Augmenting Interpolation-Based Model Checking with Auxiliary Invariants

Dirk Beyer, Po-Chun Chien, and Nian-Ze Lee

SPIN 2024 @ Luxembourg
LMU Munich, Germany
Cooperative Verification via Invariant Injection

Main analysis

- BMC [9, 13]
- $k$-Induction [6, 11]
- Predicate abstraction [15]
- ...

Auxiliary invariant generator

- Abstract interpretation
- Data-flow analysis
- ...

\textbf{effective but expensive}

\textbf{efficient but imprecise}
Cooperative Verification via Invariant Injection

Main analysis

- BMC [9, 13]
- k-Induction [6, 11]
- Predicate abstraction [15]
- IMC ??

Auxiliary invariant generator

- Abstract interpretation
- Data-flow analysis
- ...

invariants

effective but expensive

efficient but imprecise

SoSy-Lab, LMU Munich

Augmenting IMC with Auxiliary Invariants
Highlights

- Novelty: 1st to combine IMC with data-flow analysis
Highlights

- **Novelty:** 1st to combine IMC with data-flow analysis
- In our evaluation, augmented IMC
  - was faster and more effective than plain IMC
  - tackled tasks unsolvable by state-of-the-art verifiers
Example Program

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```
Example Program

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

- Reachable states \((x,j)\) at loop head:
  - \((0,0)\)
Example Program

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

- Reachable states \((x,j)\) at loop head:
  - \((0,0)\)
  - \((2,1)\)
Example Program

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

- Reachable states \((x,j)\) at loop head:
  - \((0,0)\)
  - \((2,1)\)
  - \((4,0)\)
Example Program

```c
1 int main(void) {
2    unsigned x = 0;
3    unsigned j = 0;
4    while (nondet()) {
5        x += 2;
6        if (j == 3)
7            x += 1;
8        j += 1;
9        if (j == 2)
10           j = 0;
11    }
12    if (x % 2) {
13       ERROR: return 1;
14    }
15    return 0;
16 }
```

- Reachable states \((x,j)\) at loop head:
  - \((0,0)\)
  - \((2,1)\)
  - \((4,0)\)
  - \((6,1)\)
  - \(\ldots\)
Example Program

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

- Reachable states \((x, j)\) at loop head:
  - \((0,0)\)
  - \((2,1)\)
  - \((4,0)\)
  - \((6,1)\)
  - \(\ldots\)

- **ERROR** unreachable
Example: Plain IMC

```
1 int main(void) {
2   unsigned x = 0;
3   unsigned j = 0;
4   while (nondet()) {
5     x += 2;
6     if (j == 3)
7        x += 1;
8     j += 1;
9     if (j == 2)
10        j = 0;
11   }
12   if (x % 2) {
13      ERROR: return 1;
14   }
15   return 0;
16 }
```

- Candidate fixed-point *fp* at loop head: \( x \% 2 = 0 \)
Example: Plain IMC

```c
int main(void) {
  unsigned x = 0;
  unsigned j = 0;
  while (nondet()) {
    x += 2;
    if (j == 3) x += 1;
    j += 1;
    if (j == 2) j = 0;
  }
  if (x % 2) {
    ERROR: return 1;
  }
  return 0;
}
```

- Candidate fixed-point \(fp\) at loop head: \(x \% 2 = 0\)
- CEX: \(x = 0 \land j = 3\)
Example: Plain IMC

```c
int main(void) {
  unsigned x = 0;
  unsigned j = 0;
  while (nondet()) {
    x += 2;
    if (j == 3)
      x += 1;
    j += 1;
    if (j == 2)
      j = 0;
  }
  if (x % 2) {
    ERROR: return 1;
  }
  return 0;
}
```

- Candidate fixed-point \( fp \) at loop head: \( x \% 2 = 0 \)
- CEX: \( x = 0 \land j = 3 \)  
  \[ \downarrow 1 \text{ loop iteration} \]  
  \[ x = 3 \land j = 4 \]
Example: Plain IMC

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

- Candidate fixed-point \(fp\) at loop head: \(x \mod 2 = 0\)
- CEX: \(x = 0 \land j = 3\)

\[\downarrow 1\text{ loop iteration}\]
\[x = 3 \land j = 4\]
Example: Plain IMC

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    } else {
        return 0;
    }
}
```

- Candidate fixed-point \( fp \) at loop head: \( x \% 2 = 0 \)
- CEX: \( x = 0 \land j = 3 \)
  \( \downarrow 1 \text{ loop iteration} \)
  \( x = 3 \land j = 4 \)
- \( fp \) is non-inductive
  \( \rightarrow \) needs refinement!
Example: IMC with Auxiliary Invariants

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;

    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }

    if (x % 2) {
        ERROR: return 1;
    }

    return 0;
}
```

