ▶ Formally defined and used for 4 different analyses:
     accepted for Journal of Automated Reasoning (JAR)
     Adopted by others for even more analyses

2. Understand differences and key concepts of approaches
   ▶ Interesting insights found

3. Determine potential of extensions and combinations

4. Provide solid platform for experimental research
Conclusion

1. Provide a unifying framework for predicate analyses
   - Formally defined and used for 4 different analyses: accepted for Journal of Automated Reasoning (JAR)
   - Adopted by others for even more analyses

2. Understand differences and key concepts of approaches
   - Interesting insights found

3. Determine potential of extensions and combinations
   - New combinations of features and algorithms immediately available
   - Large potential for future work

4. Provide solid platform for experimental research
Conclusion

1. Provide a unifying framework for predicate analyses
   - Formally defined and used for 4 different analyses: accepted for Journal of Automated Reasoning (JAR)
   - Adopted by others for even more analyses

2. Understand differences and key concepts of approaches
   - Interesting insights found

3. Determine potential of extensions and combinations
   - New combinations of features and algorithms immediately available
   - Large potential for future work

4. Provide solid platform for experimental research
   - Top-ranking implementation provided
   - Used for several experimental studies
Performance benchmarking is crucial for research of automatic software verification

Common tools and technologies prone to measurement errors of arbitrary size

Developed new benchmarking framework BenchExec based on modern concepts cgroups and namespaces

Ensures reliable benchmarking

Used by SV-COMP and many research teams

Published in SPIN’15 and STTT
Bounded Model Checking

- [TACAS’99]
- No abstraction
- Unroll loops up to a loop bound $k$
- Check that $P$ holds in the first $k$ iterations:

$$\bigwedge_{i=1}^{k} P(i)$$
- $k$-Induction generalizes the induction principle:
- No abstraction
- Base case: Check that $P$ holds in the first $k$ iterations:
  $\Rightarrow$ Equivalent to BMC with loop bound $k$
- Step case: Check that the safety property is $k$-inductive:

$$
\forall n : \left( \left( \bigwedge_{i=1}^{k} P(n + i - 1) \right) \Rightarrow P(n + k) \right)
$$

- Stronger hypothesis is more likely to succeed
- Add auxiliary invariants [PDMC’11]
- Heavy-weight proof technique
Comparison with SV-COMP’17 Verifiers

- SV-COMP’17 benchmark set:
  - 5,594 C programs with known result
- Time limit 900 s, memory limit 15 GB (per task)
- Comparison of
  - 4 configurations of CPAchecker with Predicate CPA: BMC, $k$-induction, IMPACT, predicate abstraction
  - 16 participants of SV-COMP’17
Comparison with SV-COMP’17 Verifiers: Results

Number of correctly solved tasks:

- Each configuration of Predicate CPA beats other tools with same approach
- Only 3 tools beat Predicate CPA with $k$-induction:
  - **SMACK**: guesses results
  - CPA-BAM-BnB, CPA-Seq: based on Predicate CPA as well
Comparison with SV-COMP’17 Verifiers: Results

Number of correctly solved tasks:

- Each configuration of Predicate CPA beats other tools with same approach
- Only 3 tools beat Predicate CPA with $k$-induction:
  - Smack: guesses results
  - CPA-BAM-BnB, CPA-Seq:
    - based on Predicate CPA as well

Number of wrong results:

- Comparable with other tools
- No wrong proofs (sound)
Comparison with SV-COMP’17 Verifiers

SV-COMP’17

CPA-BAM-BnB
CPA-KInd
CPA-Seq
Cbmc
DepthK
Esbmc
Esbmc-KInd
Smack
Ultimate Automizer
Predicate CPA
(MATHSAT5 QF_UFBVFP)

CPU time (s)
n-th fastest correct result