Predicate-Based Model Checking

Dirk Beyer

LMU Munich, Germany
Based on:

Dirk Beyer, Matthias Dangl, Philipp Wendler:  
**A Unifying View on SMT-Based Software Verification**

https://doi.org/10.1007/s10817-017-9432-6  
preprint: online on CPAchecker website under “Documentation”
SMT-based Software Model Checking

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

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

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

- \textit{k-Induction}
  \(\text{CPAchecker, Esbmc, 2ls,} \ldots\)
Base: Adjustable-Block Encoding

Originally for predicate abstraction:

- Abstraction computation is expensive
- Abstraction is not necessary after every transition
- Track precise path formula between abstraction states
- Reset path formula and compute abstraction formula at abstraction states

- Large-Block Encoding:
  abstraction only at loop heads (hard-coded)

- Adjustable-Block Encoding:
  introduce block operator "blk" to make it configurable
Configurable Program Analysis (CPA):

- Beyer, Henzinger, Théoduloz: [CAV’07]
- One single unifying algorithm for all algorithms based on state-space exploration
- Configurable components: abstract domain, abstract-successor computation, path sensitivity, ...
Using the CPA Framework

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

Diagram:
- Source Code
- Parser & CFA Builder
- CPA Algorithm
- Results
Using the CPA Framework

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

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

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

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

\[\Pi_P \quad \neg \neg_P \quad \text{merge}_P \quad \text{stop}_P \quad \text{prec}_P\]
Predicate CPA

\[ D_\mathcal{P} = (C, \mathcal{E}_\mathcal{P}, [\cdot]_\mathcal{P}) \]

- \( \Pi_\mathcal{P} \)
- \( \sim^{\mathcal{P}} \)
- \( \text{merge}_\mathcal{P} \)
- \( \text{stop}_\mathcal{P} \)
- \( \text{prec}_\mathcal{P} \)
- \( \text{fcover}_\mathcal{P} \)
- \( \text{refine}_\mathcal{P} \)
Predicate CPA: Abstract Domain

- Abstract state: \((\psi, \phi)\)
  - tuple of abstraction formula \(\psi\) and path formula \(\phi\)
    (for ABE)
  - conjunctions represents state space
  - abstraction formula can be a BDD or an SMT formula
  - path formula is always SMT formula and concrete
Predicate CPA: Abstract Domain

- Abstract state: \((\psi, \phi)\)
  - tuple of abstraction formula \(\psi\) and path formula \(\phi\) (for ABE)
  - conjunctions represents state space
  - abstraction formula can be a BDD or an SMT formula
  - path formula is always SMT formula and concrete
- Precision: set of predicates (per program location)
Predicate CPA

\[ D_\mathcal{P} = (C, \mathcal{E}_\mathcal{P}, [\cdot]_\mathcal{P}) \]

- \( \Pi_\mathcal{P} \)
- \( \sim_\mathcal{P} \)
- \( \text{merge}_\mathcal{P} \)
- \( \text{stop}_\mathcal{P} \)
- \( \text{prec}_\mathcal{P} \)
- \( \text{fcover}_\mathcal{P} \)
- \( \text{refine}_\mathcal{P} \)

Abstraction-Formula Representation

- BDD
- SMT-based

Strongest Postcondition

SMT Theory

ABVFP

QF_UFLIRA

Dirk Beyer
LMU Munich, Germany
Predicate CPA: CPA Operators

- Transfer relation:
  - computes strongest post
  - changes only path formula, new abstract state is \((\psi, \varphi')\)
  - purely syntactic, cheap
  - variety of encodings using different SMT theories possible (different approximations for arithmetic and heap operations)
Predicate CPA: CPA Operators

- **Transfer relation:**
  - Computes strongest post
  - Changes only path formula, new abstract state is \((\psi, \varphi')\)
  - Purely syntactic, cheap
  - Variety of encodings using different SMT theories possible (different approximations for arithmetic and heap operations)

