SMT-based Bounded Model Checking for Multi-threaded Software in Embedded Systems

Lucas Cordeiro

lcc08r@ecs.soton.ac.uk
Embedded systems are ubiquitous but their verification becomes more difficult.

- functionality demanded increased significantly
  - peer reviewing and testing
- multi-core processors with scalable shared memory
  - but software model checkers focus on single-threaded or multi-threaded with message passing

```c
void *threadA(void *arg) {
    lock(&mutex);
    x++; if (x == 1) lock(&lock); unlock(&mutex); (CS1)
    lock(&mutex); (CS3)
    x--; if (x == 0) unlock(&lock);
    unlock(&mutex);
}

void *threadB(void *arg) {
    lock(&mutex);
    y++; if (y == 1) lock(&lock); (CS2)
    lock(&mutex);
    y--; if (y == 0) unlock(&lock);
    unlock(&mutex);
}
```

Deadlock
Bounded Model Checking (BMC)

Basic Idea: check negation of given property up to given depth

- transition system $M$ unrolled $k$ times
  - for programs: unroll loops, unfold arrays, ...
- translated into verification condition $\psi$ such that
  $\psi$ satisfiable iff $\varphi$ has counterexample of max. depth $k$
- has been applied successfully to verify (sequential) software
BMC of Multi-threaded Software

• concurrency bugs are tricky to reproduce/debug because they usually occur under specific thread interleavings
  – most common errors: 67% related to atomicity and order violations, 30% related to deadlock [Lu et al.’08]

• problem: the number of interleavings grows exponentially with the number of threads (n) and program statements (s)
  – number of executions: $O(n^s)$
  – context switches among threads increase the number of possible executions

• two important observations help us:
  – concurrency bugs are shallow [Qadeer&Rehof’05]
  – SAT/SMT solvers produce unsatisfiable cores that allow us to remove possible undesired models of the system
Objective of this work

Exploit SMT to extend BMC of embedded software

• exploit SMT solvers to:
  – encode full ANSI-C into the different background theories
  – prune the *property and data dependent* search space
  – remove interleavings that are not relevant by analyzing the proof of unsatisfiability

• propose three approaches to SMT-based BMC:
  – *lazy exploration* of the interleavings
  – *schedule guards* to encode all interleavings
  – *underapproximation and widening (UW)* [Grumberg et al.’05]

• evaluate our approaches implemented in ESBMC over embedded software applications
Agenda

• SMT-based BMC for Embedded ANSI-C Software

• Verifying Multi-threaded Software

• Implementation of ESBMC

• Integrating ESBMC into Software Engineering Practice

• Conclusions and Future Work
Satisfiability Modulo Theories (1)

SMT decides the **satisfiability** of first-order logic formulae using the combination of different **background theories** ($\Rightarrow$ building-in operators).

<table>
<thead>
<tr>
<th>Theory</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>$x_1=x_2 \land \neg (x_1=x_3) \Rightarrow \neg (x_1=x_3)$</td>
</tr>
<tr>
<td>Bit-vectors</td>
<td>$(b &gt;&gt; i) &amp; 1 = 1$</td>
</tr>
<tr>
<td>Linear arithmetic</td>
<td>$(4y_1 + 3y_2 \geq 4) \lor (y_2 - 3y_3 \leq 3)$</td>
</tr>
<tr>
<td>Arrays</td>
<td>$(j = k \land a[k]=2) \Rightarrow a[j]=2$</td>
</tr>
<tr>
<td>Combined theories</td>
<td>$(j \leq k \land a[j]=2) \Rightarrow a[i] &lt; 3$</td>
</tr>
</tbody>
</table>
Satisfiability Modulo Theories (2)

• Given
  – a decidable $\Sigma$-theory $T$
  – a quantifier-free formula $\varphi$

$\varphi$ is $T$-satisfiable iff $T \cup \{\varphi\}$ is satisfiable, i.e., there exists a structure that satisfies both formula and sentences of $T$

• Given
  – a set $\Gamma \cup \{\varphi\}$ of first-order formulae over $T$

$\varphi$ is a $T$-consequence of $\Gamma$ ($\Gamma \models_T \varphi$) iff every model of $T \cup \Gamma$ is also a model of $\varphi$

• Checking $\Gamma \models_T \varphi$ can be reduced in the usual way to checking the $T$-satisfiability of $\Gamma \cup \{\neg \varphi\}$
Satisfiability Modulo Theories (3)

- Let \( \mathbf{a} \) be an array, \( \mathbf{b} \), \( \mathbf{c} \) and \( \mathbf{d} \) be signed bit-vectors of width 16, 32 and 32 respectively, and let \( g \) be an unary function.

\[
g \left( \text{select} \left( \text{store} \left( \mathbf{a}, \mathbf{c}, 12 \right) \right), \text{SignExt} \left( \mathbf{b}, 16 \right) + 3 \right)
\neq g \left( \text{SignExt} \left( \mathbf{b}, 16 \right) - \mathbf{c} + 4 \right) \land \text{SignExt} \left( \mathbf{b}, 16 \right) = \mathbf{c} - 3 \land \mathbf{c} + 1 = \mathbf{d} - 4
\]

\[\downarrow\]

\( \mathbf{b}' \) extends \( \mathbf{b} \) to the signed equivalent bit-vector of size 32

**step 1:** \( g \left( \text{select} \left( \text{store} \left( \mathbf{a}, \mathbf{c}, 12 \right), \mathbf{b}'+3 \right) \right) \neq g \left( \mathbf{b}' - \mathbf{c} + 4 \right) \land \mathbf{b}' = \mathbf{c} - 3 \land \mathbf{c} + 1 = \mathbf{d} - 4 \)

\[\downarrow\]

replace \( \mathbf{b}' \) by \( \mathbf{c} - 3 \) in the inequality

**step 2:** \( g \left( \text{select} \left( \text{store} \left( \mathbf{a}, \mathbf{c}, 12 \right), \mathbf{c} - 3 + 3 \right) \right) \neq g \left( \mathbf{c} - 3 - \mathbf{c} + 4 \right) \land \mathbf{c} - 3 = \mathbf{c} - 3 \land \mathbf{c} + 1 = \mathbf{d} - 4 \)

\[\downarrow\]

using facts about bit-vector arithmetic

**step 3:** \( g \left( \text{select} \left( \text{store} \left( \mathbf{a}, \mathbf{c}, 12 \right), \mathbf{c} \right) \right) \neq g \left( 1 \right) \land \mathbf{c} - 3 = \mathbf{c} - 3 \land \mathbf{c} + 1 = \mathbf{d} - 4 \)
Satisfiability Modulo Theories (4)

**step 3:** \( g(\text{select}(\text{store}(a, c, 12), c)) \neq g(1) \land c - 3 = c - 3 \land c + 1 = d - 4 \)

applying the theory of arrays

**step 4:** \( g(12) \neq g(1) \land c - 3 \land c + 1 = d - 4 \)

The function \( g \) implies that for all \( x \) and \( y \),
if \( x = y \), then \( g(x) = g(y) \) (*congruence rule*).

**step 5:** SAT \( (c = 5, d = 10) \)

- SMT solvers also apply:
  - standard algebraic reduction rules
    \[ r \land \text{false} \leftrightarrow \text{false} \]
    \[ a = 7 \land p(a) \leftrightarrow a = 7 \land p(7) \]
  - contextual simplification
Software BMC using ESBMC

- program modelled as state transition system
  - *state*: program counter and program variables
  - derived from control-flow graph
  - checked safety properties give extra nodes
- program unfolded up to given bounds
  - loop iterations
  - context switches
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions

```c
int main() {
    int a[2], i, x;
    if (x==0)
        a[i]=0;
    else
        a[i+2]=1;
    assert(a[i+1]==1);
}
```
Software BMC using ESBMC

- program modelled as state transition system
  - state: program counter and program variables
  - derived from control-flow graph
  - checked safety properties give extra nodes
- program unfolded up to given bounds
  - loop iterations
  - context switches
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions
- front-end converts unrolled and optimized program into SSA

```c
int main() {
    int a[2], i, x;
    if (x==0)
        a[i]=0;
    else
        a[i+2]=1;
    assert(a[i+1]==1);
}
```

\[
\begin{align*}
g_1 &= x_1 == 0 \\
a_1 &= a_0 \text{ WITH } [i_0:=0] \\
a_2 &= a_0 \\
a_3 &= a_2 \text{ WITH } [2+i_0:=1] \\
a_4 &= g_1 \ ? \ a_1 : a_3 \\
t_1 &= a_4 [1+i_0] == 1
\end{align*}
\]
Software BMC using ESBMC

- program modelled as state transition system
  - *state*: program counter and program variables
  - derived from control-flow graph
  - checked safety properties give extra nodes
- program unfolded up to given bounds
  - loop iterations
  - context switches
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions \{ crucial \}
- front-end converts unrolled and optimized program into SSA
- extraction of constraints $C$ and properties $P$
  - specific to selected SMT solver, uses theories
- satisfiability check of $C \land \neg P$

```c
int main() {
    int a[2], i, x;
    if (x==0)
        a[i]=0;
    else
        a[i+2]=1;
    assert(a[i+1]==1);
}
```
Encoding of Numeric Types

- SMT solvers typically provide different encodings for numbers:
  - abstract domains \( (\mathbb{Z}, \mathbb{R}) \)
  - fixed-width bit vectors (unsigned int, ...)
    \[ \triangleright \text{“internalized bit-blasting”} \]
- verification results can depend on encodings
  \[ (a > 0) \land (b > 0) \Rightarrow (a + b > 0) \]
  - valid in abstract domains such as \( \mathbb{Z} \) or \( \mathbb{R} \)
  - doesn’t hold for bitvectors, due to possible overflows

- majority of VCs solved faster if numeric types are modelled by abstract domains but possible loss of precision
- ESBMC supports both types of encoding and also combines them to improve scalability and precision
Encoding Numeric Types as Bitvectors

Bitvector encodings need to handle

- type casts and implicit conversions
  - arithmetic conversions implemented using word-level functions
    (part of the bitvector theory: Extract, SignExt, ...)
    - different conversions for every pair of types
    - uses type information provided by front-end
  - conversion to / from bool via if-then-else operator
    \[ t = \text{ite}(v \neq k, \text{true}, \text{false}) \quad //\text{conversion to bool} \]
    \[ v = \text{ite}(t, 1, 0) \quad //\text{conversion from bool} \]

- arithmetic over- / underflow
  - standard requires modulo-arithmetic for unsigned integer
    \[ \text{unsigned\_overflow} \iff (r - (r \mod 2^w)) < 2^w \]
  - define error literals to detect over- / underflow for other types
    \[ \text{res\_op} \iff \neg \text{overflow}(x, y) \land \neg \text{underflow}(x, y) \]
    - similar to conversions
Floating-Point Numbers

- over-approximate floating-point by fixed-point numbers
  - encode the integral (i) and fractional (f) parts
- **binary encoding**: get a new bit-vector \( b = i @ f \) with the same bitwidth before and after the radix point of \( a \).

\[
i = \begin{cases} 
  \text{Extract}(b, n_b + m_a - 1, n_b) & : m_a \leq m_b \\
  \text{SignExt}(\text{Extract}(b, t_b - 1, n_b), m_a - m_b) & : \text{otherwise}
\end{cases} \\
\text{f} = \begin{cases} 
  \text{Extract}(b, n_b - 1, n_b - n_b) & : n_a \leq n_b \\
  \text{ZeroExt}(\text{Extract}(b, n_b - 1, 0), n_a - n_b) & : \text{otherwise}
\end{cases}
\]

- **rational encoding**: convert \( a \) to a rational number

\[
a = \begin{cases} 
  \left( \frac{i \times p + \left( \frac{f \times p}{2^n} + 1 \right)}{p} \right) & : f \neq 0 \\
  i & : \text{otherwise}
\end{cases} \\
\text{where } p = \text{number of decimal places}
\]
Encoding of Pointers

- arrays and records / tuples typically handled directly by SMT-solver
- pointers modelled as tuples
  - p.o ≜ representation of underlying object
  - p.i ≜ index (if pointer used as array base)

```c
int main() {
    int a[2], i, x, *p;
    p=a;
    if (x==0)
        a[i]=0;
    else
        a[i+1]=1;
    assert(*(p+2)==1);
}
```

\[
\begin{align*}
p_1 & := \text{store}(p_0, 0, &a[0]) \\
p_2 & := \text{store}(p_1, 1, 0) \\
g_2 & := (x_2 == 0) \\
a_1 & := \text{store}(a_0, i_0) \\
& a_2 := \text{select}(a_1, a_1, a_3) \\
p_3 & := \text{store}(p_2, 1, \text{select}(p_2, 1)+2)
\end{align*}
\]
Encoding of Pointers

- arrays and records / tuples typically handled directly by SMT-solver
- pointers modelled as tuples
  - p.o ≝ representation of underlying object
  - p.i ≝ index (if pointer used as array base)

```c
int main() {
    int a[2], i, x, *p;
p=a;
if (x==0)
a[i]=0;
else
    a[i+1]=1;
assert(*(p+2)==1);
}
```


\[
P := \begin{cases} 
    i_0 \geq 0 \land i_0 < 2 \\
    1 + i_0 \geq 0 \land 1 + i_0 < 2 \\
    \land \text{select}(p_3, 0) = \&a[0] \\
    \land \text{select}(\text{select}(p_3, 0), \\
    \text{select}(p_3, 1)) == 1
\end{cases}
\]

negation satisfiable
(a[2] unconstrained) ⇒ assert fails
Encoding of Memory Allocation

- model memory just as an array of bytes (*array theories*)
  - read and write operations to the memory array on the logic level
- each dynamic object $d_o$ consists of
  - $m \triangleq$ memory array
  - $s \triangleq$ size in bytes of $m$
  - $\rho \triangleq$ unique identifier
  - $v \triangleq$ indicate whether the object is still alive
  - $l \triangleq$ the location in the execution where $m$ is allocated
- to detect invalid reads/writes, we check whether
  - $d_o$ is a dynamic object
  - $i$ is within the bounds of the memory array

$$
\text{is\_dynamic\_object} \iff \left( \bigvee_{j=1}^{k} d_o \cdot \rho = j \right) \land (0 \leq i < n)
$$
Encoding of Memory Allocation