- Candidate fixed-points at loop head produced by IMC: \( x \% 2 = 0 \)
Example: IMC with Auxiliary Invariants

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3) x += 1;
        j += 1;
        if (j == 2) j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

- Candidate fixed-points at loop head produced by IMC: \( x \% 2 = 0 \)
  Data-flow: \( 0 \leq j \leq 1 \)
Example: IMC with Auxiliary Invariants

```c
int main(void) {
  unsigned x = 0;
  unsigned j = 0;
  while (nondet()) {
    x += 2;
    if (j == 3) x += 1;
    j += 1;
    if (j == 2) j = 0;
  }
  if (x % 2) {
    ERROR: return 1;
  }
  return 0;
}
```

- Candidate fixed-points at loop head produced by IMC: \( x \% 2 = 0 \)
  Data-flow: \( 0 \leq j \leq 1 \)

- \( x \% 2 = 0 \land 0 \leq j \leq 1 \) is an inductive (and safe) invariant!
Agenda

1. Background: IMC and data-flow analysis
2. Augmenting IMC with auxiliary invariants
3. Experimental evaluation
1. Background: IMC and data-flow analysis

2. Augmenting IMC with auxiliary invariants

3. Experimental evaluation
Interpolation and SAT-Based Model Checking

- K. L. McMillan, CAV 2003 [16]
- Interpolation-based model checking (IMC)
  - Originally designed for finite-state transition systems
  - Compute fixed points by interpolating unsatisfiable BMC queries
Interpolation and SAT-Based Model Checking

- K. L. McMillan, CAV 2003 [16]
- Interpolation-based model checking (IMC)
  - Originally designed for finite-state transition systems
  - Compute fixed points by interpolating unsatisfiable BMC queries
- State of the art for hardware verification
- Recently adopted for verifying software programs

---

1Interpolation and SAT-Based Model Checking Revisited: Adoption to Software Verification
(to appear in JAR [7])
Craig Interpolation

- If $A(X, Y) \land B(Y, Z)$ is UNSAT: interpolant $\tau(Y)$
  - $A(X, Y) \Rightarrow \tau(Y)$ is valid
  - $\tau(Y) \land B(Y, Z)$ is UNSAT
BMC Stage of IMC

- State-transition system: $\text{Init}(s), T(s, s')$
- Safety property: $P(s)$
BMC Stage of IMC

- State-transition system: $Init(s), T(s, s')$
- Safety property: $P(s)$
- BMC query with unrolling bound $k$:

$Init(s_0) T(s_0, s_1) T(s_1, s_2) \ldots T(s_{k-1}, s_k)(\neg P(s_1) \lor \ldots \lor \neg P(s_k))$

- if SAT, counterexample found
- if UNSAT, enter interpolation stage
BMC Stage of IMC

- State-transition system: $Init(s), T(s, s')$
- Safety property: $P(s)$
- BMC query with unrolling bound $k$:
  
  $Init(s_0) T(s_0, s_1) T(s_1, s_2) ... T(s_{k-1}, s_k)(\neg P(s_1) \lor ... \lor \neg P(s_k))$

  - if SAT, counterexample found
  - if UNSAT, enter interpolation stage
Interpolation Stage of IMC

- Construct a fixed point via interpolation
Interpolation Stage of IMC

- Construct a fixed point via interpolation
- $\text{Init}(s_0) \ T(s_0, s_1) \ T(s_1, s_2) \ldots \ T(s_{k-1}, s_k)(\neg P(s_1) \lor \ldots \lor \neg P(s_k))$
Interpolation Stage of IMC

- Construct a fixed point via interpolation

\[ \text{Init}(s_0) \underbrace{T(s_0, s_1) T(s_1, s_2) \ldots T(s_{k-1}, s_k)}_{A_0(s_0, s_1)} (\neg P(s_1) \lor \ldots \lor \neg P(s_k)) \underbrace{T(s_1, s_2) \ldots T(s_{k-1}, s_k)}_{B(s_1, s_2, \ldots, s_k)} \]

- Interpolant \( \tau_1(s_1) \): 1-step safe overapproximation
Interpolation Stage of IMC