- **Merge operator:**
  - Standard for ABE: create disjunctions inside block
Predicate CPA: CPA Operators

- **Transfer relation:**
  - computes strongest post
  - changes only path formula, new abstract state is \((\psi, \varphi')\)
  - purely syntactic, cheap
  - variety of encodings using different SMT theories possible (different approximations for arithmetic and heap operations)

- **Merge operator:**
  - standard for ABE: create disjunctions inside block

- **Stop operator:**
  - standard for ABE: check coverage only at block ends
Predicate CPA: CPA Operators

- **Transfer relation:**
  - computes strongest post
  - changes only path formula, new abstract state is \((\psi, \varphi')\)
  - purely syntactic, cheap
  - variety of encodings using different SMT theories possible
    (different approximations for arithmetic and heap operations)

- **Merge operator:**
  - standard for ABE: create disjunctions inside block

- **Stop operator:**
  - standard for ABE: check coverage only at block ends

- **Precision-adjustment operator:**
  - only active at block ends (as determined by \(blk\))
  - computes abstraction of current abstract state
  - new abstract state is \((\psi', \text{true})\)
Predicate CPA

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

Block Diagram:
- Abstraction-Formula Representation
  - BDD
  - SMT-based
  - ABVFP
  - QF_UFLIRA
- Strongest Postcondition
  - SMT Theory
  - ABVFP
- Predicate Abstraction
  - blk
  - blk^{SBE}
  - Cartesian
  - Boolean
  - blk^i
  - blk^{lf}
  - blk^{never}

Strongest Postcondition

SMT Theory

Predicate CPA $P$

Predicates:
- $P$
- $\Pi_P$
- $\sim_P$
Predicate CPA: Refinement

Four steps:

1. Reconstruct ARG path to abstract error state
2. Check feasibility of path
3. Discover abstract facts, e.g.,
   - interpolants
   - weakest precondition
   - heuristics
4. Refine abstract model
   - add predicates to precision, cut ARG
     or
   - conjoin interpolants to abstract states, recheck coverage relation
Predicate CPA

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

\[ \Pi_P \quad \sim_P \quad \text{merge}_P \quad \text{stop}_P \quad \text{prec}_P \quad \text{fcover}_P \quad \text{refine}_P \]

**Abstraction-Formula Representation**

**Strongest Postcondition**

**Predicate Abstraction**

**Abstract Facts**

**Refinement Strategy**

- **BDD**
- **SMT Theory**
- **SMT-based**
- **ABVFP**
- **QF_UFLIRA**
- **blk**
- **blk^{SBE}**
- **blk^i**
- **blk^{lf}**
- **blk^{never}**
- **Cartesian**
- **Boolean**
- **fcover^{id}**
- **fcover^{IMPACT}**
- **Interpolants**
- **Path Invariants**
- **Predicate**
- **Unsat Cores**
- **Weakest Preconditions**
- **Heuristic Predicates**

Dirk Beyer
LMU Munich, Germany
13 / 32
Predicate Abstraction

- 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
  - Abstraction formula as BDD
Expressing Predicate Abstraction

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

Use CEGAR Algorithm:

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

$D_p = (C, E_p, [\cdot]_p)$

- Predicate CPA $P$
  - Abstract Formula Representation
    - BDD
    - SMT Theory
      - SMT-based
      - ABVFP
        - QF_UFLIRA
        - $blk_{\text{lf}}$
        - $blk_{\text{never}}$
      - Cartesian
      - Boolean
      - $blk^{SBE}$
    - $blk^I$
  - Strongest Postcondition
  - Predicate Abstraction
    - $fcover^I$
    - $fcover^\text{id}$
  - Refinement Strategy
    - Interpolants
    - Predicate
    - Path Invariants
    - Unsat Cores
    - Weakest Preconditions
    - Heuristic Predicates