- to check for invalid objects, we
  - set \( v \) to \textit{true} when the function \textit{malloc} is called (\( d_o \) is alive)
  - set \( v \) to \textit{false} when the function \textit{free} is called (\( d_o \) is not longer alive)

\[
I_{\text{valid\_object}} \iff (I_{\text{is\_dynamic\_object}} \Rightarrow d_o \cdot v)
\]

- to detect forgotten memory, at the end of the (unrolled) program we check
  - whether the \( d_o \) has been deallocated by the function \textit{free}

\[
I_{\text{deallocated\_object}} \iff (I_{\text{is\_dynamic\_object}} \Rightarrow \neg d_o \cdot v)
\]
Example of Memory Allocation

```c
#include <stdlib.h>
void main() {
    char *p = malloc(5);
    char *q = malloc(5);
    p=q;
    free(p)
    p = malloc(5);    // ρ = 3
    free(p)
}
```

memory leak: pointer reassignment makes d_{o1,v} to become an orphan
Example of Memory Allocation

```c
#include <stdlib.h>
void main() {
    char *p = malloc(5); // ρ = 1
    char *q = malloc(5); // ρ = 2
    p = q;              // p := (¬d_{o1}.v ∧ ¬d_{o2}.v ∧ ¬d_{o3}.v)
    free(p)
    p = malloc(5);      // ρ = 3
    free(p)
}
```

\[
C := \begin{cases} 
  d_{o1}.ρ=1 \land d_{o1}.s=5 \land d_{o1}.v=true \land p=d_{o1} \\
  \land d_{o2}.ρ=2 \land d_{o2}.s=5 \land d_{o2}.v=true \land q=d_{o2} \\
  \land p=d_{o2} \land d_{o2}.v=false \\
  \land d_{o3}.ρ=3 \land d_{o3}.s=5 \land d_{o3}.v=true \land p=d_{o3} \\
  \land d_{o3}.v=false 
\end{cases}
\]
Example of Memory Allocation

```c
#include <stdlib.h>

void main() {
    char *p = malloc(5); // ρ = 1
    char *q = malloc(5); // ρ = 2
    p = q;
    free(p)
    p = malloc(5); // ρ = 3
    free(p)
}
```

\[
P := \neg d_{o1}.v \land \neg d_{o2}.v \land \neg d_{o3}.v
\]

\[
C := \left\{ \begin{array}{l}
d_{o1}.\rho = 1 \land d_{o1}.s = 5 \land d_{o1}.v = true \land p = d_{o1} \\
\land d_{o2}.\rho = 2 \land d_{o2}.s = 5 \land d_{o2}.v = true \land q = d_{o2} \\
\land p = d_{o2} \land d_{o2}.v = false \\
\land d_{o3}.\rho = 3 \land d_{o3}.s = 5 \land d_{o3}.v = true \land p = d_{o3} \\
\land d_{o3}.v = false
\end{array} \right. \]
Evaluation
Comparison of SMT solvers

• Goal: compare efficiency of different SMT-solvers
  – CVC3 (2.2)
  – Boolector (1.4)
  – Z3 (2.11)

• Set-up:
  – identical ESBMC front-end, individual back-ends
  – operations not supported by SMT-solvers are axiomatized
  – standard desktop PC, time-out 3600 seconds
## Comparison of SMT solvers

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>CVC3</th>
<th>Booletor</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td>Error</td>
<td>Time</td>
</tr>
<tr>
<td>SelectionSort (n=35)</td>
<td>34</td>
<td>17</td>
<td>16</td>
<td>5 (5)</td>
<td>161</td>
</tr>
<tr>
<td>(n=140)</td>
<td>34</td>
<td>17</td>
<td>Mb</td>
<td>(209)</td>
<td></td>
</tr>
<tr>
<td>InsertionSort (n=35)</td>
<td>86</td>
<td>17</td>
<td>4</td>
<td>5 (5)</td>
<td>350</td>
</tr>
<tr>
<td>(n=140)</td>
<td>86</td>
<td>17</td>
<td>194</td>
<td>(283)</td>
<td></td>
</tr>
<tr>
<td>Prim</td>
<td>24</td>
<td>5</td>
<td>0</td>
<td>&lt;1 (&lt;1)</td>
<td>0</td>
</tr>
<tr>
<td>StrCmp</td>
<td>0</td>
<td>195</td>
<td>(257)</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>MinMax</td>
<td>19</td>
<td>9</td>
<td>Tb</td>
<td>(Mb)</td>
<td>1</td>
</tr>
<tr>
<td>Ims</td>
<td>258</td>
<td>23</td>
<td>225</td>
<td>(324)</td>
<td>0</td>
</tr>
<tr>
<td>Bitwise</td>
<td>18</td>
<td>1</td>
<td>3</td>
<td>(6)</td>
<td>7</td>
</tr>
<tr>
<td>adpcm_encode</td>
<td>149</td>
<td>12</td>
<td>6</td>
<td>(26)</td>
<td>6</td>
</tr>
<tr>
<td>adpcm_decode</td>
<td>111</td>
<td>10</td>
<td>3</td>
<td>(27)</td>
<td>3</td>
</tr>
</tbody>
</table>

**Notes:**
- **Lines of code**
- **Number of properties checked**
- **SMT-LIB interface**
- **Native API**
## Comparison of SMT solvers

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>CVC3</th>
<th>Boolector</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td>Error</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BubbleSort (n=35)</td>
<td>43</td>
<td>17</td>
<td>17 (5)</td>
<td>0</td>
<td>2 (2)</td>
</tr>
<tr>
<td>(n=140)</td>
<td>43</td>
<td>17</td>
<td>$M_b$</td>
<td>1</td>
<td>282 (311)</td>
</tr>
<tr>
<td>SelectionSort (n=35)</td>
<td>34</td>
<td>17</td>
<td>18 (3)</td>
<td>0</td>
<td>1 (1)</td>
</tr>
<tr>
<td>(n=140)</td>
<td>34</td>
<td>17</td>
<td>$M_b$</td>
<td>1</td>
<td>161 (171)</td>
</tr>
<tr>
<td>InsertionSort (n=35)</td>
<td>8</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>3 (3)</td>
</tr>
<tr>
<td>(n=140)</td>
<td>8</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>212 (222)</td>
</tr>
<tr>
<td>Prim</td>
<td>7</td>
<td></td>
<td>0</td>
<td>&lt;1 (&lt;1)</td>
<td>0</td>
</tr>
<tr>
<td>StrCmp</td>
<td>14</td>
<td>6</td>
<td>195 (257)</td>
<td>0</td>
<td>35 (46)</td>
</tr>
<tr>
<td>MinMax</td>
<td>19</td>
<td></td>
<td>42 (7)</td>
<td>0</td>
<td>6 (7)</td>
</tr>
<tr>
<td>Lms</td>
<td>258</td>
<td>23</td>
<td>303 (307)</td>
<td>0</td>
<td>306 (307)</td>
</tr>
<tr>
<td>Bitwise</td>
<td>18</td>
<td>1</td>
<td>3 (6)</td>
<td>0</td>
<td>7 (8)</td>
</tr>
<tr>
<td>adpcm_encode</td>
<td>149</td>
<td>12</td>
<td>6 (26)</td>
<td>0</td>
<td>6 (6)</td>
</tr>
<tr>
<td>adpcm_decode</td>
<td>111</td>
<td>10</td>
<td>3 (27)</td>
<td>0</td>
<td>3 (3)</td>
</tr>
</tbody>
</table>

*All SMT-solvers can handle the VCs from the embedded applications*
## Comparison of SMT solvers

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>CVC3</th>
<th>Booletor</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td>Error</td>
<td>Time</td>
</tr>
<tr>
<td>BubbleSort (n=35)</td>
<td>43</td>
<td>17</td>
<td>17 (5)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(n=140)</td>
<td>43</td>
<td>17</td>
<td>M_b(M_b)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SelectionSort (n=35)</td>
<td>34</td>
<td>17</td>
<td>18 (3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(n=140)</td>
<td>34</td>
<td>17</td>
<td>M_b (209)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>InsertionSort (n=35)</td>
<td>86</td>
<td>17</td>
<td>4 (5)</td>
<td>0</td>
<td>3 (3)</td>
</tr>
<tr>
<td>(n=140)</td>
<td>86</td>
<td>17</td>
<td>194 (283)</td>
<td>0</td>
<td>350 (219)</td>
</tr>
<tr>
<td>Prim</td>
<td>79</td>
<td>30</td>
<td>5 (2)</td>
<td>0</td>
<td>&lt;1 (&lt;1)</td>
</tr>
<tr>
<td>StrCmp</td>
<td>14</td>
<td>6</td>
<td>11 (454)</td>
<td>0</td>
<td>195 (257)</td>
</tr>
<tr>
<td>MinMax</td>
<td>19</td>
<td>9</td>
<td>T_b (Mb)</td>
<td>1</td>
<td>42 (7)</td>
</tr>
<tr>
<td>lms</td>
<td>258</td>
<td>23</td>
<td>225 (324)</td>
<td>0</td>
<td>303 (307)</td>
</tr>
<tr>
<td>Bitwise</td>
<td>18</td>
<td>1</td>
<td>3 (6)</td>
<td>0</td>
<td>7 (8)</td>
</tr>
<tr>
<td>adpcm_encode</td>
<td>149</td>
<td>12</td>
<td>6 (26)</td>
<td>0</td>
<td>6 (6)</td>
</tr>
<tr>
<td>adpcm_decode</td>
<td>111</td>
<td>10</td>
<td>3 (27)</td>
<td>0</td>
<td>3 (3)</td>
</tr>
</tbody>
</table>

**CVC3 doesn’t scale that well and runs out of memory and time**
## Comparison of SMT solvers

<table>
<thead>
<tr>
<th>Module</th>
<th>Boolector</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td><strong>Error</strong></td>
<td><strong>Time</strong></td>
</tr>
<tr>
<td><strong>BubbleSort (n=35)</strong></td>
<td>43 17</td>
<td>M_b(M_b)</td>
</tr>
<tr>
<td><strong>SelectionSort (n=35)</strong></td>
<td>34 17</td>
<td>18 (3)</td>
</tr>
<tr>
<td><strong>InsertionSort (n=35)</strong></td>
<td>86 17</td>
<td>4 (5)</td>
</tr>
<tr>
<td><strong>Prim</strong></td>
<td>79 30</td>
<td>5 (2)</td>
</tr>
<tr>
<td><strong>StrCmp</strong></td>
<td>14 6</td>
<td>11 (454)</td>
</tr>
<tr>
<td><strong>MinMax</strong></td>
<td>19 9</td>
<td>T_b (Mb)</td>
</tr>
<tr>
<td><strong>Lms</strong></td>
<td>258 23</td>
<td>225 (324)</td>
</tr>
<tr>
<td><strong>Bitwise</strong></td>
<td>18 1</td>
<td>3 (6)</td>
</tr>
<tr>
<td><strong>adpcm_encode</strong></td>
<td>149 12</td>
<td>6 (26)</td>
</tr>
<tr>
<td><strong>adpcm_decode</strong></td>
<td>111 10</td>
<td>3 (27)</td>
</tr>
</tbody>
</table>

*Boolector and Z3 roughly comparable, with some advantages for Z3*
## Comparison of SMT solvers

The native API is slightly faster than the SMT-LIB interface

<table>
<thead>
<tr>
<th>Model</th>
<th>Model 3C</th>
<th>Boolector</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Error</td>
<td>Time</td>
</tr>
<tr>
<td>BubbleSort (n=35)</td>
<td>43</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>(n=140)</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>SelectionSort (n=35)</td>
<td>34</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>(n=140)</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>InsertionSort (n=35)</td>
<td>86</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>(n=140)</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Prim</td>
<td>79</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>StrCmp</td>
<td>14</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>MinMax</td>
<td>19</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Lms</td>
<td>258</td>
<td>23</td>
<td>225</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Bitwise</td>
<td>18</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>adpcm_encode</td>
<td>149</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>adpcm_decode</td>
<td>111</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
# Comparison of SMT solvers

The native API is slightly faster than the SMT-LIB interface, **but not always**

<table>
<thead>
<tr>
<th>Model</th>
<th>Time</th>
<th>Error</th>
<th>Time</th>
<th>Error</th>
<th>Time</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BubbleSort</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=35</td>
<td>43</td>
<td>17</td>
<td>17</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>n=140</td>
<td>43</td>
<td>17</td>
<td>17 M_b(M_b)</td>
<td>1</td>
<td>282</td>
<td>265</td>
</tr>
<tr>
<td><strong>SelectionSort</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=35</td>
<td>34</td>
<td>17</td>
<td>18</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>n=140</td>
<td>34</td>
<td>17</td>
<td>18 M_b (209)</td>
<td>1</td>
<td>161</td>
<td>165</td>
</tr>
<tr>
<td><strong>InsertionSort</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=35</td>
<td>86</td>
<td>17</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>n=140</td>
<td>86</td>
<td>17</td>
<td>4 M_b (283)</td>
<td>0</td>
<td>350</td>
<td>212</td>
</tr>
<tr>
<td><strong>Prim</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>30</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>&lt;1</td>
</tr>
<tr>
<td><strong>StrCmp</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>6</td>
<td>11</td>
<td>(454)</td>
<td>0</td>
<td>195</td>
</tr>
<tr>
<td><strong>MinMax</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>9</td>
<td>T_b (Mb)</td>
<td>1</td>
<td>42</td>
<td>6</td>
</tr>
<tr>
<td><strong>Lms</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>258</td>
<td>23</td>
<td>225</td>
<td>(324)</td>
<td>0</td>
<td>303</td>
</tr>
<tr>
<td><strong>Bitwise</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>1</td>
<td>3</td>
<td>(6)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td><strong>adpcm_encode</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>149</td>
<td>12</td>
<td>6</td>
<td>(26)</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td><strong>adpcm_decode</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>10</td>
<td>3</td>
<td>(27)</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
Comparison to SMT-CBMC [A. Armando et al.]

