Bounded Model Checking of Multi-threaded Software using SMT solvers

Lucas Cordeiro and Bernd Fischer

lcc08r@ecs.soton.ac.uk
Bounded Model Checking of Multi-threaded Software using SMT solvers

Lucas Cordeiro and Bernd Fischer

lcc08r@ecs.soton.ac.uk
Bounded Model Checking (BMC)

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

\[ \neg \varphi_0 \lor \neg \varphi_1 \lor \neg \varphi_2 \lor \neg \varphi_{k-1} \lor \neg \varphi_k \]

• 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 (embedded) 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 logic that is not relevant
Objective of this work

Exploit SMT to improve BMC of multi-threaded software

• exploit SMT solvers to:
  – prune the property and data dependent search space (non-chronological backtracking and conflict clauses learning)
  – remove interleavings that are not relevant by analyzing the proof of unsatisfiability [NOT Craig Interpolants yet]

• 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 over multi-threaded applications
Lazy exploration of interleavings

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

C/C++ source → IRep tree → properties → BMC → SMT solver

- multi-threaded goto programs
- symbolic execution engine
- scan, parse, and type-check
- deadlock, atomicity and order violations, etc...

reused/extended from the CBMC model checker
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 → scheduler → BMC → SMT solver

scan, parse, and type-check

properties

guide the symbolic execution

symbolic execution engine

QF formula generation

check satisfiability using an SMT solver

deadlock, atomicity and order violations, etc...

reused/extended from the CBMC model checker

stop the generate-and-test loop if there is an error
Lazy exploration: Scheduler

- the scheduler allows one thread to execute at a given time (emulate the smallest sub-
  Task graphs)

```
Thread T1  Thread T2
  a1      b1
  a2      b2
```

Thread interleavings:
- \(a_1; a_2; b_1; b_2\)
- \(a_1; b_1; a_2; b_2\)
- ...

- allow preemptions only before visible statements (global variables and synchronization points)
Running Example

- the program has sequences of operations that need to be protected together to avoid atomicity violation
  - requirement: the region of code (\textit{val1} and \textit{val2}) should execute atomically

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

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

program counter: 0
mutexes: \textit{m1} = 0; \textit{m2} = 0;
global variables: \textit{val1} = 0; \textit{val2} = 0;
local variables: \textit{t1} = -1; \textit{t2} = -1;

A state \( s \in S \) consists of the value of the program counter \( pc \) and the values of all program variables
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: 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 $l_s$

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: $\text{val1}=1; \text{val2}=0$
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===(t1+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);

program counter: 8
mutexes: \( m1=1; m2=0; \)
global variables: \( \text{val1}=1; \text{val2}=0; \)
local variables: \( t1= -1; t2= -1; \)

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));
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11
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==val1+1);

program counter: 11
mutexes: $m1=1; m2=0$;
global variables: $\text{val1}=1; \text{val2}=0$;
local variables: $\text{t1}=1; \text{t2}=-1$;
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: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$
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: 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: $W_{\text{twoStage,2}} - R_{\text{reader,8}} - R_{\text{reader,11}} - R_{\text{twoStage,5}}$

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: 5**

mutexes: $m1=0; m2=1$;
global variables: $\text{val1}=1; \text{val2}=2$;
local variabes: $t1= 1; t2= -1$;
Lazy exploration: interleaving $I_s$

statements: 1-2-3-7-8-11-12-4-5-6
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: 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: 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
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 \( l_s \)

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

\[
\text{val1-access: } W_{\text{twoStage},2} - R_{\text{reader},8} - R_{\text{reader},11} - R_{\text{twoStage},5} \\
\text{val2-access: } W_{\text{twoStage},5} - R_{\text{reader},14}
\]

**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 \( \Gamma_f \)

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; \textbf{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: 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 $l_f$

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: }
12: t1 = val1;
13: unlock(m1);
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