Dirk Beyer  
LMU Munich, Germany
Example Program

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

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

**Abstraction-Formula Representation**

**Strongest Postcondition**

\( P = \Pi_P \circ \sim_P \circ \text{merge}_P \circ \text{stop}_P \circ \text{prec}_P \circ \text{fcover}_P \circ \text{refine}_P \)

**Predicate CPA**

**SMT Theory**

- BDD
- ABVFP
- QF_UFLIRA

**SMT-based**

**Predicates**

- Cartesian
- Boolean
- blk
- \( \text{fcover}^{id} \)
- \( \text{fcover}^{\text{impact}} \)

**Abstract Facts**

**Refinement Strategy**

- Interpolants
- Path Invariants
- Unsat Cores
- Weakest Preconditions
- Heuristic Predicates

\( \text{Dirk Beyer \ LMU Munich, Germany 18 / 32} \)
Predicate Abstraction: Example

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

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

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

with $\pi(l_4) = \{x = y\}$ and $\pi(l_8) = \{false\}$
Predicate Abstraction: Example

with \( \text{blk}^l \), \( \pi(l_4) = \{x = y\} \) and \( \pi(l_8) = \{\text{false}\} \)
Predicate Abstraction: Example

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

with blk\(^l\), \(\pi(l_4) = \{x = y\}\) and \(\pi(l_8) = \{false\}\)
Predicate Abstraction: Example

with blk\(^l\), \(\pi(l_4) = \{x = y\}\) and \(\pi(l_8) = \{false\}\)
Predicate Abstraction: Example

with blk, \( \pi(l_4) = \{ x = y \} \) and \( \pi(l_8) = \{ \text{false} \} \)
Predicate Abstraction: Example

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

$\begin{align*}
\text{unsigned int } x &= 0; \\
\text{unsigned int } y &= 0; \\
[l_2] &\xrightarrow{[x < 2]} l_3 \\
[l_3] &\xrightarrow{\text{start}} l_4 \\
[l_4] &\xrightarrow{[! (x \neq y)]} l_5 \\
l_5 &\xrightarrow{\text{unsigned int } y = 0;} l_6 \\
l_6 &\xrightarrow{y++;} l_7 \\
l_7 &\xrightarrow{[x \neq y]} l_8 \\
l_8 &\xrightarrow{[! (x < 2)]} l_9 \\
l_9 &\xrightarrow{\text{ERROR: return 1;}} l_{10} \\
l_{10} &\xrightarrow{\text{return 0;}} l_{11} \\
l_{11} &\xrightarrow{\text{return 0;}} l_{12}
\end{align*}$

\[\begin{align*}
e_0 &: (l_2, (\text{true, true})) \\
e_1 &: (l_3, (\text{true, } x_0 = 0)) \\
e_2 &: (l_4, (x = y, \text{true})) \\
e_3 &: (l_11, (x = y, \neg (x_0 < 2))) \\
e_4 &: (l_12, (x = y, \neg (x_0 < 2))) \\
e_5 &: (l_5, (x = y, x_0 < 2)) \\
e_6 &: (l_6, (x = y, x_0 < 2 \land x_1 = x_0 + 1)) \\
e_7 &: (l_7, (x = y, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1)) \\
e_8 &: (l_4, (x = y, \text{true})) \\
e_9 &: (l_8, (\text{false, true}))
\end{align*}\]
Predicate Abstraction: Example

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

"Lazy Abstraction with Interpolants" [CAV’06]
Abstraction is derived dynamically/lazily
Solution to avoiding expensive abstraction computations
Compute fixed point over three operations
  ▶ Expand
  ▶ Refine
  ▶ Cover
Abstraction formula as SMT formula
Optimization: forced covering
Expressing **IMPACT**

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

**Furthermore:**

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

\[ D_p = (C, E_p, \cdot) \]

\( \Pi_p \)

\( \sim_p \)

merge\( P \)

stop\( P \)

prec\( P \)

fcover\( P \)

refine\( P \)