- SMT-based BMC for C, built on top of CVC3 (hard-coded)
  - limited coverage of language
- Goal: compare efficiency of encodings

<table>
<thead>
<tr>
<th>Module</th>
<th>ESBMC</th>
<th>SMT-CBMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Z3</td>
<td>CVC3</td>
</tr>
<tr>
<td>BubbleSort</td>
<td>&lt;1 (&lt;1)</td>
<td>2 (2)</td>
</tr>
<tr>
<td>(n=35) (n=140)</td>
<td>259 (265)</td>
<td>M_b (M_b)</td>
</tr>
<tr>
<td>SelectionSort</td>
<td>&lt;1 (&lt;1)</td>
<td>&lt;1 (&lt;1)</td>
</tr>
<tr>
<td>(n=35) (n=140)</td>
<td>157 (162)</td>
<td>160 (193)</td>
</tr>
<tr>
<td>BellmanFord</td>
<td>&lt;1 (&lt;1)</td>
<td>&lt;1 (&lt;1)</td>
</tr>
<tr>
<td>Prim</td>
<td>&lt;1 (&lt;1)</td>
<td>&lt;1 (&lt;1)</td>
</tr>
<tr>
<td>StrCmp</td>
<td>27 (38)</td>
<td>7 (261)</td>
</tr>
<tr>
<td>SumArray</td>
<td>25 (&lt;1)</td>
<td>&lt;1 (108)</td>
</tr>
<tr>
<td>MinMax</td>
<td>6 (6)</td>
<td>T_b (M_b)</td>
</tr>
</tbody>
</table>
Comparison to SMT-CBMC [A. Armando et al.]

- SMT-based BMC for C, built on top of CVC3 (hard-coded)
  - limited coverage of language
- Goal: compare efficiency of encodings

<table>
<thead>
<tr>
<th>Module</th>
<th>Z3 (s)</th>
<th>CVC3 (s)</th>
<th>CVC3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BubbleSort</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n=35) (n=140)</td>
<td>&lt;1 (&lt;1)</td>
<td>2 (2)</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>259 (265)</td>
<td>M_b (M_b)</td>
<td>MO</td>
</tr>
<tr>
<td><strong>SelectionSort</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n=35) (n=140)</td>
<td>&lt;1 (&lt;1)</td>
<td>&lt;1 (&lt;1)</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>157 (162)</td>
<td>160 (193)</td>
<td>T</td>
</tr>
<tr>
<td><strong>BellmanFord</strong></td>
<td>&lt;1 (&lt;1)</td>
<td>&lt;1 (&lt;1)</td>
<td>43</td>
</tr>
<tr>
<td><strong>Prim</strong></td>
<td>&lt;1 (&lt;1)</td>
<td>&lt;1 (&lt;1)</td>
<td>96</td>
</tr>
<tr>
<td><strong>StrCmp</strong></td>
<td>27 (38)</td>
<td>7 (261)</td>
<td>T</td>
</tr>
<tr>
<td><strong>SumArray</strong></td>
<td>25 (&lt;1)</td>
<td>&lt;1 (108)</td>
<td>98</td>
</tr>
<tr>
<td><strong>MinMax</strong></td>
<td>6 (6)</td>
<td>T_b (M_b)</td>
<td>65</td>
</tr>
</tbody>
</table>

All benchmarks taken from SMT-CBMC suite
Comparison to SMT-CBMC [A. Armando et al.]

- SMT-based BMC for C, built on top of CVC3 (hard-coded)
  - limited coverage of language
- Goal: compare efficiency of encodings

<table>
<thead>
<tr>
<th>Module</th>
<th>ESBMC</th>
<th>SMT-CBMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Z3 CVC3</td>
<td>Z3 CVC3</td>
</tr>
<tr>
<td>BubbleSort</td>
<td>(&lt;1) 2 (2)</td>
<td>100 MO</td>
</tr>
<tr>
<td>(n=35)</td>
<td>Mb (Mb)</td>
<td>Mb (Mb)</td>
</tr>
<tr>
<td></td>
<td>&lt;1 (&lt;1) 160 (193)</td>
<td>&lt;1 (&lt;1) T T</td>
</tr>
<tr>
<td>Prim</td>
<td>&lt;1 (&lt;1) &lt;1 (&lt;1)</td>
<td>96</td>
</tr>
<tr>
<td>StrCmp</td>
<td>27 (38) &lt;7 (261)</td>
<td>T</td>
</tr>
<tr>
<td>SumArray</td>
<td>25 (&lt;1) &lt;1 (108)</td>
<td>98</td>
</tr>
<tr>
<td>MinMax</td>
<td>6 (6) Tb (Mb)</td>
<td>65</td>
</tr>
</tbody>
</table>

ESBMC substantially faster, even with identical solvers ⇒ probably better encoding
Comparison to SMT-CBMC [A. Armando et al.]

- SMT-based BMC for C, built on top of CVC3 (hard-coded)
  - limited coverage of language
- Goal: compare efficiency of encodings

<table>
<thead>
<tr>
<th>Module</th>
<th>ESBMC</th>
<th>SMT-CBMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Z3</td>
<td>CVC3</td>
</tr>
<tr>
<td>BubbleSort</td>
<td>259 (265)</td>
<td>2 (2)</td>
</tr>
<tr>
<td>(n=35)</td>
<td>&lt;1 (1)</td>
<td>157 (162)</td>
</tr>
<tr>
<td>(n=10)</td>
<td></td>
<td>160 (193)</td>
</tr>
<tr>
<td>BellmanFord</td>
<td>&lt;1 (1)</td>
<td>&lt;1 (1)</td>
</tr>
<tr>
<td>Prim</td>
<td>&lt;1 (1)</td>
<td>&lt;1 (1)</td>
</tr>
<tr>
<td>StrCmp</td>
<td>27 (38)</td>
<td>7 (261)</td>
</tr>
<tr>
<td>SumArray</td>
<td>25 (&lt;1)</td>
<td>&lt;1 (108)</td>
</tr>
<tr>
<td>MinMax</td>
<td>6 (6)</td>
<td>T</td>
</tr>
</tbody>
</table>
Agenda

- SMT-based BMC for Embedded ANSI-C Software
- Verifying Multi-threaded Software
- Implementation of ESBMC
- Integrating ESBMC into Software Engineering Practice
- Conclusions and Future Work
Lazy exploration of interleavings

Idea: iteratively generate all possible interleavings and call the BMC procedure on each interleaving

multi-threaded goto programs

C/C++ source → IRep tree

scan, parse, and type-check

properties

BMC

symbolic execution engine

verification conditions

SMT solver

deadlock, atomicity and order violations, etc...

reused/extended from the Cprover framework
Lazy exploration of interleavings

Idea: iteratively generate all possible interleavings and call the BMC procedure on each interleaving

- multi-threaded goto programs
- symbolic execution engine
- QF formula generation
- guide the symbolic execution

C/C++ source

IRep tree

scheduler

BMC

verification conditions

SMT solver

scan, parse, and type-check

properties

reused/extended from the Cprover framework

deadlock, atomicity and order violations, etc…

check satisfiability using an SMT solver

stop the generate-and-test loop if there is an error
Lazy exploration of interleavings

- Main steps of the algorithm:

1. Initialize the stack with the initial node $v_0$ and the initial path $\pi_0 = \langle v_0 \rangle$
2. If the stack is empty, terminate with “no error”.
3. Pop the current node $v$ and current path $\pi$ off the stack and compute the set $v'$ of successors of $v$ using rules R1-R8.
4. If $v'$ is empty, derive the VC $\varphi_k^{\pi}$ for $\pi$ and call the SMT solver on it. If $\varphi_k^{\pi}$ is satisfiable, terminate with “error”; otherwise, goto step 2.
5. If $v'$ is not empty, then for each node $v \in v'$, add $v$ to $\pi$, and push node and extended path on the stack. goto step 3.

\[
\begin{align*}
\pi &= \{v_1, \ldots v_n\} \\
\varphi_k^{\pi} &= I(s_0) \land R(s_0, s_1) \land \ldots \land R(s_{k-1}, s_k) \land \neg \phi_k
\end{align*}
\]
Running Example

• the program has sequences of operations that need to be protected together to avoid atomicity violation
  – requirement: the region of code (val1 and val2) should execute atomically

Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
7:  lock(m2);
8:  if (val1 == 0) {
9:    val1 = 1;
10:   unlock(m1);
11:   return NULL; }
12:  t1 = val1;
13:  unlock(m1);
14:  lock(m2);
15:  t2 = val2;
16:  unlock(m2);
17:  assert(t2==(t1+1));

program counter: 0
mutexes: m1=0; m2=0;
global variables: val1=0; val2=0;
local variables: t1= -1; t2= -1;
Lazy exploration: interleaving $I_s$

statements:
val1-access:
val2-access:

Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);

Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:     unlock(m1);
10:    return NULL;
11:  }
12:  t1 = val1;
13:  unlock(m1);
14:  t2 = val2;
15:  unlock(m2);
16:  assert(t2==(t1+1));

program counter: 0
mutexes: m1=0; m2=0;
global variables: val1=0; val2=0;
local variables: t1= -1; t2= -1;
Lazy exploration: interleaving $l_s$

statements: 1
val1-access:
val2-access:

Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);

Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:   return NULL;
11:  t1 = val1;
12:  unlock(m1);
13:  lock(m2);
14:  t2 = val2;
15:  unlock(m2);
16:  assert(t2===(t1+1));

program counter: 1
mutexes: $m1=1; m2=0;$
global variables: val1=0; val2=0;
local variables: t1= -1; t2= -1;
Lazy exploration: interleaving $I_s$

statements: 1-2
val1-access: $W_{\text{twoStage},2}$
val2-access:

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL;
}
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2 == (t1+1));

program counter: 2
mutexes: m1=1; m2=0;
global variables: val1=1; val2=0;
local variables: t1= -1; t2= -1;
Lazy exploration: interleaving $I_s$

statements: 1-2-3
val1-access: $W_{\text{twoStage},2}$
val2-access: 

<table>
<thead>
<tr>
<th>Thread twoStage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: lock(m1);</td>
</tr>
<tr>
<td>2: val1 = 1;</td>
</tr>
<tr>
<td><strong>3: unlock(m1);</strong></td>
</tr>
<tr>
<td>4: lock(m2);</td>
</tr>
<tr>
<td>5: val2 = val1 + 1;</td>
</tr>
<tr>
<td>6: unlock(m2);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>7: lock(m1);</td>
</tr>
<tr>
<td>8: if (val1 == 0) {</td>
</tr>
<tr>
<td>9: unlock(m1);</td>
</tr>
<tr>
<td>10: return NULL; }</td>
</tr>
<tr>
<td>11: t1 = val1;</td>
</tr>
<tr>
<td>12: unlock(m1);</td>
</tr>
<tr>
<td>13: lock(m2);</td>
</tr>
<tr>
<td>14: t2 = val2;</td>
</tr>
<tr>
<td>15: unlock(m2);</td>
</tr>
<tr>
<td>16: assert(t2==val1+1);</td>
</tr>
</tbody>
</table>

**program counter: 3**

**mutexes:** $m1=0$; $m2=0$;
**global variables:** val1=1; val2=0;
**local variables:** t1= -1; t2= -1;
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7
val1-access: $W_{\text{twoStage},2}$
val2-access:

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));

**Program counter:** 7

**Mutexes:** m1=1; m2=0;
**Global variables:** val1=1; val2=0;
**Local variables:** t1= -1; t2= -1;
Lazy exploration: interleaving I.

statements: 1-2-3-7-8
val1-access: \( W_{\text{twoStage},2} \) - \( R_{\text{reader},8} \)
val2-access:

**Thread twoStage**
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

**Thread reader**
7: lock(m1);
8: if (val1 == 0) {
9:  unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));

**Program counter: 8**
mutexes: m1=1; m2=0;
global variables: val1=1; val2=0;
local variables: t1= -1; t2= -1;
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11
val1-access: $W_{twoStage,2} - R_{reader,8} - R_{reader,11}$
val2-access:

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:    unlock(m1);
10:    return NULL;
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2 == (t1 + 1));

program counter: 11
mutexes: m1=1; m2=0;
global variables: val1=1; val2=0;
local variables: $t1=1; t2=-1$;
Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12
val1-access: \( W_{\text{twoStage},2} - R_{\text{reader},8} - R_{\text{reader},11} \)
val2-access:

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));

Program counter: 12
mutexes: m1=0; m2=0;
global variables: val1=1; val2=0;
local variables: t1= 1; t2= -1;
Lazy exploration: interleaving \( I_s \)

statements: 1-2-3-7-8-11-12

val1-access: \( W_{\text{twoStage}, 2} - R_{\text{reader}, 8} - R_{\text{reader}, 11} \)

val2-access:

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:   unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2 == (t1 + 1));

**program counter:** 4

**mutexes:** m1=0; m2=0;

**global variables:** val1=1; val2=0;

**local variables:** t1 = 1; t2 = -1;
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11-12-4
val1-access: $W_{\text{twoStage}_2}$ - $R_{\text{reader}_8}$ - $R_{\text{reader}_{11}}$
val2-access:

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));

program counter: 4
mutexes: $m1=0$; $m2=1$
global variables: val1=1; val2=0;
local variables: t1 = 1; t2 = -1;
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5

val1-access: $W_{twoStage,2} - R_{reader,8} - R_{reader,11} - R_{twoStage,5}$

val2-access: $W_{twoStage,5}$

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:   unlock(m1);
10:  return NULL;
11:  t1 = val1;
12:  unlock(m1);
13:  lock(m2);
14:  t2 = val2;
15:  unlock(m2);
16:  assert(t2===(t1+1));

program counter: 5
mutexes: m1=0; m2=1;
global variables: val1=1; val2=2;
local variables: t1= 1; t2= -1;
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6
val1-access: $W_{\text{twoStage},2}$ - $R_{\text{reader},8}$ - $R_{\text{reader},11}$ - $R_{\text{twoStage},5}$
val2-access: $W_{\text{twoStage},5}$

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
  9:   unlock(m1);
10:  return NULL;
}
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));

**program counter: 6**
mutexes: $m1=0$; $m2=0$;
global variables: val1=1; val2=2;
local variables: t1= 1; t2= -1;
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6
val1-access: $W_{\text{twoStage},2} - R_{\text{reader},8} - R_{\text{reader},11} - R_{\text{twoStage},5}$
val2-access: $W_{\text{twoStage},5}$

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:    unlock(m1);
10:   return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));

program counter: 13
mutexes: m1=0; m2=0;
global variables: val1=1; val2=2;
local variables: t1= 1; t2= -1;
Lazy exploration: interleaving \( I_s \)

statements: 1-2-3-7-8-11-12-4-5-6-13

val1-access: \( W_{\text{twoStage},2} \) - \( R_{\text{reader},8} \) - \( R_{\text{reader},11} \) - \( R_{\text{twoStage},5} \)

val2-access: \( W_{\text{twoStage},5} \)