- **twoStage**, reader
  - val1=0, val2=0,
  - m1=1, m2=0,…

- **twoStage**, reader
  - val1=0, val2=0,
  - m1=0, m2=0,…

**thread identifiers**

**global and local variables**

---

**CS1**

**twoStage**, reader
- val1=0, val2=0,
- m1=1, m2=0,…

**twoStage**, reader
- val1=0, val2=0,
- m1=1, m2=0,…

---

**CS2**

**twoStage**, reader
- val1=0, val2=0,
- m1=1, m2=0,…

**twoStage**, reader
- val1=0, val2=0,
- m1=1, m2=0,…

---

**execution paths**

---

**blocked execution paths** (eliminate)
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 \( 0 < ts_i \leq \text{number of threads} \)
  - 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

**twoStage, reader**

- **twoStage, reader**
  - **ts₁==1 → lock(m1)**
  - **ts₁==2 → lock(m1)**

**CS1**

- **twoStage, reader**
  - **ts₁==1 ∧ ts₂==1 → val1=1**
  - **ts₁==1 ∧ ts₂==2 → lock(m1)**

- **twoStage, reader**
  - **ts₁==2 ∧ ts₂==1 → lock(m1)**

**CS2**

- **twoStage, reader**
  - **ts₁==2 ∧ ts₂==2 → unlock(m1)**

**thread identifiers**

**program statement**
Schedule Recording: Execution Paths

SMT solver instantiates \( ts \) to evaluate all possible paths

**twoStage, reader**

- \( ts_1 = 1 \rightarrow \text{lock}(m1) \)

**twoStage, reader**

- \( ts_1 = 2 \rightarrow \text{lock}(m1) \)

If the guard of the parent node is \textit{false} then the guard of the child node is \textit{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);

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

ECS block: sequence of program statements that are executed with no intervening ECS
Schedule Recording: Interleaving $I_s$

Statements: 1
TwoStage-ECS: $ts_{1,1}$
Reader-ECS:

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

Guarded statement can only be executed if statement 1 is scheduled in the ECS block 1.

```
Thread reader
7: lock(m1);
8: val1 = 1;
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));
```

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: $t_{s1,1}-t_{s2,2}-t_{s3,3}$

reader-ECS:

Thread twoStage
1: lock(m1); $t_{s1} == 1$
2: val1 = 1; $t_{s2} == 1$
3: unlock(m1); $t_{s3} == 1$
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: 1-2-3-7

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);
10: return NULL;
11: t1 = val1; $ts_5 == 2$
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) {  $ts_5 = 2$
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

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;
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; $ts_5 == 2$
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2===(t1+1));
Schedule Recording: Interleaving $I_s$

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

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

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

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);

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);
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: $t_{s_{1,1}}-t_{s_{2,2}}-t_{s_{3,3}}-t_{s_{4,8}}-t_{s_{5,9}}-t_{s_{6,10}}$

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

Thread twoStage
1: lock(m1); $t_{s_1} = 1$
2: val1 = 1; $t_{s_2} = 1$
3: unlock(m1); $t_{s_3} = 1$
4: lock(m2); $t_{s_8} = 1$
5: val2 = val1 + 1; $t_{s_9} = 1$
6: unlock(m2); $t_{s_{10}} = 1$

Thread reader
7: lock(m1); $t_{s_4} = 2$
8: if (val1 == 0) { $t_{s_5} = 2$
9: unlock(m1);
10: return NULL; }
11: t1 = val1; $t_{s_6} = 2$
12: unlock(m1); $t_{s_7} = 2$
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: $t_{s_{1,1}} - t_{s_{2,2}} - t_{s_{3,3}} - t_{s_{4,8}} - t_{s_{5,9}} - t_{s_{6,10}}$

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

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;
} $ts_5 == 2$
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: $ts_{1,1}$-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$_{13,11}$