Abstraction-Formula Representation

Strongest Postcondition

Predicate CPA \( P \)

SMT Theory

\( \Pi \)

\( \sim \)

merge

stop

prec

fcover

refine

Abstract Facts

Refinement Strategy

SMT-based

ABVFP

QF_UFLIRA

BDD

SBE

\( blk^{SBE} \)

\( blk^{L} \)

\( blk^{lf} \)

\( blk^{never} \)

Cartesian

Boolean

fcover\( id \)

fcover\( \text{IMPACT} \)

Interpolants

Impact

Path Invariants

Unsat Cores

Weakest Preconditions

Heuristic Predicates

Dirk Beyer

LMU Munich, Germany

22 / 32
Predicate CPA

\[ D_P = \langle C, \mathcal{E}_P, [\cdot]_P \rangle \]

- **Abstraction-Formula Representation**
- **Strongest Postcondition**
- **Predicate CPA \( P \)**
  - **Predicate Abstraction**
  - **Refinement Strategy**
  - **Abstract Facts**
  - **Interpolants**
  - **Path Invariants**
  - **Unsat Cores**
  - **Weakest Preconditions**
  - **Heuristic Predicates**

**Predicates**
- \( P \)
- \( \Pi_P \)
- \( \sim_P \)
- \( \text{merge}_P \)
- \( \text{stop}_P \)
- \( \text{prec}_P \)
- \( \text{fcover}_P \)
- \( \text{refine}_P \)

**Abstraction-Formula Representation**
- SMT-based
- BDD
- ABVFP
- QF_UFLIRA

**SMT Theory**
- \( \text{blk}^{SBE} \)
- \( \text{blk}^l \)
- \( \text{blk}^{lf} \)
- \( \text{blk}^{never} \)

**SMT-based**
- Cartesian
- Boolean

**Refinement Strategy**
- \( \text{fcover}^id \)
- \( \text{fcover}^{\text{IMPACT}} \)

**Impact**
- \( \text{blk}^l \)
Impact: Example

with blk_l

\[
\begin{align*}
\text{start} & \rightarrow l_2 \\
& \downarrow \text{unsigned int } x = 0; \\
& \downarrow l_3 \downarrow \text{unsigned int } y = 0; \\
& \downarrow l_4 \downarrow [x < 2] \\
& \downarrow l_5 \downarrow x++; \\
& \downarrow l_6 \downarrow y++; \\
& \downarrow l_7 \downarrow [x != y] \\
& \downarrow l_8 \downarrow [x != y] \\
& \downarrow l_{11} \downarrow \text{ERROR: return 1;} \\
& \downarrow l_{12} \downarrow \text{return 0;
\end{align*}
\]

\[
e_0: (l_2, (\text{true, true})) \\
e_1: (l_3, (\text{true, } x_0 = 0)) \\
e_2: (l_4, (\text{true, } x_0 = 0 \land y_0 = 0))
\]
**Impact:** Example

with blk

```c
unsigned int x = 0;
unsigned int y = 0;

[x < 2]
[!(x < 2)]

y++;
![x != y]

return 0;
```

**ERROR:** return 1;
**Impact: Example with blk**

```c
unsigned int x = 0;
unsigned int y = 0;

if (x < 2) {
    x++;
    y++;
    if (x != y) {
        ERROR: return 1;
    }
}
return 0;
```
**Impact: Example**

with \( \text{blk}^l \)

```
unsigned int x = 0;
unsigned int y = 0;
[x < 2]
[!(x < 2)]

[(x != y)]
[x != y]

[l2]
[l3]
[l4]
[x < 2]
[l5]
x++;
[l6]
y++;
[l7]
[l8]
[l11]
ERROR: return 1;
[l12]
```

```
e0: (l2, (true, true))
e1: (l3, (true, x0 = 0))
e2: (l4, (true, true))
e3: (l11, (true, !(x < 2)))
e4: (l12, (true, !(x < 2)))
e5: (l5, (true, x0 < 2))
e6: (l6, (true, x0 < 2 ∧ x1 = x0 + 1))
e7: (l7, (true, x0 < 2 ∧ x1 = x0 + 1 ∧ y1 = y0 + 1))
```