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));

program counter: 13
mutexes: \( m1=0; m2=1; \)
global variables: val1=1; val2=2;
local variables: t1= 1; t2= -1;
Lazy exploration: interleaving \( I_s \)

Statements: 1-2-3-7-8-11-12-4-5-6-13-14

Val1-access: \( W_{\text{twoStage},2} - R_{\text{reader},8} - R_{\text{reader},11} - R_{\text{twoStage},5} \)

Val2-access: \( W_{\text{twoStage},5} - R_{\text{reader},14} \)

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));

Program counter: 14

Mutexes: \( m1=0; m2=1; \)
Global variables: \( \text{val1}=1; \text{val2}=2; \)
Local variables: \( t1=1; t2=2; \)
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15
val1-access: $W_{\text{twoStage},2} - R_{\text{reader},8} - R_{\text{reader},11} - R_{\text{twoStage},5}$
val2-access: $W_{\text{twoStage},5} - R_{\text{reader},14}$

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2 == (t1+1));

program counter: 15
mutexes: $m1=0; m2=0$;
global variables: val1=1; val2=2;
local variables: t1 = 1; t2 = 2;
Lazy exploration: interleaving \( I_s \)

**Statements:** 1-2-3-7-8-11-12-4-5-6-13-14-15-16

**Val1-access:** \( W_{\text{twoStage,2}} \) - \( R_{\text{reader,8}} \) - \( R_{\text{reader,11}} \) - \( R_{\text{twoStage,5}} \)

**Val2-access:** \( W_{\text{twoStage,5}} \) - \( R_{\text{reader,14}} \)

**Thread twoStage**
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

**Thread reader**
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));

**Program counter:** 16
**Mutexes:** m1=0; m2=0;
**Global variables:** val1=1; val2=2;
**Local variables:** t1= 1; t2= 2;
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15-16

val1-access: $W_{\text{twoStage},2}$ - $R_{\text{reader},8}$ - $R_{\text{reader},11}$ - $R_{\text{twoStage},5}$

val2-access: $W_{\text{twoStage},5}$ - $R_{\text{reader},14}$

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:   unlock(m1);
10:  return NULL;
}
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2 == (t1 + 1));

QF formula is unsatisfiable, i.e., assertion holds
Lazy exploration: interleaving $I_f$

statements:

val1-access:

val2-access:

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:   unlock(m1);
10:  return NULL;
11:  t1 = val1;
12:  unlock(m1);
13:  lock(m2);
14:  t2 = val2;
15:  unlock(m2);
16:  assert(t2===(t1+1));

program counter: 0
mutexes: m1=0; m2=0;
global variables: val1=0; val2=0;
local variables: t1= -1; t2= -1;
Lazy exploration: interleaving $I_f$

statements: 1-2-3
val1-access: $W_{\text{twoStage},2}$
val2-access:

```
Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

```
Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:   unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

program counter: 3
mutexes: $m1=0; m2=0$
global variables: $\textbf{val1}=1; \text{val2}=0$
local variables: $t1= -1; t2= -1$
Lazy exploration: interleaving $I_f$

statements: 1-2-3

val1-access: $W_{twoStage,2}$

val2-access:

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:   unlock(m1);
10:  return NULL;
11:  t1 = val1;
12:  unlock(m1);
13:  lock(m2);
14:  t2 = val2;
15:  unlock(m2);
16:  assert(t2==(t1+1));

program counter: 7
mutexes: m1=0; m2=0;
global variables: val1=1; val2=0;
local variables: t1= -1; t2= -1;
Lazy exploration: interleaving $I_f$

statements: 1-2-3-7-8-11-12-13-14-15-16
val1-access: $W_{\text{twoStage},2}$ - $R_{\text{reader},8}$ - $R_{\text{reader},11}$
val2-access: $R_{\text{reader},14}$

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
  9:   unlock(m1);
  10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));

**program counter: 16**
mutexes: $m1=0$; $m2=0$;
global variables: val1=1; val2=0;
local variables: $t1= 1$; $t2= 0$;
Lazy exploration: interleaving $I_f$

statements: 1-2-3-7-8-11-12-13-14-15-16
val1-access: $W_{\text{twoStage}, 2} - R_{\text{reader}, 8} - R_{\text{reader}, 11}$
val2-access: $R_{\text{reader}, 14}$

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL;
} 11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2 == (t1 + 1));

**Program counter:** 4
**Mutexes:** m1 = 0; m2 = 0;
**Global variables:** val1 = 1; val2 = 0;
**Local variables:** t1 = 1; t2 = 0;
Lazy exploration: interleaving $I_f$

statements: 1-2-3-7-8-11-12-13-14-15-16-4-5-6

val1-access: $W_{\text{twoStage},2} - R_{\text{reader},8} - R_{\text{reader},11} - R_{\text{twoStage},5}$

val2-access: $R_{\text{reader},14} - W_{\text{twoStage},5}$

**Thread twoStage**
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

**Thread reader**
7: lock(m1);
8: if (val1 == 0) {
9:   unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));

**program counter:** 6
mutexes: $m1 = 0; m2 = 0$
**global variables:** val1 = 1; **val2 = 2**
**local variables:** t1 = 1; t2 = 0;
Lazy exploration: interleaving $I_f$

statements: 1-2-3-7-8-11-12-13-14-15-16-4-5-6

val1-access: $W_{\text{twoStage},2} - R_{\text{reader},8} - R_{\text{reader},11} - R_{\text{twoStage},5}$

val2-access: $R_{\text{reader},14} - W_{\text{twoStage},5}$

Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9:  unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));

QF formula is satisfiable, i.e., assertion does not hold
Lazy Approach: State Transitions

Initial state: $v_0: t_{\text{main}}, 0, \text{val}1=0, \text{val}2=0, m1=0, m2=0, ...$

Active thread, context bound: $v_0: t_{\text{main}}, 0, \text{val}1=0, \text{val}2=0, m1=0, m2=0, ...$

Global and local variables:

- $\text{val}1=0, \text{val}2=0$
- $m1=1, m2=0, ...$

Execution paths:

- $v_1: t_{\text{twoStage}}, 1, \text{val}1=0, \text{val}2=0, m1=1, m2=0, ...$
- $v_2: t_{\text{twoStage}}, 2, \text{val}1=1, \text{val}2=0, m1=1, m2=0, ...$
- $v_3: t_{\text{reader}}, 2, \text{val}1=0, \text{val}2=0, m1=1, m2=0, ...$
- $v_4: t_{\text{reader}}, 1, \text{val}1=0, \text{val}2=0, m1=1, m2=0, ...$
- $v_5: t_{\text{twoStage}}, 2, \text{val}1=0, \text{val}2=0, m1=1, m2=0, ...$
- $v_6: t_{\text{reader}}, 2, \text{val}1=0, \text{val}2=0, m1=1, m2=0, ...$

Blocked execution paths (eliminate): 

- $v_1, v_2, v_4$
- $v_3, v_5, v_6$
Exploring the Reachability Tree

• use a reachability tree (RT) to describe reachable states of a multi-threaded program

• each node in the RT is a tuple \( v = \left( A_i, C_i, s_i, \left< l_i^j, G_i^j \right>_{j=1}^n \right) \) for a given time step \( i \), where:
  – \( A_i \) represents the currently active thread
  – \( C_i \) represents the context switch number
  – \( s_i \) represents the current state
  – \( l_i^j \) represents the current location of thread \( j \)
  – \( G_i^j \) represents the control flow guards accumulated in thread \( j \) along the path from \( l_0^j \) to \( l_i^j \)

• expand the RT by executing symbolically each instruction of the multi-threaded program
Expansion Rules of the RT

**R1 (assign):** If \( l \) is an assignment, we execute \( l \), which generates \( s_{i+1} \). We add as child to \( \nu \) a new node \( \nu' \)

\[
\nu' = (A_i, C_i, s_{i+1}, \langle l_{i+1}^j, G_i^j \rangle)_{i+1} \quad \rightarrow \quad l_{i+1}^{A_i} = l_i^{A_i} + 1
\]

- we have fully expanded \( \nu \) if
  - \( l \) within an atomic block; or
  - \( l \) contains no global variable; or
  - the upper bound of context switches (\( C_i = C \)) is reached
- if \( \nu \) is not fully expanded, for each thread \( j \neq A_i \) where \( G_i^j \) is enabled in \( s_{i+1} \), we thus create a new child node

\[
\nu_j' = (j, C_i + 1, s_{i+1}, \langle l_{i+1}^j, G_i^j \rangle)_{i+1}
\]
Expansion Rules of the RT

R2 (skip): If \( l \) is a skip-statement with target \( l \), we increment the location of the current thread and continue with it. We explore no context switches:

\[
v' = (A_i, C_i, s_i, \langle l_{i+1}^j, G_i^j \rangle)_{i+1} \\
l_{i+1}^j = \begin{cases} 
  l_i^j + 1 & : j = A_i \\
  l_i^j & : otherwise
\end{cases}
\]

R3 (unconditional goto): If \( l \) is an unconditional goto-statement with target \( l \), we set the location of the current thread and continue with it. We explore no context switches:

\[
v' = (A_i, C_i, s_i, \langle l_{i+1}^j, G_i^j \rangle)_{i+1} \\
l_{i+1}^j = \begin{cases} 
  l & : j = A_i \\
  l_i^j & : otherwise
\end{cases}
\]
Expansion Rules of the RT

R4 (conditional goto): If \( l \) is a conditional goto-statement with test \( c \) and target \( l \), we create two child nodes \( \nu' \) and \( \nu'' \).

- for \( \nu' \), we assume that \( c \) is true and proceed with the target instruction of the jump:

\[
\nu' = (A_i, C_i, s_i, \langle l^j \rangle_{i+1})
\]

\[
l^{j}_{i+1} = \begin{cases} 
  l & : \ j = A_i \\
  l^j_i & : \ otherwise
\end{cases}
\]

- for \( \nu'' \), we add \( \neg c \) to the guards and continue with the next instruction in the current thread:

\[
\nu'' = (A_i, C_i, s_i, \langle l^{j\neg} \rangle_{i+1})
\]

\[
l^{j}_{i+1} = \begin{cases} 
  l^{j\neg}_i + 1 & : \ j = A_i \\
  l^j_i & : \ otherwise
\end{cases}
\]

- prune one of the nodes if the condition is determined statically
Expansion Rules of the RT

R5 (assume): If \( l \) is an \textit{assume}-statement with argument \( c \), we proceed similar to R1.

- we continue with the unchanged state \( s_i \) but add \( c \) to all guards, as described in R4
- If \( c \land G^j_i \) evaluates to \textit{false}, we prune the execution path

R6 (assert): If \( l \) is an \textit{assert}-statement with argument \( c \), we proceed similar to R1.

- we continue with the unchanged state \( s_i \) but add \( c \) to all guards, as described in R4
- we generate a verification condition to check the validity of \( c \)
Expansion Rules of the RT

**R5 (start_thread):** If \( l \) is a `start_thread` instruction, we add the indicated thread to the set of active threads:

\[
v' = \left( A_i, C_i, s_i, \langle l_{i+1}^j, G_{i+1}^j \rangle_{j=1}^{n+1} \right)_{i+1}
\]

- where \( l_{i+1}^{n+1} \) is the initial location of the thread and \( G_{i+1}^{n+1} = G_{i}^{A_i} \)
- the thread starts with the guards of the currently active thread

**R6 (join_thread):** If \( l \) is a `join_thread` instruction with argument \( ld \), we add a child node:

\[
v' = \left( A_i, C_i, s_i, \langle l_{i+1}^j, G_i^j \rangle \right)_{i+1}
\]

- where \( l_{i+1}^j = l_i^{A_i} + 1 \) only if the joining thread \( ld \) has exited
Observations about the lazy approach

• naïve but useful:
  – bugs usually manifest with few context switches [Qadeer&Rehof’05]
  – keep in memory the parent nodes of all unexplored paths only
  – exploit which transitions are enabled in a given state
  – bound the number of preemptions (C) allowed per threads
    ▶ *number of executions*: $O(n^c)$
  – as each formula corresponds to one possible path only, its size is relatively small

• can suffer performance degradation:
  – in particular for correct programs where we need to invoke the SMT solver once for each possible execution path
Schedule Recording

Idea: systematically encode all possible interleavings into one formula

- add a **fresh variable** \((ts)\) for each context switch block \((i)\) so that \(0 < ts_i \leq\) number of threads
  - record in which order the *scheduler* has executed the program (*aka scheduler guards*)
  - SMT solver determines the order in which threads are simulated

- add scheduler guards only to **effective statements** (assignments and assertions)
  - record **effective context switches (ECS)**
    - *context switches to an effective statement*
  - **ECS block**: sequence of program statements that are executed with no intervening ECS
Schedule Recording: Execution Paths

![Diagram showing execution paths for a two-stage reader algorithm with conditions and actions for lock and unlock operations.](Diagram.png)
Schedule Recording: Execution Paths

SMT solver instantiates ts to evaluate all possible paths

twoStage, reader

thread identifiers

program statement

twoStage, reader

twoStage, reader

twoStage, reader

twoStage, reader

If the guard of the parent node is false then the guard of the child node is false as well.
Schedule Recording: Interleaving \( I_s \)

statements:
twoStage-ECS:
reader-ECS:

Thread twoStage

1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

ECS block: sequence of program statements that are executed with no intervening ECS

10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
### Schedule Recording: Interleaving $I_s$

statements: 1

twoStage-ECS: $t_{s_{1,1}}$

reader-ECS:

<table>
<thead>
<tr>
<th>Thread twoStage</th>
<th>Thread reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: lock(m1);</td>
<td>7: lock(m1);</td>
</tr>
<tr>
<td>$t_{s_{1}} == 1$</td>
<td>8: unlock(m1);</td>
</tr>
<tr>
<td>2: val1 = 1;</td>
<td>9: lock(m1);</td>
</tr>
<tr>
<td>3: unlock(m1);</td>
<td>10: unlock(m1);</td>
</tr>
<tr>
<td>4: lock(m2);</td>
<td>11: return NULL;</td>
</tr>
<tr>
<td>5: val2 = val1 + 1;</td>
<td>12: unlock(m2);</td>
</tr>
<tr>
<td>6: unlock(m2);</td>
<td>13: lock(m2);</td>
</tr>
<tr>
<td>14: t2 = val2;</td>
<td>15: unlock(m2);</td>
</tr>
<tr>
<td>16: assert(t2===(t1+1));</td>
<td></td>
</tr>
</tbody>
</table>