- Construct a fixed point via interpolation
- \(\text{Init}(s_0) \ T(s_0, s_1) \ T(s_1, s_2) \ ... \ T(s_{k-1}, s_k)(\neg P(s_1) \lor \ldots \lor \neg P(s_k))\)
  - \(A_0(s_0, s_1)\)
  - \(B(s_1, s_2, \ldots, s_k)\)
  - Interpolant \(\tau_1(s_1)\): 1-step safe overapproximation
- \(\tau_1(s_0) \ T(s_0, s_1) \ T(s_1, s_2) \ ... \ T(s_{k-1}, s_k)(\neg P(s_1) \lor \ldots \lor \neg P(s_k))\)
  - \(A_1(s_0, s_1)\)
  - \(B(s_1, s_2, \ldots, s_k)\)
  - Interpolant \(\tau_2(s_1)\): 2-step safe overapproximation
- Derive \(n\)-step overapproximation \(\tau_n\) iteratively
Fixed Point

- Repeat until $Init \lor \bigvee \tau_i$ becomes a fixed point
Fixed Point

- Repeat until $Init \lor \bigvee \tau_i$ becomes a fixed point

$Init \lor \tau_1 \lor \tau_2 \cdots \lor \tau_{n-1} \Rightarrow Init \lor \neg \tau_1 \lor \neg \tau_2 \cdots \lor \neg \tau_{n-1}$
Fixed Point

- Repeat until $\text{Init} \lor \bigvee \tau_i$ becomes a fixed point

$\tau_n \Rightarrow \text{Init} \lor \bigvee_{i=1}^{n-1} \tau_i$ holds

$\text{Init} \lor \tau_1 \lor \tau_2 \cdots \lor \tau_{n-1}$

$\neg P$
Potentially-Spurious Counterexample

- Increment unrolling bound $k$ if a query becomes satisfiable

\[ \neg P \]

\[ \text{Init} \lor \tau_1 \lor \tau_2 \ldots \lor \tau_n \]
Potentially-Spurious Counterexample

- Increment unrolling bound $k$ if a query becomes satisfiable

\[ \text{Init} \lor \tau_1 \lor \tau_2 \cdots \lor \tau_n \]

\[ \neg P \]
Potentially-Spurious Counterexample

- Increment unrolling bound $k$ if a query becomes satisfiable

$cex$ might be spurious because $\tau_n$ is an overapproximation
Termination Conditions of IMC

- IMC terminates when
  - A fixed point is reached: return a safety proof
  - $\text{Init}(s_0) \land \bigwedge_{i=1}^{k} T(s_{i-1}, s_i) \land (\bigvee_{i=1}^{k} \neg P(s_i))$ is SAT for some $k$: return a counterexample
- This work improves IMC’s capability of constructing proofs
IMC for Software Verification

- Extract formulas by *large-block encoding* [5, 7]

\[
\begin{align*}
(l_0, \text{true})
\downarrow \\
(l_H, \varphi_0|_{i=0})
\downarrow \\
(l_H, (C \land \varphi_L)|_{i=1})
\downarrow \\
(l_H, (C \land \varphi_L)|_{i=k})
\downarrow \\
(l_E, ((\neg C \land \varphi_E) \lor (C \land \varphi'_E))|_{i=k})
\end{align*}
\]

(For multi-loop programs: standard transformation to single loop [1, 12])
IMC for Software Verification

- Extract formulas by *large-block encoding* [5, 7]

(For multi-loop programs: standard transformation to single loop [1, 12])
- Extract formulas by *large-block encoding* [5, 7]

\[
\begin{align*}
(l_0, true) \quad & \downarrow \\
(l_H, \varphi_0 | i=0) & : \text{Init}(s_0) \\
(l_H, (C \land \varphi_L) | i=1) & : T(s_0, s_1) \\
(l_H, (C \land \varphi_L) | i=k) & : T(s_{k-1}, s_k) \\
(l_E, ((\neg C \land \varphi_E) \lor (C \land \varphi'_E)) | i=k) & \downarrow \\
\end{align*}
\]

(For multi-loop programs: standard transformation to single loop [1, 12])
- Extract formulas by *large-block encoding* [5, 7]

(FOR multi-loop programs: standard transformation to single loop [1, 12])
Auxiliary Invariant Generator

- Continuously-refining data-flow analysis (DF) based on intervals [3]
- Produce **inductive** invariants
- Invariants are expressions over intervals
  - e.g. \((0 \leq j \leq 1) \land (x < 5 \lor x > 7)\)
- Invariant injection denoted as \(\in\) DF
1. Background: IMC and data-flow analysis

