DSSE Dependable Systems & Software Engineering

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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.

University of Southampton
School of Electronics and Computer Science

- 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

```
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);
}
```

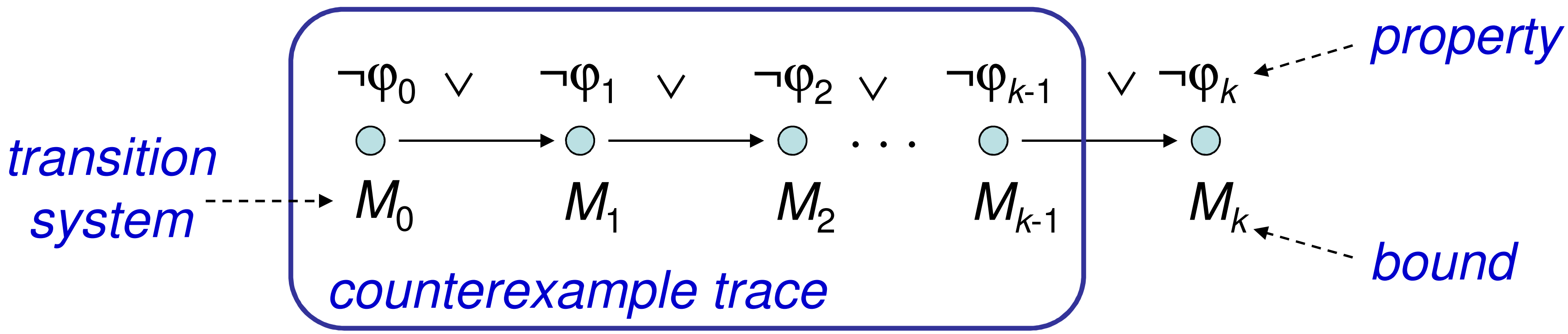
Deadlock

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

Bounded Model Checking (BMC)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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 ψ such that

  ψ satisfiable iff φ has counterexample of max. depth k
- has been applied successfully to verify (sequential) software

BMC of Multi-threaded Software

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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** (⇒ building-in operators).

Theory	**Example**
Equality	$x_1=x_2 \land \neg (x_1=x_3) \Rightarrow \neg(x_1=x_3)$
Bit-vectors	(b >> i) & 1 = 1
Linear arithmetic	$(4y_1 + 3y_2 \geq 4) \lor (y_2 - 3y_3 \leq 3)$
Arrays	$(j = k \land a[k]=2) \Rightarrow a[j]=2$
Combined theories	$(j \leq k \land a[j]=2) \Rightarrow a[i] < 3$

Satisfiability Modulo Theories (2)

- Given
  - a decidable Σ-theory *T*
  - a quantifier-free formula φ

  φ 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*

  φ is a *T*-consequence of Γ ($\Gamma \vDash_T \varphi$) iff *every model of* $T \cup \Gamma$ is also a *model of* φ
- Checking $\Gamma \vDash_T \varphi$ can be reduced in the usual way to checking the T-satisfiability of $\Gamma \cup \{\neg\varphi\}$

Satisfiability Modulo Theories (3)

- let **a** be an array, **b**, **c** and **d** be signed bit-vectors of width 16, 32 and 32 respectively, and let **g** be an unary function.

$$g(select(store(a,c,12)), SignExt(b,16)+3) \neq g(SignExt(b,16)-c+4) \wedge SignExt(b,16)=c-3 \wedge c+1=d-4$$

⇓ **b'** extends **b** to the signed equivalent bit-vector of size 32

$$step\,1: g(select(store(a,c,12),b'+3)) \neq g(b'-c+4) \wedge b'=c-3 \wedge c+1=d-4$$

⇓ replace b' by c–3 in the inequality

$$step\,2: g(select(store(a,c,12),c-3+3)) \neq g(c-3-c+4) \wedge c-3=c-3 \wedge c+1=d-4$$

⇓ using facts about bit-vector arithmetic

$$step\,3: g(select(store(a,c,12),c)) \neq g(1) \wedge c-3=c-3 \wedge c+1=d-4$$

Satisfiability Modulo Theories (4)

$$step\ 3: g(select(store(a,c,12),c)) \neq g(1) \wedge c-3 = c-3 \wedge c+1 = d-4$$