 Covered by Dirk Beyer LMU Munich, Germany 24 / 32
Impact: Example with blk\textsuperscript{l}

unsigned int x = 0;
unsigned int y = 0;

[x < 2]
[x != y]

[l2] start
[l3] unsigned int x = 0;
[l4] unsigned int y = 0;
[l5] [x < 2]
[l6] [!(x != y)]
[l7] [x != y]
[l8] return 0;
[l9] ERROR: return 1;
[l10] return 0;
[l11]
[l12]

\begin{itemize}
\item \textbf{e}0: \((l_2, \text{true, true}))
\item \textbf{e}1: \((l_3, \text{true, } x_0 = 0))
\item \textbf{e}2: \((l_4, \text{true, true}))
\item \textbf{e}3: \((l_{11}, \text{true, } \neg(x_0 < 2)))
\item \textbf{e}4: \((l_{12}, \text{true, } \neg(x_0 < 2)))
\item \textbf{e}5: \((l_5, \text{true, } x_0 < 2))
\item \textbf{e}6: \((l_6, \text{true, } x_0 < 2 \land x_1 = x_0 + 1))
\item \textbf{e}7: \((l_7, \text{true, } x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1))
\item \textbf{e}8: \((l_8, \text{true, } x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)))
\end{itemize}
**Impact:** Example

with blk$^l$

```
unsigned int x = 0;
unsigned int y = 0;

[x < 2]
x++;

y++;

[x != y]

return 0;
```

```
e_0: (l_2, (true, true))
e_1: (l_3, (true, x_0 = 0))
e_2: (l_4, (true, true))
e_3: (l_11, (true, ! (x_0 < 2)))
e_4: (l_12, (true, ! (x_0 < 2)))
e_5: (l_5, (true, x_0 < 2))
e_6: (l_6, (true, x_0 < 2 ∧ x_1 = x_0 + 1))
e_7: (l_7, (true, x_0 < 2 ∧ x_1 = x_0 + 1 ∧ y_1 = y_0 + 1))
e_8: (l_8, (true, true))
```
**Impact:** Example

with $\text{blk}^l$

```c
unsigned int x = 0;
unsigned int y = 0;

[x < 2] [!(x != y)]
```

```
    l_2
    \downarrow
    \text{unsigned int } x = 0;

    l_3
    \downarrow
    \text{unsigned int } y = 0;

    l_4
    \downarrow
    [x < 2]

    l_5
    \downarrow
    x++;

    l_6
    \downarrow
    y++;

    l_7
    \downarrow
    [x != y]

    l_8
    \downarrow
    return 0;

    l_12
```

```
    e_0: (l_2, (true, true))
    \downarrow
    e_1: (l_3, (true, x_0 = 0))
    \downarrow
    e_2: (l_4, (x = y, true))
    \downarrow
    e_3: (l_{11}, (true, \neg(x_0 < 2)))
    \downarrow
    e_4: (l_{12}, (true, \neg(x_0 < 2)))
    \downarrow
    e_5: (l_5, (true, x_0 < 2))
    \downarrow
    e_6: (l_6, (true, x_0 < 2 \land x_1 = x_0 + 1))
    \downarrow
    e_7: (l_7, (true, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1))
    \downarrow
    e_8: (l_8, (false, true))
```
**Impact:** Example

with blk

```
unsigned int x = 0;
unsigned int y = 0;

[l2: (true, true)]
[l3: (true, x0 = 0)]
[l4: (true, x < 2)]
[l5: (true, x++)]
[l6: (true, y++)]
[l7: (true, [x != y])]  
[l8: (true, [x < 2])]
[l9: (false, true)]
[l10: (true, [x != y])]
[l11: (true, [x < 2])]
[l12: (false, true)]
```