2. Augmenting IMC with auxiliary invariants

3. Experimental evaluation
Strengthen Interpolants with Auxiliary Invariants

- Given an **inductive** invariant $Inv$, interpolant $\tau_i$ can be strengthened by
  \[ \tau'_i \leftarrow \tau_i \land Inv \]
Given an inductive invariant $Inv$, interpolant $\tau_i$ can be strengthened by

$$\tau'_i \leftarrow \tau_i \land Inv$$

$\tau'_i$ is a valid interpolant for IMC.
Strengthen Interpolants with Auxiliary Invariants

- Given an **inductive** invariant $\text{Inv}$, interpolant $\tau_i$ can be strengthened by

$$\tau'_i \leftarrow \tau_i \land \text{Inv}$$

- $\tau'_i$ is a valid interpolant for IMC

- $\text{Inv}$ helps remove some **unreachable** states in $\tau_i$

- $\tau'(s_0) \land \bigwedge_{i=1}^{k} T(s_{i-1}, s_i) \land (\bigvee_{i=1}^{k} \neg P(s_i))$ is more likely to remain UNSAT
Alternative Ways to Utilize Auxiliary Invariants

- Injecting auxiliary invariants into
  - Fixed-point check: $\text{Inv} \land \tau_n \Rightarrow \text{Init} \lor \bigvee_{i=1}^{n-1} \tau_i$
  - Safety property: $P' \leftarrow P \land \text{Inv}$

SoSy-Lab, LMU Munich

Augmenting IMC with Auxiliary Invariants
Alternative Ways to Utilize Auxiliary Invariants

- Injecting auxiliary invariants into
  - Fixed-point check: $\text{Inv} \land \tau_n \Rightarrow \text{Init} \lor \bigvee_{i=1}^{n-1} \tau_i$
  - Safety property: $P' \leftarrow P \land \text{Inv}$
- Not as effective as strengthening interpolants
Agenda

1. Background: IMC and data-flow analysis

2. Augmenting IMC with auxiliary invariants

3. Experimental evaluation
Evaluation

- We conducted experiments to answer the following research questions:
  - **RQ1**: Can auxiliary invariants help improve IMC?
  - **RQ2**: Is the augmented IMC competitive?
We conducted experiments to answer the following research questions:

- **RQ1**: Can auxiliary invariants help improve IMC?
- **RQ2**: Is the augmented IMC competitive?

Implementation and Configurations in CPAchecker

- **CPAchecker**: revision 42901 of branch *imc-with-invariants*
- Interpolants computed by **MathSAT5** [10] (theory: QF_ABVFPUF)

---

[^2]: [https://cpachecker.sosy-lab.org/](https://cpachecker.sosy-lab.org/)
Implementation and Configurations in CPAchecker

- **CPAchecker**: revision 42901 of branch *imc-with-invariants*
- Interpolants computed by **MathSAT5** [10] (theory: QF_ABVFPUF)
- Compared SMT-based algorithms
  - IMC [7] vs. IMC←DF
  - KI←DF [6], predicate abstraction [14], **IMPACT** [17]

²https://cpachecker.sosy-lab.org/
Benchmark Tools and Tasks

- Compared software verifiers (from SV-COMP 2022 [2])
  - 2LS [8]: $k$-induction boosted by auxiliary invariants
  - Symbiotic [18]: the overall winner of SV-COMP 2022
Benchmark Tools and Tasks

- Compared software verifiers (from SV-COMP 2022 [2])
  - 2LS [8]: $k$-induction boosted by auxiliary invariants
  - SYMBIOTIC [18]: the overall winner of SV-COMP 2022
- Benchmark set: ReachSafety tasks of SV-COMP 2022 [2]
  - No property violation (i.e., safe)
  - Focus on single-loop programs
  - Eliminate easy ones solvable by CPAchecker’s BMC within 900 s
  - 1623 after filtering
  - DF can produce non-trivial invariants on 870 tasks
Experimental Setup

- Environment
  - OS: Ubuntu 22.04 (64 bit)
  - Machine: 3.4 GHz CPU (8 cores) and 33 GB of RAM

- Each task is limited to
  - 4 CPU cores
  - 900 s of CPU time (max 150 s for DF)
  - 15 GB of RAM

(reliable resource management by \texttt{BenchExec}³)