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:** $t_{s_{1,1}}-t_{s_{2,2}}-t_{s_{3,3}}-t_{s_{4,8}}-t_{s_{5,9}}-t_{s_{6,10}}$

**reader-ECS:** $t_{s_{7,4}}-t_{s_{8,5}}-t_{s_{11,6}}-t_{s_{12,7}}-t_{s_{13,11}}-t_{s_{14,12}}-t_{s_{15,13}}$

---

**Thread twoStage**
1: lock(m1); $t_{s_1} == 1$
2: val1 = 1; $t_{s_2} == 1$
3: unlock(m1); $t_{s_3} == 1$
4: lock(m2); $t_{s_8} == 1$
5: val2 = val1 + 1; $t_{s_9} == 1$
6: unlock(m2); $t_{s_{10}} == 1$

---

**Thread reader**
7: lock(m1); $t_{s_4} == 2$
8: if (val1 == 0) {
9: unlock(m1);
10: return NULL; }
11: t1 = val1; $t_{s_6} == 2$
12: unlock(m1); $t_{s_7} == 2$
13: lock(m2); $t_{s_{11}} == 2$
14: t2 = val2; $t_{s_{12}} == 2$
15: unlock(m2); $t_{s_{13}} == 2$
16: assert(t2==(t1+1));
Schedule Recording: Interleaving $I_s$

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

twoStage-ECS: $t_{s1,1}$-$t_{s2,2}$-$t_{s3,3}$-$t_{s4,8}$-$t_{s5,9}$-$t_{s6,10}$

reader-ECS: $t_{s7,4}$-$t_{s8,5}$-$t_{s11,6}$-$t_{s12,7}$-$t_{s13,11}$-$t_{s14,12}$-$t_{s15,13}$-$t_{s16,14}$

<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>2: val1 = 1;</td>
<td>8: if (val1 == 0) {</td>
</tr>
<tr>
<td>3: unlock(m1);</td>
<td>9: unlock(m1);</td>
</tr>
<tr>
<td>4: lock(m2);</td>
<td>10: return NULL; }</td>
</tr>
<tr>
<td>5: val2 = val1 + 1;</td>
<td>11: t1 = val1;</td>
</tr>
<tr>
<td>6: unlock(m2);</td>
<td>12: unlock(m1);</td>
</tr>
<tr>
<td></td>
<td>13: lock(m2);</td>
</tr>
<tr>
<td></td>
<td>14: t2 = val2;</td>
</tr>
<tr>
<td></td>
<td>15: unlock(m2);</td>
</tr>
<tr>
<td></td>
<td>16: assert(t2===(t1+1));</td>
</tr>
</tbody>
</table>

$ts_1 = 1$
$ts_2 = 1$
$ts_3 = 1$
$ts_8 = 1$
$ts_9 = 1$
$ts_{10} = 1$
$ts_4 = 2$
$ts_5 = 2$
$ts_6 = 2$
$ts_7 = 2$
$ts_{11} = 2$
$ts_{12} = 2$
$ts_{13} = 2$
$ts_{14} = 2$
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: $ts_{7,4} - ts_{8,5} - ts_{11,6} - ts_{12,7} - ts_{13,8} - ts_{14,9} - ts_{15,10} - ts_{16,11}$
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>
</tr>
</thead>
<tbody>
<tr>
<td>1: lock(m1);</td>
</tr>
<tr>
<td>2: val1 = 1;</td>
</tr>
<tr>
<td>3: unlock(m1);</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>

\[
\begin{align*}
cl_{1,\text{twoStage}} & \rightarrow ts_1 = 1 \\
cl_{2,\text{twoStage}} & \rightarrow ts_2 = 1 \\
cl_{3,\text{twoStage}} & \rightarrow ts_3 = 1 \\
cl_{8,\text{twoStage}} & \rightarrow ts_8 = 1 \\
cl_{9,\text{twoStage}} & \rightarrow ts_9 = 1 \\
cl_{10,\text{twoStage}} & \rightarrow ts_{10} = 1
\end{align*}
\]