ERROR: return 1;

return 0;

DIRK BEYER
LMU MUNICH, GERMANY
Impact: Example

with blk

\begin{align*}
\text{unsigned int } x &= 0; \\
\text{unsigned int } y &= 0; \\
[x < 2] & \rightarrow l_4 \\
[x = y] & \rightarrow l_5 \\
x++; & \rightarrow l_6 \\
x & \rightarrow l_7 \\
x ! = y & \rightarrow l_8 \\
\text{return } 0; & \rightarrow l_{11} \\
\text{ERROR: return } 1; & \rightarrow l_{12}
\end{align*}
**Impact:** Example

with blk

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

---

[Diagram of the example code flow with labeled blocks and conditions.

- \( e_0: (l_2, (true, true)) \)
- \( e_1: (l_3, (true, x_0 = 0)) \)
- \( e_2: (l_4, (x = y, true)) \)
- \( e_3: (l_{11}, (true, \neg(x_0 < 2))) \)
- \( e_4: (l_{12}, (true, \neg(x_0 < 2))) \)
- \( e_5: (l_5, (true, x_0 < 2)) \)
- \( e_6: (l_6, (true, x_0 < 2 \land x_1 = x_0 + 1)) \)
- \( e_7: (l_7, (true, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1)) \)
- \( e_8: (l_8, (false, true)) \)
- \( e_9: (l_4, (true, true)) \)
- \( e_{10}: (l_5, (true, x_1 < 2)) \)
- \( e_{11}: (l_6, (true, x_1 < 2 \land x_2 = x_1 + 1)) \)
- \( e_{12}: (l_7, (true, x_1 < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1)) \)
- \( e_{13}: (l_8, (true, x_1 < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1 \land \neg(x_2 = y_2))) \)
Impact: Example with blk

\[ \text{unsigned int } x = 0; \]
\[ \text{unsigned int } y = 0; \]
\[ \text{\[x < 2\]} \]
\[ \text{\[!x = y\]} \]
\[ \text{\[!x < 2\]} \]

start → \( l_2 \)
\( l_2 \) → \( l_3 \)
\( l_3 \) → \( l_4 \)
\( l_4 \) → \( [x < 2] \)
\( l_5 \) → \( \text{\[x != y\]} \)
\( l_6 \) → \( \text{\[!x < 2\]} \)
\( l_7 \) → \( l_8 \)
\( l_11 \) → \( \text{ERROR: return 1;} \)
\( l_12 \) → end

\[ e_0: \text{(l_2, (true, true))} \]
\[ e_1: \text{(l_3, (true, x_0 = 0))} \]
\[ e_2: \text{(l_4, (x = y, true))} \]
\[ e_3: \text{(l_11, (true, \neg(x_0 < 2)))} \]
\[ e_4: \text{(l_12, (true, \neg(x_0 < 2)))} \]
\[ e_5: \text{(l_5, (true, x_0 < 2))} \]
\[ e_6: \text{(l_6, (true, x_0 < 2 \land x_1 = x_0 + 1))} \]
\[ e_7: \text{(l_7, (true, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1))} \]
\[ e_9: \text{(l_4, (true, true))} \]
\[ e_{10}: \text{(l_5, (true, x_1 < 2))} \]
\[ e_{11}: \text{(l_6, (true, x_1 < 2 \land x_2 = x_1 + 1))} \]
\[ e_{12}: \text{(l_7, (true, x_1 < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1))} \]
\[ e_{13}: \text{(l_8, (true, true))} \]
**Impact:** Example

with \( \text{blk}^I \)

\[
\begin{align*}
\text{unsigned int } x &= 0; \\
\text{unsigned int } y &= 0; \\
\text{if } (x < 2) \quad \text{then}\quad &x++; \\
\text{if } (x != y) \quad \text{then}\quad &y++; \\
\text{else}\quad &\text{ERROR: return 1;}
\end{align*}
\]
**Impact: Example**