⇩ applying the theory of arrays

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

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

$$step\ 5: SAT\ (\mathrm{c} = 5, \mathrm{d} = 10)$$

- SMT solvers also apply:
  - standard algebraic reduction rules $r \wedge false \mapsto false$
  - contextual simplification $a = 7 \wedge p(a) \mapsto a = 7 \wedge p(7)$

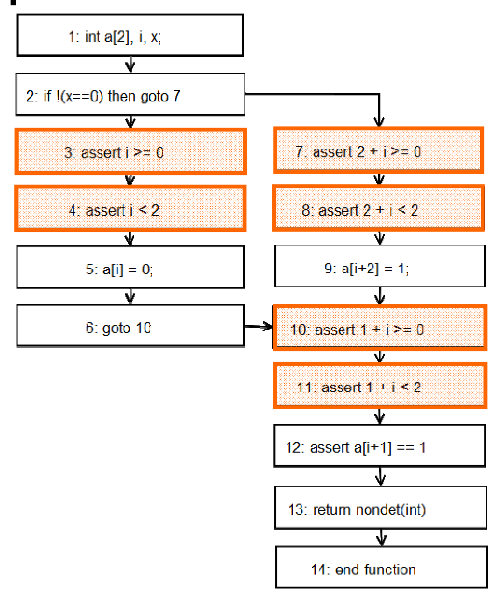
Software BMC using ESBMC

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

```
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

  } crucial
- front-end converts unrolled and optimized program into SSA

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

$g_1 = x_1 == 0$
$a_1 = a_0$ WITH $[i_0 := 0]$
$a_2 = a_0$
$a_3 = a_2$ WITH $[2+i_0 := 1]$
$a_4 = g_1 \; ? \; a_1 : a_3$
$t_1 = a_4 [1+i_0] == 1$

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

  } 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 $C \wedge \neg P$

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

$$C := \begin{bmatrix} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2+i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{bmatrix}$$

$$P := \begin{bmatrix} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\, 2+i_0 \geq 0 \wedge 2+i_0 < 2 \\ \wedge\, 1+i_0 \geq 0 \wedge 1+i_0 < 2 \\ \wedge\, select(a_4, i_0+1) = 1 \end{bmatrix}$$

Encoding of Numeric Types

- SMT solvers typically provide different encodings for numbers:
  - abstract domains ($\mathbb{Z}$, $\mathbb{R}$)
  - fixed-width bit vectors (`unsigned int`, …)
    - ▹ "internalized bit-blasting"
- verification results can depend on encodings

$$(a > 0) \wedge (b > 0) \Rightarrow (a + b > 0)$$

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

    $t = ite(v \neq k, true, false)$ *//conversion to bool*

    $v = ite(t, 1, 0)$ *//conversion from bool*
- arithmetic over- / underflow
  - standard requires modulo-arithmetic for unsigned integer

    $$unsigned\_overflow \Leftrightarrow (r - (r \bmod 2^w)) < 2^w$$
  - define error literals to detect over- / underflow for other types

    $$res\_op \Leftrightarrow \neg\, overflow(x, y) \wedge \neg\, underflow(x, y)$$
    - ▹ 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 $b = i \mathbin{@} f$ with the same bitwidth before and after the radix point of *a*.

$$i = \begin{cases} Extract(b, n_b + m_a - 1, n_b) & : \quad m_a \le m_b \\ SignExt(Extract(b, t_b - 1, n_b), m_a - m_b) & : \quad otherwise \end{cases}$$

// m = number of bits of i

$$f = \begin{cases} Extract(b, n_b - 1, n_b - n_b) & : \quad n_a \le n_b \\ ZeroExt(Extract(b, n_b - 1, 0), n_a - n_b) & : \quad otherwise \end{cases}$$

// n = number of bits of f

- **rational encoding:** convert a to *a* rational number

$$a = \begin{cases} \dfrac{\left(i * p + \left(\dfrac{f * p}{2^n} + 1\right)\right)}{p} & : \quad f \ne 0 \\ i \quad : \quad otherwise \end{cases}$$

// p = number of decimal places

Encoding of Pointers

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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)