- 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
  – under-approximation and widening

• Set-up:
  – ESBMC v1.6 together with the SMT solver Z3 v2.7
  – support the logics \textit{QF}\textunderscore\textit{AUFBV} and \textit{QF}\textunderscore\textit{AUFLIRA}
  – standard desktop PC, time-out 3600 seconds
### About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>k</th>
<th>#T</th>
<th>#C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frangipani file system</td>
<td>81</td>
<td>47</td>
<td>27</td>
<td>26</td>
<td>2</td>
<td>Frangipani file system with array out of bounds</td>
</tr>
<tr>
<td>Frangipani file system switches</td>
<td>48</td>
<td>48</td>
<td>27</td>
<td>27</td>
<td>2</td>
<td>Frangipani file system with array out of bounds</td>
</tr>
<tr>
<td>insert messages into a hash table concurrently</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>13</td>
<td>4</td>
<td>insert messages into a hash table concurrently</td>
</tr>
<tr>
<td>Multi-threaded download</td>
<td>84</td>
<td>84</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>Multi-threaded download</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>1233</td>
<td>1233</td>
<td>1233</td>
<td>1233</td>
<td>1233</td>
<td>aget-0.4_bad</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>84</td>
<td>84</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>reorder_bad</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>128</td>
<td>128</td>
<td>8</td>
<td>16</td>
<td>30</td>
<td>twostage_bad</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>110</td>
<td>110</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>wronglock_bad</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>1060</td>
<td>1060</td>
<td>1060</td>
<td>1060</td>
<td>1060</td>
<td>exStbHDMI_ok</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>425</td>
<td>425</td>
<td>425</td>
<td>425</td>
<td>425</td>
<td>exStbLED_ok</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>1109</td>
<td>1109</td>
<td>1109</td>
<td>1109</td>
<td>1109</td>
<td>exStbThumb_bad</td>
</tr>
</tbody>
</table>

- **lines of code**
- **number of context switches**
- **number of properties checked**
- **Number of threads**
- **the number of BMC unrolling steps**

- Demonstrate how thumbnail images can be displayed and layer blending can be used
## About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>L</th>
<th>P</th>
<th>k</th>
<th>T</th>
<th>C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_ok</td>
<td>1</td>
<td>47</td>
<td>27</td>
<td>26</td>
<td>2</td>
<td>Frangipani file system</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>80</td>
<td>48</td>
<td>28</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>21</td>
<td>129</td>
<td>13</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>170</td>
<td>257</td>
<td>3</td>
<td>2</td>
<td>Multi-threaded download accelerator</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>84</td>
<td>16</td>
<td>11</td>
<td>20</td>
<td>3</td>
<td>contains a data race</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>128</td>
<td>8</td>
<td>16</td>
<td>30</td>
<td>3</td>
<td>contains an atomicity violation</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>110</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>threads do not obtain the same lock instance</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>1060</td>
<td>25</td>
<td>16</td>
<td>2</td>
<td>20</td>
<td>set various capabilities in the HDMI device</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>425</td>
<td>45</td>
<td>10</td>
<td>2</td>
<td>-</td>
<td>front panel LED display</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>1109</td>
<td>243</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Demonstrate how thumbnail images can be displayed and layer blending can be used</td>
</tr>
</tbody>
</table>