- each program statement is then prefixed by a schedule guard $ts_i = j$, where:
  - $i$ is the ECS block number
  - $j$ is the thread identifier
Schedule Recording: Interleaving $I_s$

statements: 1-2

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$

reader-ECS:

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); 4: lock(m2); 5: val2 = val1 + 1; 6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3

twoStage-ECS: $ts_{1,1}-ts_{2,2}-ts_{3,3}$

reader-ECS:

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving I_s

statements: 1-2-3

twoStage-ECS: ts_{1,1}-ts_{2,2}-ts_{3,3}

reader-ECS:

Thread twoStage
1: lock(m1); \quad ts_1 == 1
2: val1 = 1; \quad ts_2 == 1
3: unlock(m1); \quad ts_3 == 1
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1);
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m1);
14: t2 = val2;
15: unlock(m2);
16: assert(t2 == (t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS: $ts_{7,4}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving \( I_s \)

statements: 1-2-3-7-8

twoStage-ECS: \( ts_{1,1} - ts_{2,2} - ts_{3,3} \)

reader-ECS: \( ts_{7,4} - ts_{8,5} \)

Thread twoStage
1: lock(m1); \( ts_1 == 1 \)
2: val1 = 1; \( ts_2 == 1 \)
3: unlock(m1); \( ts_3 == 1 \)
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1); \( ts_4 == 2 \)
8: if (val1 == 0) { \( ts_5 == 2 \)
9: unlock(m1);
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m1);
14: t2 = val2;
15: unlock(m1);
16: assert(t2==(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS: $ts_{7,4}$-$ts_{8,5}$-$ts_{11,6}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) {
9: unlock(m1); $ts_5 == 2$
10: return NULL; }
11: t1 = val1; $ts_6 == 2$
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS: $ts_{7,4}$-$ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1; $ts_6 == 2$
12: unlock(m1); $ts_7 == 2$
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$
reader-ECS: $ts_{7,4}$-$ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) { $ts_5 == 2$
9: unlock(m1);
10: return NULL; }
11: t1 = val1; $ts_6 == 2$
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12-4

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$

reader-ECS: $ts_{7,4}$-$ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2); $ts_8 == 1$
5: val2 = val1 + 1;
6: unlock(m2);

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL;
}
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5

twoStage-ECS: $ts_{1,1}-ts_{2,2}-ts_{3,3}-ts_{4,8}-ts_{5,9}$

reader-ECS: $ts_{7,4}-ts_{8,5}-ts_{11,6}-ts_{12,7}$

Thread twoStage
1: lock(m1); \hspace{1cm} ts_1 == 1
2: val1 = 1; \hspace{1cm} ts_2 == 1
3: unlock(m1); \hspace{1cm} ts_3 == 1
4: lock(m2); \hspace{1cm} ts_8 == 1
5: val2 = val1 + 1; \hspace{1cm} ts_9 == 1
6: unlock(m2);

Thread reader
7: lock(m1); \hspace{1cm} ts_4 == 2
8: if (val1 == 0) {
9: unlock(m1); \hspace{1cm} ts_5 == 2
10: return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2 == (t1 + 1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$

reader-ECS: $ts_{7,4}$-$ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2); $ts_8 == 1$
5: val2 = val1 + 1; $ts_9 == 1$
6: unlock(m2); $ts_{10} == 1$

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) { $ts_5 == 2$
9: unlock(m1);
10: return NULL; }
11: t1 = val1; $ts_6 == 2$
12: unlock(m1); $ts_7 == 2$
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6

twoStage-ECS: $ts_{1,1}-ts_{2,2}-ts_{3,3}-ts_{4,8}-ts_{5,9}-ts_{6,10}$

reader-ECS: $ts_{7,4}-ts_{8,5}-ts_{11,6}-ts_{12,7}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2); $ts_8 == 1$
5: val2 = val1 + 1; $ts_9 == 1$
6: unlock(m2); $ts_{10} == 1$

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1; $ts_6 == 2$
12: unlock(m1); $ts_7 == 2$
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6-13

twoStage-ECS: $\texttt{ts}_{1,1}$-$\texttt{ts}_{2,2}$-$\texttt{ts}_{3,3}$-$\texttt{ts}_{4,8}$-$\texttt{ts}_{5,9}$-$\texttt{ts}_{6,10}$

reader-ECS: $\texttt{ts}_{7,4}$-$\texttt{ts}_{8,5}$-$\texttt{ts}_{11,6}$-$\texttt{ts}_{12,7}$-$\texttt{ts}_{13,11}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2); $ts_8 == 1$
5: val2 = val1 + 1; $ts_9 == 1$
6: unlock(m2); $ts_{10} == 1$

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL;
}
11: t1 = val1; $ts_6 == 2$
12: unlock(m1); $ts_7 == 2$
13: lock(m2); $ts_{11} == 2$
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6-13-14

twoStage-ECS: $ts_{1,1}-ts_{2,2}-ts_{3,3}-ts_{4,8}-ts_{5,9}-ts_{6,10}$

reader-ECS: $ts_{7,4}-ts_{8,5}-ts_{11,6}-ts_{12,7}-ts_{13,11}-ts_{14,12}$

Thread twoStage
1: lock(m1); $ts_1 = 1$
2: val1 = 1; $ts_2 = 1$
3: unlock(m1); $ts_3 = 1$
4: lock(m2); $ts_8 = 1$
5: val2 = val1 + 1; $ts_9 = 1$
6: unlock(m2); $ts_{10} = 1$

Thread reader
7: lock(m1); $ts_4 = 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1; $ts_6 = 2$
12: unlock(m1); $ts_7 = 2$
13: lock(m2); $ts_{11} = 2$
14: t2 = val2; $ts_{12} = 2$
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving \( I_s \)

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15

twoStage-ECS: \( ts_{1,1} - ts_{2,2} - ts_{3,3} - ts_{4,8} - ts_{5,9} - ts_{6,10} \)
reader-ECS: \( ts_{7,4} - ts_{8,5} - ts_{11,6} - ts_{12,7} - ts_{13,11} - ts_{14,12} - ts_{15,13} \)
Schedule Recording: Interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15-16

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$

reader-ECS: $ts_{7,4}$-$ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$-$ts_{13,11}$-$ts_{14,12}$-$ts_{15,13}$-$ts_{16,14}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2); $ts_8 == 1$
5: val2 = val1 + 1; $ts_9 == 1$
6: unlock(m2); $ts_{10} == 1$

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL;
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1)); $ts_{14} == 2$

Thread values:
- $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$
- $ts_{7,4}$-$ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$-$ts_{13,11}$-$ts_{14,12}$-$ts_{15,13}$-$ts_{16,14}$
Schedule Recording: Interleaving $I_f$

statements: 1-2-3-7-8-11-12-13-14-15-16-4-5-6

twoStage-ECS: $ts_{1,1}-ts_{2,3}-ts_{3,4}-ts_{4,12}-ts_{5,13}-ts_{6,14}$

reader-ECS: $ts_{7,4}-ts_{8,5}-ts_{11,6}-ts_{12,7}-ts_{13,8}-ts_{14,9}-ts_{15,10}-ts_{16,11}$

Thread twoStage
1: lock(m1); $ts_1 == 1$
2: val1 = 1; $ts_2 == 1$
3: unlock(m1); $ts_3 == 1$
4: lock(m2); $ts_{12} == 1$
5: val2 = val1 + 1; $ts_{13} == 1$
6: unlock(m2); $ts_{14} == 1$

Thread reader
7: lock(m1); $ts_4 == 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1; $ts_6 == 2$
12: unlock(m1); $ts_7 == 2$
13: lock(m2); $ts_8 == 2$
14: t2 = val2; $ts_9 == 2$
15: unlock(m2); $ts_{10} == 2$
16: assert(t2==(t1+1)); $ts_{11} == 2$
Observations about the schedule recoding approach

- we systematically explore the thread interleavings as before, but now:
  - add schedule guards to record in which order the scheduler has executed the program
  - encode all execution paths into one formula
    - bound the number of preemptions
    - exploit which transitions are enabled in a given state
- the number of threads and context switches can grow very large quickly, and easily “blow-up” the solver:
  - there is a clear trade-off between usage of time and memory resources
Under-approximation and Widening

Idea: check models with an increased set of allowed interleavings [Grumberg&et al.’05]

- start from a single interleaving (under-approximation) and widen the model by adding more interleavings incrementally
- main steps of the algorithm:
  1. encode control literals \((c_{l_{i,j}})\) into the verification condition \(\psi\)
     - \(c_{l_{i,j}}\) where \(i\) is the ECS block number and \(j\) is the thread identifier
  2. check the satisfiability of \(\psi\) (stop if \(\psi\) is satisfiable)
  3. extract proof objects generated by the SMT solver
  4. check whether the proof depends on the control literals (stop if the proof does not depend on the control literals)
  5. remove literals that participated in the proof and go to step 2
UW Approach: Running Example

• use the same guards as in the schedule recording approach as control literals
  – but here the schedule is updated based on the information extracted from the proof

<table>
<thead>
<tr>
<th>Thread twoStage</th>
<th>cl_{i,twoStage} → ts_{i,twoStage} == 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: lock(m1);</td>
<td>cl_{1,twoStage} → ts_{1,twoStage} == 1</td>
</tr>
<tr>
<td>2: val1 = 1;</td>
<td>cl_{2,twoStage} → ts_{2,twoStage} == 1</td>
</tr>
<tr>
<td>3: unlock(m1);</td>
<td>cl_{3,twoStage} → ts_{3,twoStage} == 1</td>
</tr>
<tr>
<td>4: lock(m2);</td>
<td>cl_{8,twoStage} → ts_{8,twoStage} == 1</td>
</tr>
<tr>
<td>5: val2 = val1 + 1;</td>
<td>cl_{9,twoStage} → ts_{9,twoStage} == 1</td>
</tr>
<tr>
<td>6: unlock(m2);</td>
<td>cl_{10,twoStage} → ts_{10,twoStage} == 1</td>
</tr>
</tbody>
</table>

• reduce the number of control points from \( m \times n \) to \( e \times n \)
  – \( m \) is the number of program statements; \( n \) is the number of threads, and \( e \) is the number of ECS blocks
Evaluation
Comparison of the Approaches

- Goal: compare efficiency of the proposed approaches
  - lazy exploration
  - schedule recording
  - underapproximation and widening

- Set-up:
  - ESBMC v1.15.1 together with the SMT solver Z3 v2.11
  - support the logics $QF\_AUFBV$ and $QF\_AUFLIRA$
  - standard desktop PC, time-out 3600 seconds
### About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#T</th>
<th>#P</th>
<th>B</th>
<th>#C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data compressor</td>
<td>8568</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>9695</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>1233</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>6366</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reorder_bad</td>
<td>84</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>twostage_bad</td>
<td>128</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>110</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>1060</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>425</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>1109</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>1171</td>
<td>29</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Lines of code**: 8568
- **Number of context switches**: 9695
- **Number of properties checked**: 1233
- **Number of threads**: 6366
- **The number of BMC unrolling steps**: 1233

- **Contains a data race**: 84
- **Contains an atomicity violation**: 128
- **Contains wrong lock acquisition ordering**: 110
- **Configures the HDMI device**: 1060
- **Front panel LED display**: 425
- **Demonstrate how thumbnail images can be manipulated**: 1109
- **Synthetic micro-benchmark**: 1171
# About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>#threads</th>
<th>#bytes</th>
<th>#calls</th>
<th>#cycles</th>
<th>C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_ok</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Frangipani file system</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Frangipani file system with array out of bounds</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>77</td>
<td>13</td>
<td>21</td>
<td>129</td>
<td>4</td>
<td>Insert messages into a hash table concurrently</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>1233</td>
<td>3</td>
<td>279</td>
<td>200</td>
<td>2</td>
<td>Multi-threaded download accelerator</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>6366</td>
<td>3</td>
<td>8568</td>
<td>1</td>
<td>9</td>
<td>Data compressor</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>84</td>
<td>10</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>Contains a data race</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>128</td>
<td>100</td>
<td>13</td>
<td>100</td>
<td>4</td>
<td>Contains an atomicity violation</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>110</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>Contains wrong lock acquisition ordering</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>1060</td>
<td>2</td>
<td>24</td>
<td>16</td>
<td>20</td>
<td>Configures the HDMI device</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>425</td>
<td>2</td>
<td>45</td>
<td>10</td>
<td>10</td>
<td>Front panel LED display</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>1109</td>
<td>2</td>
<td>249</td>
<td>2</td>
<td>1</td>
<td>Demonstrate how thumbnail images can be manipulated</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>1171</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>17</td>
<td>Synthetic micro-benchmark</td>
</tr>
</tbody>
</table>
## About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#T</th>
<th>#P</th>
<th>B</th>
<th>#C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_ok</td>
<td>81</td>
<td>26</td>
<td>47</td>
<td>26</td>
<td>2</td>
<td>Frangipani file system</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>80</td>
<td>27</td>
<td>48</td>
<td>27</td>
<td>2</td>
<td>Frangipani file system with array out of bounds</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>89</td>
<td>27</td>
<td>48</td>
<td>27</td>
<td>2</td>
<td>Insert messages into a hash table concurrently</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Multi-threaded download accelerator</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>63</td>
<td>3</td>
<td>8568</td>
<td>1</td>
<td>9</td>
<td>Data compressor</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>84</td>
<td>10</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>Contains a data race</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>128</td>
<td>100</td>
<td>13</td>
<td>100</td>
<td>4</td>
<td>Contains an atomicity violation</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>110</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>Contains wrong lock acquisition ordering</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>1060</td>
<td>2</td>
<td>24</td>
<td>16</td>
<td>20</td>
<td>Configures the HDMI device</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>425</td>
<td>2</td>
<td>45</td>
<td>10</td>
<td>10</td>
<td>Front panel LED display</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>1109</td>
<td>2</td>
<td>249</td>
<td>2</td>
<td>1</td>
<td>Demonstrate how thumbnail images can be manipulated</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>1171</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>17</td>
<td>synthetic micro-benchmark</td>
</tr>
</tbody>
</table>
### About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#T</th>
<th>#P</th>
<th>B</th>
<th>#C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_ok</td>
<td>81</td>
<td>26</td>
<td>47</td>
<td>26</td>
<td>2</td>
<td>Frangipani file system</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>80</td>
<td>27</td>
<td>48</td>
<td>27</td>
<td>2</td>
<td>Frangipani file system with array out of bounds</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>77</td>
<td>13</td>
<td>21</td>
<td>129</td>
<td>4</td>
<td>Insert messages into a hash table concurrently</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>1233</td>
<td>3</td>
<td>279</td>
<td>200</td>
<td>2</td>
<td>Multi-threaded download accelerator</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data compressor</td>
</tr>
<tr>
<td>reorder_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contains a data race</td>
</tr>
<tr>
<td>twostage_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contains an atomicity violation</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contains wrong lock acquisition ordering</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>73</td>
<td>2</td>
<td>24</td>
<td>16</td>
<td>20</td>
<td>Configures the HDMI device</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>425</td>
<td>2</td>
<td>45</td>
<td>10</td>
<td>10</td>
<td>Front panel LED display</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>1109</td>
<td>2</td>
<td>249</td>
<td>2</td>
<td>1</td>
<td>Demonstrate how thumbnail images can be manipulated</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>1171</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>17</td>
<td>synthetic micro-benchmark</td>
</tr>
</tbody>
</table>
### About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#T</th>
<th>#P</th>
<th>B</th>
<th>#C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_ok</td>
<td>81</td>
<td>26</td>
<td>47</td>
<td>26</td>
<td>2</td>
<td>Frangipani file system</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>80</td>
<td>27</td>
<td>48</td>
<td>27</td>
<td>2</td>
<td>Frangipani file system with array out of bounds</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>77</td>
<td>13</td>
<td>21</td>
<td>129</td>
<td>4</td>
<td>Insert messages into a hash table concurrently</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>1233</td>
<td>3</td>
<td>279</td>
<td>200</td>
<td>2</td>
<td>Multi-threaded download accelerator</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>6366</td>
<td>3</td>
<td>8568</td>
<td>1</td>
<td>9</td>
<td>Data compressor</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>84</td>
<td>10</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>Contains a data race</td>
</tr>
<tr>
<td>twostage_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contains an atomicity violation</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Wrong lock acquisition</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Configures the HDMI device</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Front panel LED display</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>1109</td>
<td>2</td>
<td>249</td>
<td>2</td>
<td>1</td>
<td>Demonstrate how thumbnail images can be manipulated</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>1171</td>
<td>10</td>
<td>10</td>
<td>17</td>
<td></td>
<td>Synthetic micro-benchmark</td>
</tr>
</tbody>
</table>