³https://github.com/sosy-lab/benchexec
RQ1: Can auxiliary invariants help improve IMC?
Improved Effectiveness

<table>
<thead>
<tr>
<th>Task</th>
<th>IMC (timeout)</th>
<th>IMC-DF (solved)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#unroll #itp</td>
<td>wall-time</td>
</tr>
<tr>
<td>Problem03_label51</td>
<td>10 57 878</td>
<td>5 11 24.3</td>
</tr>
<tr>
<td>benchmark37_conj</td>
<td>317 316 892</td>
<td>1 2 1.83</td>
</tr>
<tr>
<td>s3_srvr_1a.BV.c.cil</td>
<td>64 441 885</td>
<td>5 13 6.26</td>
</tr>
</tbody>
</table>

(time unit: s; hand-picked tasks with significant improvement)

- IMC-DF solved 23 tasks where plain IMC ran into timeouts
  (on tasks with non-trivial auxiliary invariants)
Comparing Elapsed Wall-Time

![Graph comparing Elapsed Wall-Time](image)

- Wall-time (s)
- n-th fastest correct result (on 870 tasks with non-trivial invariants)

Legend:
- IMC
- IMC + DF

SoSy-Lab, LMU Munich
Augmenting IMC with Auxiliary Invariants
RQ2: Is the augmented IMC competitive?
Comparsion with Others

Wall-time (s)

n-th fastest correct result (on 1623 difficult tasks)
Answers to RQs

- **RQ1**: Can auxiliary invariants help improve IMC?

- **RQ2**: Is the augmented IMC competitive?
Answers to RQs

- **RQ1**: Can auxiliary invariants help improve IMC?
  Yes, effectiveness and wall-time efficiency are improved

- **RQ2**: Is the augmented IMC competitive?
Answers to RQs

- **RQ1**: Can auxiliary invariants help improve IMC?
  Yes, effectiveness and wall-time efficiency are improved

- **RQ2**: Is the augmented IMC competitive?
  Yes, more proofs compared to other verification algorithms and tools
Conclusion

- Augment IMC [16] via invariant injection
- Open-source implementation in CPAchecker

SoSy-Lab, LMU Munich

Augmenting IMC with Auxiliary Invariants


Example Revisited: Collecting Formulas

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

- Large-block encoding [5]
- \( s \leftarrow \{x, j\} \)
Example Revisited: Collecting Formulas

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3) x += 1;
        j += 1;
        if (j == 2) j = 0;
    }
    if (x % 2) { ERROR: return 1; }
    return 0;
}
```

- Large-block encoding [5]
- \( s \leftarrow \{x, j\} \)
- \( \text{Init}(s) \leftarrow (x = 0) \land (j = 0) \)
Example Revisited: Collecting Formulas

\begin{align*}
 & \text{Large-block encoding [5]} \\
 & s \leftarrow \{x, j\} \\
 & \text{} \quad \text{Init}(s) \leftarrow (x = 0) \land (j = 0) \\
 & \text{} \quad \text{T}(s, s') \leftarrow (x_1 = x + 2) \\
 & \text{} \quad \text{} \quad \land (j = 3 \Rightarrow x' = x_1 + 1) \\
 & \text{} \quad \text{} \quad \land (j \neq 3 \Rightarrow x' = x_1) \\
 & \text{} \quad \text{} \quad \land (j_1 = j + 1) \\
 & \text{} \quad \text{} \quad \land (j_1 = 2 \Rightarrow j' = 0) \\
 & \text{} \quad \text{} \quad \land (j_1 \neq 2 \Rightarrow j' = j_1) \\
\end{align*}

\begin{tabular}{|l|}
\hline
1 \textbf{int} \textbf{main(}\textbf{void}\textbf{)} \{ \\
2 \textbf{unsigned} \textbf{x} = 0; \\
3 \textbf{unsigned} \textbf{j} = 0; \\
4 \textbf{while} (\textbf{nondet}()) \{ \\
5 \textbf{x} += 2; \\
6 \textbf{if} (j == 3) \\
7 \textbf{x} += 1; \\
8 \textbf{j} += 1; \\
9 \textbf{if} (j == 2) \\
10 \textbf{j} = 0; \\
11 \} \\
12 \textbf{if} (x \% 2) \{ \\
13 \quad \textbf{ERROR: return} \textbf{1;} \\
14 \} \\
15 \textbf{return} \textbf{0;} \\
16 \} \\
\hline
\end{tabular}
Example Revisited: Collecting Formulas

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

- Large-block encoding [5]
- \( s \leftarrow \{x, j\} \)
- \( \text{Init}(s) \leftarrow (x = 0) \land (j = 0) \)
- \( \text{T}(s, s') \leftarrow (x_1 = x + 2) \land (j = 3 \Rightarrow x' = x_1 + 1) \land (j \neq 3 \Rightarrow x' = x_1) \land (j_1 = j + 1) \land (j_1 = 2 \Rightarrow j' = 0) \land (j_1 \neq 2 \Rightarrow j' = j_1) \)
- \( \text{P}(s') \leftarrow x' \% 2 = 0 \)
Example Revisited: Plain IMC