with blk$^l$

\[
\begin{align*}
\text{unsigned int } & \quad x = 0; \\
\text{unsigned int } & \quad y = 0; \\
[l_2] & \quad \text{[x < 2]} \\
[l_3] & \quad \text{[!x != y]} \\
[l_4] & \quad x++; \\
[l_5] & \quad y++; \\
[l_6] & \quad [x != y] \\
[l_7] & \quad \text{ERROR: return 1;}
\end{align*}
\]

covered by

\[
\begin{align*}
e_0: \quad & (l_2, (true, true)) \\
e_1: \quad & (l_3, (true, x_0 = 0)) \\
e_2: \quad & (l_4, (x = y, true)) \\
e_3: \quad & (l_{11}, (true, \neg(x_0 < 2))) \\
e_4: \quad & (l_{12}, (true, \neg(x_0 < 2))) \\
e_5: \quad & (l_5, (true, x_0 < 2)) \\
e_6: \quad & (l_6, (true, x_0 < 2 \land x_1 = x_0 + 1)) \\
e_7: \quad & (l_7, (true, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1)) \\
e_8: \quad & (l_8, (false, true)) \\
e_9: \quad & (l_4, (x = y, true)) \\
e_{10}: \quad & (l_5, (true, x_1 < 2)) \\
e_{11}: \quad & (l_6, (true, x_1 < 2 \land x_2 = x_1 + 1)) \\
e_{12}: \quad & (l_7, (true, x_1 < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1)) \\
e_{13}: \quad & (l_8, (false, true))
\end{align*}
\]
Bounded Model Checking:

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

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

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

Furthermore:

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

Use Algorithm for iterative BMC:

1. \(k = 1\)
2. \textbf{while} !\text{finished} \textbf{do}
3. run CPA Algorithm
4. check feasibility of each abstract error state
5. \(k++\)
Predicate CPA

\[ D_P = (C, E_P, \cdot) \]

\[ \Pi_P \]

\[ \sim_P \]

merge\_P

stop\_P

prec\_P

fcover\_P

refine\_P

Abstraction-Formula Representation

Strongest Postcondition

Predicate Abstraction

Abstract Facts

Refinement Strategy

SMT Theory

BDD

SMT-based

ABVFP

QF_UFLIRA

blk\_SBE

blk\_id

blk\_IMPACT

blk\_never

blk\_l

Cartesian

Boolean

Interpolants

Predicate

Path Invariants

Unsat Cores

Weakest Preconditions

Heuristic Predicates
Bounded Model Checking: Example with $k = 1$

```
unsigned int x = 0;
unsigned int y = 0;

[x < 2]}
\[!(x < 2)\]
\[\!(x \neq y)\]
\[\![x \neq y]\]
```

```
e_0: (l_2, (true, true), \{l_4 \mapsto -1\})
e_1: (l_3, (true, x_0 = 0), \{l_4 \mapsto -1\})
e_2: (l_4, (true, x_0 = 0 \land y_0 = 0), \{l_4 \mapsto 0}\})
e_3: (l_{11}, (true, x_0 = 0 \land y_0 = 0 \land \neg(x_0 < 2)), \{l_4 \mapsto 0\})
e_4: (l_{12}, (true, x_0 = 0 \land y_0 = 0 \land \neg(x_0 < 2)), \{l_4 \mapsto 0\})
e_5: (l_5, (true, x_0 = 0 \land y_0 = 0 \land x_0 < 2), \{l_4 \mapsto 0\})
e_6: (l_6, (true, x_0 = 0 \land y_0 = 0 \land x_0 < 2 \land x_1 = x_0 + 1), \{l_4 \mapsto 0\})
e_7: (l_7, (true, x_0 = 0 \land y_0 = 0 \land x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1), \{l_4 \mapsto 0\})
e_8: (l_8, (true, x_0 = 0 \land y_0 = 0 \land x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)), \{l_4 \mapsto 0\})
e_9: (l_{12}, (true, x_0 = 0 \land y_0 = 0 \land x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)), \{l_4 \mapsto 0\})
e_{10}: (l_4, (true, x_0 = 0 \land y_0 = 0 \land x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(\neg(x_1 = y_1))), \{l_4 \mapsto 1\})
```
1-Induction