It is used to check the scalability of multi-threaded software verification tools [Ghafari 2010]
Comparison of the approaches

<table>
<thead>
<tr>
<th>Module</th>
<th>Time</th>
<th>Result</th>
<th>#FI/#I</th>
<th>Time</th>
<th>Result</th>
<th>Time</th>
<th>Result</th>
<th>Iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_ok</td>
<td>1800</td>
<td>+</td>
<td>0/1294</td>
<td>MO</td>
<td>- MO</td>
<td>-</td>
<td>MO</td>
<td>1</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>&lt;1</td>
<td>+</td>
<td>1/154574</td>
<td>MO</td>
<td>- MO</td>
<td>-</td>
<td>MO</td>
<td>1</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>88</td>
<td>+</td>
<td>1/139</td>
<td>93</td>
<td>+ 195</td>
<td>+</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>indexer_bad</td>
<td>90</td>
<td>+</td>
<td>6/104015</td>
<td>MO</td>
<td>- MO</td>
<td>-</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>twostage_bad</td>
<td>229</td>
<td>+</td>
<td>0/1</td>
<td>226</td>
<td>+ 213</td>
<td>+</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>73</td>
<td>+</td>
<td>0/11</td>
<td>73</td>
<td>+ 787</td>
<td>+</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>micro_10_bad</td>
<td>95</td>
<td>+</td>
<td>3/3</td>
<td>14</td>
<td>+ 12</td>
<td>+</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Encoding and solver time**

**Error detected in module “+” GOOD THING**

**Error occurred in tool “–” BAD THING**

**Number of generated and failed interleavings**

**Number of iterations**
Comparison of the approaches (1)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Lazy</th>
<th>Schedule</th>
<th>UW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Result</td>
<td>#FI/#I</td>
</tr>
<tr>
<td>fxbench_k</td>
<td>282</td>
<td>+</td>
<td>0/676</td>
</tr>
<tr>
<td>fxbench_bad</td>
<td>&lt;1</td>
<td>+</td>
<td>729/729</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>595</td>
<td>+</td>
<td>0/17160</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>137</td>
<td>+</td>
<td>1/1</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>1800</td>
<td>+</td>
<td>0/1294</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>&lt;1</td>
<td>+</td>
<td>1/154574</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>88</td>
<td>+</td>
<td>1/139</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>90</td>
<td>+</td>
<td>6/104015</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>229</td>
<td>+</td>
<td>0/1</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>73</td>
<td>+</td>
<td>0/11</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>95</td>
<td>+</td>
<td>3/3</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>254</td>
<td>+</td>
<td>0/29260</td>
</tr>
</tbody>
</table>

*lazy encoding often more efficient than schedule recording and UW*
Comparison of the approaches (2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Lazy Result</th>
<th>#FI/#I</th>
<th>Schedule Result</th>
<th>Schedule Time</th>
<th>UW Result</th>
<th>UW Time</th>
<th>Iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lazy Schedule UW</td>
<td>+</td>
<td>0/676</td>
<td>+</td>
<td>304</td>
<td>+</td>
<td>301</td>
<td>+</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>&lt;1</td>
<td>729/729</td>
<td>+</td>
<td>360</td>
<td>+</td>
<td>786</td>
<td>+</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>595</td>
<td>0/17160</td>
<td>+</td>
<td>220</td>
<td>+</td>
<td>218</td>
<td>+</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>137</td>
<td>1/1</td>
<td>+</td>
<td>127</td>
<td>+</td>
<td>125</td>
<td>+</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>1800</td>
<td>0/1294</td>
<td>MO</td>
<td>-</td>
<td>MO</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>&lt;1</td>
<td>1/154574</td>
<td>MO</td>
<td>-</td>
<td>MO</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>88</td>
<td>1/139</td>
<td>+</td>
<td>93</td>
<td>+</td>
<td>195</td>
<td>+</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>90</td>
<td>6/104015</td>
<td>MO</td>
<td>-</td>
<td>MO</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>229</td>
<td>0/1</td>
<td>+</td>
<td>226</td>
<td>+</td>
<td>213</td>
<td>+</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>73</td>
<td>0/11</td>
<td>+</td>
<td>73</td>
<td>+</td>
<td>787</td>
<td>+</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>95</td>
<td>3/3</td>
<td>+</td>
<td>14</td>
<td>+</td>
<td>12</td>
<td>+</td>
</tr>
<tr>
<td>Lazy Schedule UW</td>
<td>254</td>
<td>0/29260</td>
<td>MO</td>
<td>-</td>
<td>MO</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>
### Evaluation of the approaches (3)

<table>
<thead>
<tr>
<th>Module</th>
<th>Lazy</th>
<th>Schedule</th>
<th>UW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>#FI/#I</td>
<td>Time</td>
</tr>
<tr>
<td>fsbench_ok</td>
<td>282  +</td>
<td>0/676</td>
<td>304  +</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>&lt;1   +</td>
<td>729/729</td>
<td>360  +</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>595  +</td>
<td>0/17160</td>
<td>220  +</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>137  +</td>
<td>1/1</td>
<td>127  +</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>1800 +</td>
<td>0/1294</td>
<td>MO   -</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>&lt;1   +</td>
<td>1/154574</td>
<td>MO   -</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>88   +</td>
<td>1/139</td>
<td>93   +</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>90   +</td>
<td>6/104015</td>
<td>MO   -</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>229  +</td>
<td>0/1</td>
<td>226  +</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>73   +</td>
<td>0/11</td>
<td>73   +</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>95   +</td>
<td>3/3</td>
<td>14   +</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>254  +</td>
<td>0/29260</td>
<td>MO   -</td>
</tr>
</tbody>
</table>

*lazy encoding is extremely fast for satisfiable instances*
Comparison to CHESS [Musuvathi and Qadeer]

- CHESS (v0.1.30626.0) is a concurrency testing tool for C# programs; also works for C/C++ (Windows API).
  - implements iterative context-bounding
  - requires unit tests that it repeatedly executes in a loop, exploring a different interleaving on each iteration
    - it is similar to our lazy approach
  - performs state hashing based on a happens-before graph
    - avoids exploring the same state repeatedly

- Goal: compare efficiency of the approaches
  - on identical verification problems taken from standard benchmark suites of multi-threaded software
Comparison to CHESS [Musuvathi and Qadeer]

CHESS is effective for programs where there are a small number of threads.

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>CHESS</th>
<th>Lazy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Tests</td>
</tr>
<tr>
<td>reorder_4_bad (3,1)</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>reorder_5_bad (4,1)</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>reorder_6_bad (5,1)</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>reorder_6_bad (5,1)</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>reorder_6_bad (5,1)</td>
<td>6</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>twostage_4_bad (3,1)</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>twostage_5_bad (4,1)</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>twostage_6_bad (5,1)</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>wronglock_4_bad (1,3)</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>wronglock_5_bad (1,4)</td>
<td>5</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>wronglock_6_bad (1,5)</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>micro_2_ok (100)</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>micro_2_ok (100)</td>
<td>2</td>
<td>1</td>
<td>17</td>
</tr>
</tbody>
</table>
CHESS is effective for programs where there are a small number of threads, **but it does not scale that well and consistently runs out of time when we increase the number of threads**.

<table>
<thead>
<tr>
<th>Module</th>
<th>Time</th>
<th>Tests</th>
<th>Time</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>reorder_5_bad (4,1)</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>TO</td>
</tr>
<tr>
<td>reorder_6_bad (5,1)</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>TO</td>
</tr>
<tr>
<td>reorder_6_bad (5,1)</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>TO</td>
</tr>
<tr>
<td>reorder_6_bad (5,1)</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>TO</td>
</tr>
<tr>
<td>twostage_4_bad (3,1)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>21</td>
</tr>
<tr>
<td>twostage_5_bad (4,1)</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>TO</td>
</tr>
<tr>
<td>twostage_6_bad (5,1)</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>TO</td>
</tr>
<tr>
<td>wronglock_4_bad (1,3)</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>21</td>
</tr>
<tr>
<td>wronglock_5_bad (1,4)</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>724</td>
</tr>
<tr>
<td>wronglock_6_bad (1,5)</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>TO</td>
</tr>
<tr>
<td>micro_2_ok (100)</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>316</td>
</tr>
<tr>
<td>micro_2_ok (100)</td>
<td>2</td>
<td>1</td>
<td>17</td>
<td>TO</td>
</tr>
</tbody>
</table>

**Note:** The table compares the performance of CHESS and a Lazy approach. The time is given in milliseconds. The 'TO' indicates that the time was exceeded. The #FI/#I column shows the ratio of false TEs to true TEs.
Comparison to SATABS [D. Kroening]

- SATABS (v2.5) implements predicate abstraction using SAT
  - avoids exponential number of theorem prover calls (for each potential assignment) to construct the Boolean program
  - uses BDD-based model checking (Cadence SMV) to verify the Boolean program
  - supports most ANSI-C constructs (incl. arithmetic overflow) and the verification of multi-threaded software with locks and shared variables

- Goal: compare efficiency of both approaches
  - on identical verification problems taken from standard benchmark suites of multi-threaded software
### Comparison to SATABS [D. Kroening]

<table>
<thead>
<tr>
<th>Test Case</th>
<th>SATABS</th>
<th>Lazy</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Result</td>
<td>Time</td>
<td>Result</td>
</tr>
<tr>
<td>fsbench_ok</td>
<td>†</td>
<td>-</td>
<td>282</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>†</td>
<td>-</td>
<td>&lt;1</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>TO</td>
<td>-</td>
<td>595</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>3346</td>
<td>+</td>
<td>137</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>TO</td>
<td>-</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>2</td>
<td>-</td>
<td>90</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>TO</td>
<td>-</td>
<td>229</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>RF</td>
<td>-</td>
<td>73</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>317</td>
<td>+</td>
<td>95</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>TO</td>
<td>-</td>
<td>254</td>
</tr>
</tbody>
</table>

- **failed to validate the counterexample**
- **failed to refine the predicate**
Comparison to SATABS [D. Kroening]

<table>
<thead>
<tr>
<th>Module</th>
<th>SATABS</th>
<th>Lazy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Result</td>
</tr>
<tr>
<td>fsbench_ok</td>
<td>†</td>
<td>-</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>†</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>3346</td>
<td>+</td>
</tr>
<tr>
<td>bzip2smp</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>RF</td>
<td>-</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>317</td>
<td>+</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>TO</td>
<td>-</td>
</tr>
</tbody>
</table>
SATABS uses predicate abstraction and refinement and tries to solve a harder problem than ESBMC

<table>
<thead>
<tr>
<th></th>
<th>Result</th>
<th>Time</th>
<th>Result</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_ok</td>
<td>†</td>
<td>-</td>
<td>282</td>
<td>+</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>†</td>
<td>-</td>
<td>&lt;1</td>
<td>+</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>TO</td>
<td>-</td>
<td>595</td>
<td>+</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>3346</td>
<td>+</td>
<td>137</td>
<td>+</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>TO</td>
<td>-</td>
<td>1800</td>
<td>+</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>1</td>
<td>-</td>
<td>&lt;1</td>
<td>+</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>2</td>
<td>-</td>
<td>88</td>
<td>+</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>2</td>
<td>-</td>
<td>90</td>
<td>+</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>TO</td>
<td>-</td>
<td>229</td>
<td>+</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>RF</td>
<td>-</td>
<td>73</td>
<td>+</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>317</td>
<td>+</td>
<td>95</td>
<td>+</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>TO</td>
<td>-</td>
<td>254</td>
<td>+</td>
</tr>
</tbody>
</table>
SATABS uses predicate abstraction and refinement and tries to solve a harder problem than ESBMC, **but this problem may still be too hard as SATABS is unable to prove the required properties**.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Time</th>
<th>Result</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_ok</td>
<td></td>
<td>-</td>
<td>282</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td></td>
<td>-</td>
<td>&lt;1</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>TO</td>
<td>-</td>
<td>595</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>3346</td>
<td>+</td>
<td>137</td>
</tr>
<tr>
<td>bzip2smp_ok</td>
<td>TO</td>
<td>-</td>
<td>1800</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>1</td>
<td>-</td>
<td>&lt;1</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>2</td>
<td>-</td>
<td>88</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>2</td>
<td>-</td>
<td>90</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>TO</td>
<td>-</td>
<td>229</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>RF</td>
<td>-</td>
<td>73</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>317</td>
<td>+</td>
<td>95</td>
</tr>
<tr>
<td>micro_10_ok</td>
<td>TO</td>
<td>-</td>
<td>254</td>
</tr>
</tbody>
</table>
Agenda