```c
int main(void) {
  unsigned x = 0;
  unsigned j = 0;
  while (nondet()) {
    x += 2;
    if (j == 3)
      x += 1;
    j += 1;
    if (j == 2)
      j = 0;
  }
  if (x % 2) {
    ERROR: return 1;
  }
  return 0;
}
```

One loop unrolling ($k = 1$)

- $Init \land T \land \neg P$ is UNSAT
- $\tau_1 \leftarrow x \% 2 = 0$
Example Revisited: Plain IMC

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3) x += 1;
        j += 1;
        if (j == 2) j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

One loop unrolling \((k = 1)\)

- \(Init \land T \land \neg P\) is UNSAT
- \(\tau_1 \leftarrow x \% 2 = 0\)
- \(\tau_1 \land T \land \neg P\) is SAT
  (spurious cex: \(x = 0 \land j = 3\))
Example Revisited: Plain IMC

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3) x += 1;
        j += 1;
        if (j == 2) j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

One loop unrolling \((k = 1)\)

- \(\text{Init} \land T \land \neg P\) is UNSAT
- \(\tau_1 \leftarrow x \% 2 = 0\)
- \(\tau_1 \land T \land \neg P\) is SAT
  (spurious cex: \(x = 0 \land j = 3\))
- Increment \(k\)
Example Revisited: Augmented IMC

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

One loop unrolling ($k = 1$)

- $Inv \leftarrow 0 \leq j \leq 1$
Example Revisited: Augmented IMC

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;

    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }

    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

One loop unrolling ($k = 1$)

- $Inv \leftarrow 0 \leq j \leq 1$
- $Init \land T \land \neg P$ is UNSAT
- $\tau_1 \leftarrow x \equiv 2 = 0$; $\tau'_1 \leftarrow \tau_1 \land Inv$
- $\tau'_1 \land T \land \neg P$ is UNSAT
Example Revisited: Augmented IMC

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3)
            x += 1;
        j += 1;
        if (j == 2)
            j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

One loop unrolling ($k = 1$)

- $\text{Inv} \leftarrow 0 \leq j \leq 1$
- $\text{Init} \land T \land \neg P$ is UNSAT
- $\tau_1 \leftarrow x \% 2 = 0; \quad \tau_1' \leftarrow \tau_1 \land \text{Inv}$
- $\tau_1' \land T \land \neg P$ is UNSAT
- $\tau_2 \leftarrow x \% 2 = 0; \quad \tau_2' \leftarrow \tau_2 \land \text{Inv}$
Example Revisited: Augmented IMC

```c
int main(void) {
    unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
        x += 2;
        if (j == 3) x += 1;
        j += 1;
        if (j == 2) j = 0;
    }
    if (x % 2) {
        ERROR: return 1;
    }
    return 0;
}
```

One loop unrolling \((k = 1)\)
- \(\text{Inv} \leftarrow 0 \leq j \leq 1\)
- \(\text{Init} \land T \land \neg P\) is UNSAT
- \(\tau_1 \leftarrow x \% 2 = 0; \quad \tau'_1 \leftarrow \tau_1 \land \text{Inv}\)
- \(\tau'_1 \land T \land \neg P\) is UNSAT
- \(\tau_2 \leftarrow x \% 2 = 0; \quad \tau'_2 \leftarrow \tau_2 \land \text{Inv}\)
- \(\tau'_2 \Rightarrow (I \lor \tau'_1)\) holds: fixed point!
Critical Reflection

- Auxiliary invariants brought *net improvement* to IMC
- However...
Critical Reflection

- Auxiliary invariants brought *net improvement* to IMC
- However...
  - The improvement was not remarkable
  - Some tasks became unsolvable with added invariants
- Reasons:
  - Invariant generator consumed additional CPU time
  - Interpolation queries became more difficult for the solver