1-Induction:

- Base case: Check that the safety property holds in the first loop iteration:
  \[ P(1) \]
  \[ \rightarrow \text{Equivalent to BMC with loop bound 1} \]

- Step case: Check that the safety property is 1-inductive:
  \[ \forall n : (P(n) \Rightarrow P(n + 1)) \]
$k$-Induction

- $k$-Induction generalizes the induction principle:
  - No abstraction
  - Base case: Check that $P$ holds in the first $k$ iterations:
    $\rightarrow$ Equivalent to BMC with loop bound $k$
  - Step case: Check that the safety property is $k$-inductive:
    $$\forall n : \left( \left( \bigwedge_{i=1}^{k} P(n + i - 1) \right) \Rightarrow P(n + k) \right)$$
  - Stronger hypothesis is more likely to succeed
  - Add auxiliary invariants
  - Kahsai, Tinelli: [PDMC’11]
**k-Induction with Auxiliary Invariants**

**Induction:**
1. $k = 1$
2. **while** !finished **do**
3. BMC($k$)
4. Induction($k$, invariants)
5. $k++$

**Invariant generation:**
1. prec = <weak>
2. invariants = $\emptyset$
3. **while** !finished **do**
4. invariants = GenInv(prec)
5. prec = RefinePrec(prec)
**k-Induction: Example**

- $e_0$: $(l_4, (\text{true}, \text{true}), \{l_4 \mapsto 0\})$
- $e_1$: $(l_{11}, (\text{true}, \neg(x_0 < 2)), \{l_4 \mapsto 0\})$
- $e_2$: $(l_{12}, (\text{true}, \neg(x_0 < 2)), \{l_4 \mapsto 0\})$
- $e_3$: $(l_5, (\text{true}, x_0 < 2), \{l_4 \mapsto 0\})$
- $e_4$: $(l_6, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1), \{l_4 \mapsto 0\})$
- $e_5$: $(l_7, (\text{true}, \land x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1), \{l_4 \mapsto 0\})$
- $e_6$: $(l_8, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)), \{l_4 \mapsto 0\})$
- $e_7$: $(l_{12}, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)), \{l_4 \mapsto 0\})$
- $e_8$: $(l_4, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land \neg(x_1 < 2)), \{l_4 \mapsto 1\})$
- $e_9$: $(l_{11}, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land \neg(x_1 < 2)), \{l_4 \mapsto 1\})$
- $e_{10}$: $(l_{12}, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land \neg(x_1 < 2)), \{l_4 \mapsto 1\})$
- $e_{11}$: $(l_5, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land x_1 < 2 \land x_2 = x_1 + 1), \{l_4 \mapsto 1\})$
- $e_{12}$: $(l_6, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land x_1 < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1), \{l_4 \mapsto 1\})$
- $e_{13}$: $(l_7, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land x_1 < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1), \{l_4 \mapsto 1\})$
- $e_{14}$: $(l_8, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land x < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1 \land \neg(x_2 = y_2)), \{l_4 \mapsto 1\})$
- $e_{15}$: $(l_{12}, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land x < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1 \land \neg(x_2 = y_2)), \{l_4 \mapsto 1\})$
- $e_{16}$: $(l_4, (\text{true}, x_0 < 2 \land x_1 = x_0 + 1 \land y_1 = y_0 + 1 \land \neg(x_1 = y_1)) \land x < 2 \land x_2 = x_1 + 1 \land y_2 = y_1 + 1 \land \neg(x_2 = y_2)), \{l_4 \mapsto 2\})$