- SMT-based BMC for Embedded ANSI-C Software
- Verifying Multi-threaded Software
- Implementation of ESBMC

  - Integrating ESBMC into Software Engineering Practice

- Conclusions and Future Work
Continuous Verification

- based on Fowler’s **continuous integration (CI)**: build and test full system after each change
- complement testing by verification (SMT-based bounded model checking)
  - assertions
  - language-specific properties
- exploit existing information
  - development history (SCM)
  - test cases
- limit change propagation
  - equivalence checks
Functional Equivalence Checking

• determine whether modified functions need to be re-verified
  – no need to re-verify properties if functions are equivalent
  – **less expensive** than re-verifying the function
  – **undecidable** due to unbounded memory usage
Functional Equivalence Checking

• determine whether modified functions need to be re-verified
  – no need to re-verify properties if functions are equivalent
  – less expensive than re-verifying the function
  – undecidable due to unbounded memory usage

• goal: compare input-output relation

```cpp
unsigned Inv(int signal) {
    unsigned inverter;
    if (signal >= 0)
        inverter = signal;
    else
        inverter = -1*signal;
    return inverter;
}

unsigned Inv(int signal) {
    if (signal < 0)
        return -signal;
    else
        return signal;
}
```
Functional Equivalence Checking

• determine whether modified functions need to be re-verified
  – no need to re-verify properties if functions are equivalent
  – **less expensive** than re-verifying the function
  – **undecidable** due to unbounded memory usage

• goal: compare input-output relation
  – remove variables and returns

```c
unsigned Inv(int signal) {
  unsigned inverter;
  if (signal >= 0)
    inverter = signal;
  else
    inverter = -1*signal;
  return inverter;
}
```

```c
unsigned Inv(int signal) {
  if (signal < 0)
    return -signal;
  else
    return signal;
}
```
Functional Equivalence Checking

- determine whether modified functions need to be re-verified
  - no need to re-verify properties if functions are equivalent
  - less expensive than re-verifying the function
  - undecidable due to unbounded memory usage

- goal: compare input-output relation
  - remove variables and returns
  - convert the function bodies into SSA

\[
\alpha_1 = \begin{bmatrix}
  \text{inverter}_1 = \text{signal}_1 \\
\wedge \text{inverter}_2 = -1*\text{signal}_1 \\
\wedge \text{inverter}_3 = (\text{signal}_1 \geq 0? \text{inverter}_1 : \text{inverter}_2)
\end{bmatrix}
\]

\[
\alpha_2 = \begin{bmatrix}
  \text{signal}'_2 = (\text{signal}'_1 < 0? -\text{signal}'_1 : \text{signal}'_1)
\end{bmatrix}
\]

```c
unsigned Inv(int signal) {
  unsigned inverter;
  if (signal >= 0)
    inverter = signal;
  else
    inverter = -1*signal;
  return inverter;
}
```

```c
unsigned Inv(int signal) {
  if (signal < 0)
    return -signal;
  else
    return signal;
}
```
Functional Equivalence Checking

- determine whether modified functions need to be re-verified
  - no need to re-verify properties if functions are equivalent
  - less expensive than re-verifying the function
  - undecidable due to unbounded memory usage
- goal: compare input-output relation
  - remove variables and returns
  - convert the function bodies into SSA
  - show that the input and output variables coincide

\[
(\alpha_1 \land \alpha_2 \land (\text{signal}_1 = \text{signal}'_1)) \rightarrow (\text{inverter}_3 = \text{signal}'_2)
\]
Functional Equivalence Checking

• determine whether modified functions need to be re-verified
  – no need to re-verify properties if functions are equivalent
  – less expensive than re-verifying the function
  – undecidable due to unbounded memory usage

• goal: compare input-output relation
  – remove variables and returns
  – convert the function bodies into SSA
  – show that the input and output variables coincide

\[(\alpha_1 \land \alpha_2 \land (signal_1 = signal'_1)) \rightarrow (\text{inverter}_3 = signal'_2) \land (g_1 = g'_1)\]
Generalizing Test Cases

• use **existing test cases** to reduce the state space
  – run the unit tests, keep track of inputs
  – guide model checker to visit states not yet visited

• test stubs break the **global model** into **local models**
  – use test case as initial state
  – generate reachable states on-demand

⇒ reduces the number of paths and variables

```
a=nondet_int(); assume(a>10 && a<200);
```
Generalizing Test Cases: Example

Simple circular FIFO buffer:

```c
static char buffer[BUFFER_MAX];
void initLog(int max) {
    buffer_size = max;
    first = next = 0;
}

int removeLogElem(void) {
    first++;
    return buffer[first-1];
}

void insertLogElem(int b) {
    if (next < buffer_size) {
        buffer[next] = b;
        next = (next+1)%buffer_size;
    }
}
```

Test case:
check whether messages are added to and removed from the circular buffer

```c
static void testCircularBuffer(void) {
    int senData[] = {1, -128, 98, 88, 59,
                     1, -128, 90, 0, -37};
    int i;
    initLog(5);
    for(i=0; i<10; i++)
        insertLogElem(senData[i]);
    for(i=5; i<10; i++)
        ASSERT_EQUAL_INT(senData[i],
                         removeLogElem());
```
Generalizing Test Cases: Example

Simple circular FIFO buffer:

```c
static char buffer[BUFFER_MAX];
void initLog(int max) {
    buffer_size = max;
    first = next = 0;
}

int removeLogElem(void) {
    first++;
    return buffer[first-1];
}

void insertLogElem(int b) {
    if (next < buffer_size) {
        buffer[next] = b;
        next = (next+1)%buffer_size;
    }
}
```

BUT: implementation is flawed!

The array buffer is of type char[]

Assign an integer variable
Generalizing Test Cases: Example

Simple circular FIFO buffer:

```c
static char buffer[BUFFER_MAX];
void initLog(int max) {
    buffer_size = max;
    first = next = 0;
}

int removeLogElem(void) {
    first++;
    return buffer[first-1];
}

void insertLogElem(int b) {
    if (next < buffer_size) {
        buffer[next] = nondet_int();
        next = (next+1)%buffer_size;
    }
}
```

BUT: implementation is flawed!

```c
if (next < buffer_size) {
    buffer[next] = nondet_int();
    next = (next+1)%buffer_size;
}
```

The array buffer is of type char[]

Assign an integer variable

We can detect the error by assigning a non-deterministic value

This can lead to false results
Generalizing Test Cases: Example

Rather than modifying the program we modify the test stubs

```
static void testCircularBuffer(void) {
    int senData[] = {nondet_int(), ..., nondet_int()};

    assume(senData[0] <= 1 && senData[0] >= 42);
    assume(senData[1] <= -28 && senData[1] >= -128);
    ...
    int i;
    initLog(5);
    for(i=0; i<10; i++)
        insertLogElem(senData[i]);
    for(i=5; i<10; i++)
        ASSERT_EQUAL_INT(senData[i], removeLogElem());
}
```

→ detects two bugs related to arithmetic over- and underflow

Block larger parts of the search space (combine respective values into a single interval)

- force the model checker towards the “unobvious” errors
Specifying Temporal Properties

- we translate the LTL formulae into Büchi Automata (BA) and further into ANSI-C
  - monitor the design’s progress and watch out for violations
- we extract two properties of the pulse oximeter device:
  a) verify the data flow to compute the HR value that is provided by the sensor
  b) verify whether the user is able to adjust the sample time of the device
- the properties (a) and (b) can be expressed as:
Translation from BA to ANSI-C

\[ AG(p \rightarrow F r) \]

```c
void monitor_thread(void* arg)
{
    ... 
    while(1) {
        choice = nondet_bool();
        if (p) flag=true;
        switch (state) {
            case init:
                if (r || !p) state=s3;
                ... break;
            case s1:
                ... break;
                ... 
            }  
            if (flag && !is_processing) 
                assert(state == s3); 
        }
        pthread_exit(NULL);
    }
```
Monitor and Event Threads

```c
void monitor_thread(void* arg) {
    ...
    while(1) {
        choice = nondet_bool();
        if (p) flag=true;
        switch (state) {
            case init:
                if (r || !p) state=s3;
                ...
                break;
            case s1:
                ...
                break;
            ...
        }
        if (flag && !is_processing)
            assert(state == s3);
    }
    pthread_exit(NULL);
}
```

```c
model the hardware interrupt and interacts with the pulse oximeter