*Inspect benchmark suite*
# About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>k</th>
<th>#T</th>
<th>#C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 fsbench_ok</td>
<td>81</td>
<td>47</td>
<td>27</td>
<td>26</td>
<td>2</td>
<td>Frangipani file system</td>
</tr>
<tr>
<td>2 fsbench_bad</td>
<td>80</td>
<td>48</td>
<td>28</td>
<td>27</td>
<td>2</td>
<td>Frangipani file system with array out of bounds</td>
</tr>
<tr>
<td>3 indexer_ok</td>
<td>77</td>
<td>21</td>
<td>129</td>
<td>13</td>
<td>4</td>
<td>insert messages into a hash table concurrently</td>
</tr>
<tr>
<td>4 aget-0.4_bad</td>
<td>1233</td>
<td>170</td>
<td>257</td>
<td>3</td>
<td>2</td>
<td>Multi-threaded download</td>
</tr>
<tr>
<td>5 reorder_bad</td>
<td>84</td>
<td>16</td>
<td>11</td>
<td>20</td>
<td>3</td>
<td>contains a data race</td>
</tr>
<tr>
<td>6 twostage_bad</td>
<td>128</td>
<td>8</td>
<td>16</td>
<td>30</td>
<td>3</td>
<td>contains an atomicity violation</td>
</tr>
<tr>
<td>7 wronglock_bad</td>
<td>110</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>threads do not obtain the same lock instance</td>
</tr>
<tr>
<td>8 exStbHDMI_ok</td>
<td>1060</td>
<td>25</td>
<td>16</td>
<td>2</td>
<td>20</td>
<td>set various capabilities in the HDMI device</td>
</tr>
<tr>
<td>9 exStbLED_ok</td>
<td>425</td>
<td>45</td>
<td>10</td>
<td>2</td>
<td>-</td>
<td>front panel LED display</td>
</tr>
<tr>
<td>10 exStbThumb_bad</td>
<td>1109</td>
<td>243</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Demonstrate how thumbnail images can be displayed and layer blending can be used</td>
</tr>
</tbody>
</table>
### About the benchmarks

<table>
<thead>
<tr>
<th>Module</th>
<th>#L</th>
<th>#P</th>
<th>k</th>
<th>#T</th>
<th>#C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 fsbench_ok</td>
<td>81</td>
<td>47</td>
<td>27</td>
<td>26</td>
<td>2</td>
<td>Frangipani file system</td>
</tr>
<tr>
<td>2 fsbench_bad</td>
<td>80</td>
<td>48</td>
<td>28</td>
<td>27</td>
<td>2</td>
<td>Frangipani file system with array out of bounds</td>
</tr>
<tr>
<td>3 indexer_ok</td>
<td>77</td>
<td>21</td>
<td>129</td>
<td>13</td>
<td>4</td>
<td>insert messages into a hash table concurrently</td>
</tr>
<tr>
<td>4 aget-0.4_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Multi-threaded download</td>
</tr>
<tr>
<td>5 reorder_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Set applications from NXP semiconductors contain a data race</td>
</tr>
<tr>
<td>6 twostage_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Set applications from NXP semiconductors contain an atomicity violation</td>
</tr>
<tr>
<td>7 wronglock_bad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>threads do not obtain the same lock instance</td>
</tr>
<tr>
<td>8 exStbHDMI_ok</td>
<td>1060</td>
<td>25</td>
<td>16</td>
<td>2</td>
<td>20</td>
<td>set various capabilities in the HDMI device</td>
</tr>
<tr>
<td>9 exStbLED_ok</td>
<td>425</td>
<td>45</td>
<td>10</td>
<td>2</td>
<td>-</td>
<td>front panel LED display</td>
</tr>
<tr>
<td>10 exStbThumb_bad</td>
<td>1109</td>
<td>243</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Demonstrate how thumbnail images can be displayed and layer blending can be used</td>
</tr>
</tbody>
</table>
## Comparison of the approaches (1)

<table>
<thead>
<tr>
<th>Module</th>
<th>Lazy</th>
<th>UW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>#P</td>
<td>#I</td>
</tr>
<tr>
<td>Module</td>
<td>Total</td>
<td>Fail</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>729</td>
<td>1</td>
</tr>
<tr>
<td>fsbench</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>248</td>
<td>1</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>&lt;1</td>
<td>1</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>84</td>
<td>1</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>268</td>
<td>0</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>900</td>
<td>0</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>14</td>
<td>1</td>
</tr>
</tbody>
</table>
## Comparison of the approaches (1)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Lazy #P</th>
<th>Lazy #I</th>
<th>Schedule #P</th>
<th>Schedule Time</th>
<th>UW #P</th>
<th>UW Time</th>
<th>UW Iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsbench_bad</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>729/729</td>
<td>452</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>fsbench_ok</td>
<td>282</td>
<td>0</td>
<td>0</td>
<td>676/0</td>
<td>334</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>575</td>
<td>0</td>
<td>0</td>
<td>17160/0</td>
<td>238</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>248</td>
<td>1</td>
<td>0</td>
<td>2/1</td>
<td>254</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>&lt;1</td>
<td>1</td>
<td>0</td>
<td>32/20</td>
<td>&lt;1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>41/15</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>84</td>
<td>1</td>
<td>0</td>
<td>20507/16</td>
<td>293</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>268</td>
<td>0</td>
<td>0</td>
<td>1/0</td>
<td>260</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>900</td>
<td>0</td>
<td>0</td>
<td>11/0</td>
<td>MO</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>14</td>
<td>1</td>
<td>0</td>
<td>3/1</td>
<td>MO</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Note:** Lazy encoding often more efficient than schedule recording and UW
## Comparison of the approaches (2)

<table>
<thead>
<tr>
<th>Task</th>
<th>Schedule</th>
<th>UW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#I</td>
<td>#I/ #IF</td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>fsbench_ok</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>indexer_ok</td>
<td>575</td>
<td>0</td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>248</td>
<td>1</td>
</tr>
<tr>
<td>reorder_bad</td>
<td>&lt;1</td>
<td>1</td>
</tr>
<tr>
<td>twostage_bad</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>84</td>
<td>1</td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>268</td>
<td>0</td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>900</td>
<td>0</td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>14</td>
<td>1</td>
</tr>
</tbody>
</table>

*lazy encoding often more efficient than schedule recording and UW, but not always*
Comparison 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>Total</td>
<td>Time</td>
<td>Size</td>
</tr>
<tr>
<td></td>
<td>Fail</td>
<td>#P</td>
<td>#I</td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td>/IF</td>
<td>#I/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>#I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fsbench_bad</td>
<td>1</td>
<td>452</td>
<td>729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>729</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>729</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fsbench_ok</td>
<td>282</td>
<td>334</td>
<td>676</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>676</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>676</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>indexer_ok</td>
<td>575</td>
<td>238</td>
<td>1716</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1716</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1716</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aget-0.4_bad</td>
<td>248</td>
<td>254</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2/1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>2/1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reorder_bad</td>
<td>&lt;1</td>
<td>&lt;1</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>32/20</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>32/20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>twostage_bad</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>41/15</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>41/15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wronglock_bad</td>
<td>84</td>
<td>293</td>
<td>2050</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2050</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>2050</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exStbHDMI_ok</td>
<td>268</td>
<td>260</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1/0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1/0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exStbLED_ok</td>
<td>900</td>
<td>MO</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>11/0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>11/0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exStbThumb_bad</td>
<td>14</td>
<td>MO</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>3/1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>3/1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lazy encoding is extremely fast for satisfiable instances.
Results

- **lazy, schedule recording, and UW algorithms**
  - **lazy**: check constraints lazily is *fast for satisfiable instances*
  - **schedule recording**: the number of threads and context switches can grow quickly (and easily “blow-up” the model checker)
  - **UW**: memory overhead and slowdowns to extract the *unsat core*

- handle more than two threads, detect concurrency bugs (e.g., atomicity and order violations, deadlocks)

- [users.ecs.soton.ac.uk/lcc08r/esbmc/](users.ecs.soton.ac.uk/lcc08r/esbmc/)

Future Work

- partial order reduction (static and dynamic)
- interpolants to prove no interference of context switches
- fault localization in multi-threaded C programs