```c
void event_thread(void* arg) {
    while(1) {
        if (nondet_bool()) {
            is_processing = true;
            timer_interrupt(); //hardware interrupt
            is_processing = false;
        }
    }
    pthread_exit(NULL);
}
```

indicate whether a hardware interrupt has occurred
Concurrent Execution of Main, Monitor and event Threads

event_thread  main_thread  monitor_thread

- ECS block 0
- ECS block 1
- ECS block 2
- ECS block 3
- ECS block 4
- ECS block 5
- ECS block 6
- ECS block 7
- ECS block 8
- ECS block 9
Evaluation
Set-top Box Case Study

• Goal: evaluate the feasibility of the elements of the continuous verification approach
  – use of the unit tests and function equivalence checking

• embedded software used in a commercial product from NXP
  – high definition internet protocol and hybrid digital TV applications
  – Linux operating system (LinuxDVB, DirectFB and ALSA)

• Set-up:
  – ESBMC v1.15.1 together with the SMT solver Z3 v2.11
  – standard desktop PC, time-out 3600 seconds
## Verification of the Test Cases

<table>
<thead>
<tr>
<th>Test Program</th>
<th>L</th>
<th>B</th>
<th>P</th>
<th>VC</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandLoop.TC1</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC2</td>
<td>545</td>
<td>500*</td>
<td>18</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>commandLoop.TC3</td>
<td>545</td>
<td>500*</td>
<td>18</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>commandLoop.TC4</td>
<td>545</td>
<td>17</td>
<td>18</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>commandLoop.TC5</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC6</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC7</td>
<td>545</td>
<td>1</td>
<td>18</td>
<td>15</td>
<td>19</td>
</tr>
<tr>
<td>checkCommandParams.TC1</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>56</td>
<td>9</td>
</tr>
<tr>
<td>checkCommandParams.TC2</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>checkCommandParams.TC3</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>37</td>
<td>5</td>
</tr>
<tr>
<td>checkCommandParams.TC4</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>30</td>
</tr>
<tr>
<td>checkCommandParams.TC5</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>80</td>
<td>50</td>
</tr>
<tr>
<td>checkCommandParams.TC6</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>664</td>
<td>44</td>
</tr>
<tr>
<td>checkCommandParams.TC7</td>
<td>238</td>
<td>20*</td>
<td>17</td>
<td>1117</td>
<td>215</td>
</tr>
</tbody>
</table>
Verification of the Test Cases

<table>
<thead>
<tr>
<th>Test Program</th>
<th>TC</th>
<th>T</th>
<th>B</th>
<th>P</th>
<th>VC</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandLoop.TC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commandLoop.TC2</td>
<td>545</td>
<td>500*</td>
<td>18</td>
<td>3</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>commandLoop.TC3</td>
<td>545</td>
<td>17</td>
<td>18</td>
<td>3</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>commandLoop.TC4</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>5</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>commandLoop.TC5</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>commandLoop.TC6</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>commandLoop.TC7</td>
<td>545</td>
<td>1</td>
<td>18</td>
<td>15</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>checkCommandParams.TC1</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>56</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>checkCommandParams.TC2</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>checkCommandParams.TC3</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>37</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>checkCommandParams.TC4</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>checkCommandParams.TC5</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>80</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>checkCommandParams.TC6</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>664</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>checkCommandParams.TC7</td>
<td>238</td>
<td>20*</td>
<td>17</td>
<td>1117</td>
<td>215</td>
<td></td>
</tr>
</tbody>
</table>

ESBMC fails to verify these functions due to memory limitations and time-outs.
## Verification of the Test Cases

If we use the test cases to guide the symbolic execution, ESBMC can verify these functions with a larger bound:

<table>
<thead>
<tr>
<th>Function</th>
<th>L</th>
<th>B</th>
<th>P</th>
<th>VC</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandLoop.TC1</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC4</td>
<td>545</td>
<td>500*</td>
<td>18</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>commandLoop.TC5</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC6</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC7</td>
<td>545</td>
<td>1</td>
<td>18</td>
<td>15</td>
<td>19</td>
</tr>
<tr>
<td>checkCommandParams.TC1</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>56</td>
<td>9</td>
</tr>
<tr>
<td>checkCommandParams.TC2</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>checkCommandParams.TC3</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>37</td>
<td>5</td>
</tr>
<tr>
<td>checkCommandParams.TC4</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>30</td>
</tr>
<tr>
<td>checkCommandParams.TC5</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>80</td>
<td>50</td>
</tr>
<tr>
<td>checkCommandParams.TC6</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>664</td>
<td>44</td>
</tr>
<tr>
<td>checkCommandParams.TC7</td>
<td>238</td>
<td>20*</td>
<td>17</td>
<td>1117</td>
<td>215</td>
</tr>
</tbody>
</table>
**Verification of the Test Cases**

<table>
<thead>
<tr>
<th></th>
<th>L</th>
<th>B</th>
<th>P</th>
<th>VC</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandLoop.TC3</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC4</td>
<td>545</td>
<td>17</td>
<td>18</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>commandLoop.TC5</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC6</td>
<td>545</td>
<td>-</td>
<td>18</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>commandLoop.TC7</td>
<td>545</td>
<td>1</td>
<td>18</td>
<td>15</td>
<td>19</td>
</tr>
<tr>
<td>checkCommandParams.TC1</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>56</td>
<td>9</td>
</tr>
<tr>
<td>checkCommandParams.TC2</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>checkCommandParams.TC3</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>37</td>
<td>5</td>
</tr>
<tr>
<td>checkCommandParams.TC4</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>36</td>
<td>30</td>
</tr>
<tr>
<td>checkCommandParams.TC5</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>80</td>
<td>50</td>
</tr>
<tr>
<td>checkCommandParams.TC6</td>
<td>238</td>
<td>17</td>
<td>17</td>
<td>664</td>
<td>44</td>
</tr>
<tr>
<td>checkCommandParams.TC7</td>
<td>238</td>
<td>20</td>
<td>17</td>
<td>1117</td>
<td>215</td>
</tr>
</tbody>
</table>

ESBMC is not able to prove or falsify some of the properties due to unwinding violations.
## Equivalence Checking

<table>
<thead>
<tr>
<th>Test Program</th>
<th>L</th>
<th>B</th>
<th>P</th>
<th>Time</th>
<th>PR10</th>
<th>PR11</th>
<th>PR12</th>
<th>PR13</th>
</tr>
</thead>
<tbody>
<tr>
<td>threadRename</td>
<td>6</td>
<td>17</td>
<td>0</td>
<td>3</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fileExists</td>
<td>19</td>
<td>17</td>
<td>0</td>
<td>3</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>readLine</td>
<td>27</td>
<td>17</td>
<td>11</td>
<td>3</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getCommand</td>
<td>269</td>
<td>17</td>
<td>61</td>
<td>3</td>
<td>X</td>
<td>N/3</td>
<td>N/3</td>
<td></td>
</tr>
<tr>
<td>powerDown</td>
<td>9</td>
<td>17</td>
<td>0</td>
<td>2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>digitStart</td>
<td>12</td>
<td>17</td>
<td>0</td>
<td>2</td>
<td>X</td>
<td>Y/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>digitAdd</td>
<td>34</td>
<td>17</td>
<td>2</td>
<td>2</td>
<td>X</td>
<td>Y/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkEndOfPvrStream</td>
<td>32</td>
<td>17</td>
<td>13</td>
<td>2</td>
<td>X</td>
<td></td>
<td></td>
<td>Y/2</td>
</tr>
<tr>
<td>checkEndOfMediaStream</td>
<td>28</td>
<td>17</td>
<td>1</td>
<td>2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commandLoop</td>
<td>545</td>
<td>17</td>
<td>53</td>
<td>M_f</td>
<td>X</td>
<td>M_f</td>
<td>M_f</td>
<td></td>
</tr>
<tr>
<td>checkCommandParams</td>
<td>238</td>
<td>17</td>
<td>269</td>
<td>T_b</td>
<td>X</td>
<td>T_b</td>
<td>T_b</td>
<td></td>
</tr>
<tr>
<td>singal_handler</td>
<td>13</td>
<td>17</td>
<td>0</td>
<td>2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setupFBResolution</td>
<td>29</td>
<td>17</td>
<td>0</td>
<td>2</td>
<td>X</td>
<td>Y/3</td>
<td>Y/3</td>
<td>Y/3</td>
</tr>
<tr>
<td>setupFramebuffers</td>
<td>115</td>
<td>17</td>
<td>8</td>
<td>3</td>
<td>X</td>
<td>N/3</td>
<td>N/2</td>
<td>N/2</td>
</tr>
<tr>
<td>main_Thread</td>
<td>68</td>
<td>17</td>
<td>4</td>
<td>4</td>
<td>X</td>
<td></td>
<td>Y/3</td>
<td>Y/2</td>
</tr>
</tbody>
</table>
## Equivalence Checking

<table>
<thead>
<tr>
<th>Test Program</th>
<th>L</th>
<th>B</th>
<th>P</th>
<th>Ti</th>
</tr>
</thead>
<tbody>
<tr>
<td>threadRename</td>
<td>6</td>
<td>17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>fileExists</td>
<td>19</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>readLine</td>
<td>27</td>
<td>17</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td><strong>getCommand</strong></td>
<td>269</td>
<td>17</td>
<td>61</td>
<td>X</td>
</tr>
<tr>
<td>powerDown</td>
<td>9</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>digitStart</strong></td>
<td>12</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>difgitAdd</strong></td>
<td>34</td>
<td>17</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>checkEndOfPvrStream</td>
<td>32</td>
<td>17</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>checkEndOfMediaStream</td>
<td>28</td>
<td>17</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>commandLoop</td>
<td>545</td>
<td>17</td>
<td>53</td>
<td>M_f</td>
</tr>
<tr>
<td>checkCommandParams</td>
<td>238</td>
<td>17</td>
<td>269</td>
<td>T_b</td>
</tr>
<tr>
<td>singal_handler</td>
<td>13</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>setupFBResolution</strong></td>
<td>29</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>setupFramebuffers</strong></td>
<td>115</td>
<td>17</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>main_Thread</td>
<td>68</td>
<td>17</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Each PR only changes a few functions, but while six functions remain unchanged over all PRs, there are changes in each individual PR.
## Equivalence Checking

<table>
<thead>
<tr>
<th>Test Program</th>
<th>L</th>
<th>B</th>
<th>P</th>
<th>Ti</th>
</tr>
</thead>
<tbody>
<tr>
<td>threadRename</td>
<td>6</td>
<td>17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>fileExists</td>
<td>19</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>readLine</td>
<td>27</td>
<td>17</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>getCommand</td>
<td>269</td>
<td>17</td>
<td>61</td>
<td>3</td>
</tr>
<tr>
<td>powerDown</td>
<td>9</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>digitStart</td>
<td>12</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>difdigitAdd</td>
<td>34</td>
<td>17</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>checkEndOfPvrStream</td>
<td>32</td>
<td>17</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>checkEndOfMediaStream</td>
<td>28</td>
<td>17</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>commandLoop</td>
<td>545</td>
<td>17</td>
<td>53</td>
<td>M</td>
</tr>
<tr>
<td>checkCommandParams</td>
<td>238</td>
<td>17</td>
<td>269</td>
<td>T</td>
</tr>
<tr>
<td>singal_handler</td>
<td>13</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>setupFBResolution</td>
<td>29</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>setupFramebuffers</td>
<td>115</td>
<td>17</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>main_Thread</td>
<td>68</td>
<td>17</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

We have 19 changes over all PRs, where 8 changes are equivalent, 5 changes are not equivalent and we fail to check 5 changes.
Medical Device Case Study

• Goal: check ESBMC’s performance in verifying temporal properties

• embedded software of a pulse oximeter device
  – device drivers (display, keyboard, serial, sensor, and timer)
  – system log to debug code
  – applications that call the services provided by the platform

• Set-up:
  – ESBMC v1.15.1 together with the SMT solver Z3 v2.11
  – standard desktop PC, time-out 3600 seconds
Medical Device Case Study

- **P1:** whenever the bit 0 of the micro-controller port is set to 1, the start button will eventually be detected
  - include two Boolean variables (*BIT0* and *startButton*)

\[
AG (BIT0 \rightarrow F\ startButton)
\]

- **P2:** whenever the start button is pressed, the application will eventually be initialized
  - include two Boolean variables (*startButton* and *startApp*)

\[
AG (next < buffer\_size)
\]

- **P3:** it is possible to get to a state where the next position of the buffer is less than its total size
  - no changes to the program

\[
AG (startButton \rightarrow F\ startApp)
\]
Faults Injected

- **keyboard**: we comment out the break statement (of the case `START: command=startButton`)
  - if `START` was pressed, the code would fall through to the next line, and have the wrong value assigned to `command`

- **menu_app**: we do not initialize the application after the start button is pressed

- **log**: we change the program statements so that in a situation where the `next` index is at the end of the array `buffer`, an overflowing index by one byte can occur

  original: \[ next = (next+1) \mod buffer\_size \]

  fault: \[ next \%= buffer\_size \]
  \[ next+=1 \]
## Verification of the LTL Properties

<table>
<thead>
<tr>
<th>Test Program</th>
<th>L</th>
<th>T</th>
<th>B</th>
<th>C</th>
<th>Time</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyboard</td>
<td>49</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>7</td>
<td>0/120</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>-</td>
<td>80</td>
<td>0/1001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>-</td>
<td>107</td>
<td>0/8568</td>
</tr>
<tr>
<td>keyboard†</td>
<td>49</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>1</td>
<td>2/6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>-</td>
<td>1</td>
<td>3/8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>-</td>
<td>1</td>
<td>4/10</td>
</tr>
<tr>
<td>menu_app</td>
<td>847</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>16</td>
<td>0/3003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>20</td>
<td>271</td>
<td>0/50456</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>20</td>
<td>625</td>
<td>0/87386</td>
</tr>
<tr>
<td>menu_app†</td>
<td>847</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>9</td>
<td>663/3003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>20</td>
<td>121</td>
<td>7584/50456</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>20</td>
<td>218</td>
<td>12548/87386</td>
</tr>
<tr>
<td>log</td>
<td>135</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>12</td>
<td>0/12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>-</td>
<td>820</td>
<td>0/22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>10</td>
<td>1149</td>
<td>0/8</td>
</tr>
<tr>
<td>log†</td>
<td>135</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>1</td>
<td>12/16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>-</td>
<td>3</td>
<td>27/31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>-</td>
<td>5</td>
<td>48/52</td>
</tr>
</tbody>
</table>
## Verification of the LTL Properties

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>B</th>
<th>C</th>
<th>Time</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>7</td>
<td>0/120</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
<td>80</td>
<td></td>
<td>0/1001</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-</td>
<td>107</td>
<td></td>
<td>0/8568</td>
</tr>
<tr>
<td>keyboard†</td>
<td>49</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyboard†</td>
<td>49</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-</td>
<td>1</td>
<td></td>
<td>2/6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
<td>1</td>
<td></td>
<td>3/8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-</td>
<td>1</td>
<td></td>
<td>4/10</td>
</tr>
<tr>
<td>menu_app</td>
<td>847</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>menu_app</td>
<td>847</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>menu_app†</td>
<td>847</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>menu_app†</td>
<td>847</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-</td>
<td>16</td>
<td></td>
<td>0/3003</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
<td>20</td>
<td>271</td>
<td>0/50456</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-</td>
<td>20</td>
<td>625</td>
<td>0/87386</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-</td>
<td>9</td>
<td></td>
<td>663/3003</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20</td>
<td>121</td>
<td></td>
<td>7584/50456</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>20</td>
<td>218</td>
<td></td>
<td>12548/87386</td>
</tr>
<tr>
<td>log</td>
<td>135</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>135</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-</td>
<td>10</td>
<td>1149</td>
<td>0/8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-</td>
<td>12</td>
<td></td>
<td>0/12</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
<td>820</td>
<td></td>
<td>0/22</td>
</tr>
<tr>
<td>log†</td>
<td>135</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>log†</td>
<td>135</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-</td>
<td>5</td>
<td></td>
<td>48/52</td>
</tr>
</tbody>
</table>

The reactive system: ESBMC can check the LTL properties up to a certain unwinding bound.
## Verification of the LTL Properties

<table>
<thead>
<tr>
<th>T</th>
<th>B</th>
<th>C</th>
<th>Time</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>-</td>
<td>7</td>
<td>0/120</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>80</td>
<td>0/1001</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>107</td>
<td>0/8568</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>1</td>
<td>2/6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>1</td>
<td>3/8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>1</td>
<td>4/10</td>
<td></td>
</tr>
</tbody>
</table>

### for small values of the unwinding bound, ESBMC verifies the properties without a specified upper bound on the context switches

<table>
<thead>
<tr>
<th>T</th>
<th>B</th>
<th>C</th>
<th>Time</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>-</td>
<td>16</td>
<td>0/3003</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>271</td>
<td>0/50456</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>625</td>
<td>0/87386</td>
<td></td>
</tr>
</tbody>
</table>

### menu_app

<table>
<thead>
<tr>
<th>T</th>
<th>B</th>
<th>C</th>
<th>Time</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>-</td>
<td>9</td>
<td>663/3003</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>121</td>
<td>7584/50456</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>218</td>
<td>12548/87386</td>
<td></td>
</tr>
</tbody>
</table>

### menu_app†

<table>
<thead>
<tr>
<th>T</th>
<th>B</th>
<th>C</th>
<th>Time</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-</td>
<td>12</td>
<td>0/12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>820</td>
<td>0/22</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>1149</td>
<td>0/8</td>
<td></td>
</tr>
</tbody>
</table>

### log

<table>
<thead>
<tr>
<th>T</th>
<th>B</th>
<th>C</th>
<th>Time</th>
<th>#FI/#I</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-</td>
<td>3</td>
<td>27/31</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>5</td>
<td>48/52</td>
<td></td>
</tr>
</tbody>
</table>
### Verification of the LTL Properties

<table>
<thead>
<tr>
<th>Test Problem</th>
<th>#FI/#I</th>
<th>Violation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyboard</td>
<td>2/6</td>
<td>1</td>
</tr>
<tr>
<td>keyboard†</td>
<td>3/8</td>
<td>1</td>
</tr>
<tr>
<td>menu_app</td>
<td>0/3003</td>
<td>16</td>
</tr>
<tr>
<td>menu_app†</td>
<td>663/3003</td>
<td>9</td>
</tr>
<tr>
<td>log</td>
<td>0/12</td>
<td>12</td>
</tr>
<tr>
<td>log†</td>
<td>12/16</td>
<td>1</td>
</tr>
</tbody>
</table>

ESBMC is able to detect the violation in few seconds and about 15% of the generated interleavings fail.
Agenda

• SMT-based BMC for Embedded ANSI-C Software
• Verifying Multi-threaded Software
• Implementation of ESBMC
• Integrating ESBMC into Software Engineering Practice

• Conclusions and Future Work
Results

• described and evaluated first SMT-based BMC for full ANSI-C
  – no SMT tool existed that can reliably handle full ANSI-C
  – provided encodings for typical ANSI-C constructs not directly supported by SMT-solvers
    ⇒ used three different SMT solvers to check the effectiveness of our encoding
  – found undiscovered bugs related to arithmetic overflow, buffer overflow and invalid pointer in standard benchmarks suite
    ⇒ confirmed by the benchmark’s creators

• lazy, schedule recording, and UW algorithms
  – lazy: check constraints lazily is fast for satisfiable instances and to a lesser extent even for safe programs
    ⇒ it has not been described or evaluated in the literature
Results

• lazy, schedule recording, and UW algorithms
  – schedule recording: the number of threads and context switches can grow quickly (and easily “blow-up” the model checker)
    \[ \Rightarrow \text{combines symbolic with explicit state space exploration} \]
  – UW: memory overhead and slowdowns to extract the unsat core
    \[ \Rightarrow \text{it has not been used for BMC of multi-threaded software} \]
    \[ \Rightarrow \text{uses a different encoding based on the notion of ECS blocks} \]

Future Work

• fault localization in multi-threaded C programs
• interpolants to prove no interference of context switches
• verify real-time software using SMT techniques