```
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);
}
```

⟹ C :=

$$
\left(\begin{array}{l}
p_1 := store(p_0, 0, \&a[0]) \\
\wedge\ p_2 := store(p_1, 1, 0) \\
\wedge\ g_2 := (x_2 == 0) \\
\wedge\ a_1 := store(a_0, i_0, \ldots \\
\ldots \\
\wedge\ a_3 := store(a_2, 1+ i_0, 1) \\
\wedge\ a_4 := ite(g_1, a_1, a_3) \\
\wedge\ p_3 := store(p_2, 1, select(p_2 , 1)+2)
\end{array}\right)
$$

Store object at position 0

Store index at position 1

Update index

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)

```
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);
}
```

$$P := \left(\begin{array}{l} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\ 1 + i_0 \geq 0 \wedge 1 + \ldots 2 \\ \wedge\ \mathrm{select}(p_3, 0) == \&a[0] \\ \wedge\ \mathrm{select}(\mathrm{select}(p_3, 0), \\ \qquad \mathrm{select}(p_3, 1)) == 1 \end{array} \right)$$

negation satisfiable
(a[2] unconstrained)
⇒ assert fails

Encoding of Memory Allocation

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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
  - $\upsilon \triangleq$ 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

$$l_{is\_dynamic\_object} \Leftrightarrow \left(\bigvee_{j=1}^{k} d_o.\rho = j \right) \wedge (0 \le i < n)$$

Encoding of Memory Allocation

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- to check for invalid objects, we
  - set υ to *true* when the function *malloc* is called (d_o is alive)
  - set υ to *false* when the function *free* is called (d_o is not longer alive)

$$l_{valid\_object} \Leftrightarrow (l_{is\_dynamic\_object} \Rightarrow d_o.\upsilon)$$

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

$$l_{deallocated\_object} \Leftrightarrow (l_{is\_dynamic\_object} \Rightarrow \neg\, d_o.\upsilon)$$

Example of Memory Allocation

```
#include <stdlil
void main() {
  char *p = ma
  char *q =
  p=q;
  free(p)
  p = malloc(5);          // ρ = 3
  free(p)
}
```

memory leak: pointer reassignment makes $d_{o1}.\upsilon$ to become an orphan

Example of Memory Allocation

University of Southampton
School of Electronics and Computer Science

```
#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)
}
```

$$P := \left(\neg d_{o1}.\upsilon \wedge \neg d_{o2}.\upsilon \ \neg d_{o3}.\upsilon \right)$$

$$C := \left(\begin{array}{l} d_{o1}.\rho=1 \wedge d_{o1}.s=5 \wedge d_{o1}.\upsilon=\mathit{true} \wedge p=d_{o1} \\ \wedge\ d_{o2}.\rho=2 \wedge d_{o2}.s=5 \wedge d_{o2}.\upsilon=\mathit{true} \wedge q=d_{o2} \\ \wedge\ p=d_{o2} \wedge d_{o2}.\upsilon=\mathit{false} \\ \wedge\ d_{o3}.\rho=3 \wedge d_{o3}.s=5 \wedge d_{o3}.\upsilon=\mathit{true} \wedge p=d_{o3} \\ \wedge\ d_{o3}.\upsilon=\mathit{false} \end{array} \right)$$

Example of Memory Allocation


```
#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)
}
```

$$P := \left(\neg \mathbf{d_{o1}.\upsilon} \wedge \neg d_{o2}.\upsilon \; \neg d_{o3}.\upsilon \right)$$

$$C := \left(\begin{array}{l} d_{o1}.\rho=1 \wedge d_{o1}.s=5 \wedge \mathbf{d_{o1}.\upsilon=\mathit{true}} \wedge p=d_{o1} \\ \wedge\, d_{o2}.\rho=2 \wedge d_{o2}.s=5 \wedge d_{o2}.\upsilon=\mathit{true} \wedge q=d_{o2} \\ \wedge\, p=d_{o2} \wedge d_{o2}.\upsilon=\mathit{false} \\ \wedge\, d_{o3}.\rho=3 \wedge d_{o3}.s=5 \wedge d_{o3}.\upsilon=\mathit{true} \wedge p=d_{o3} \\ \wedge\, d_{o3}.\upsilon=\mathit{false} \end{array} \right)$$

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Evaluation

Comparison of SMT solvers

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#L	#P	CVC3 Time	CVC3 Error	Boolector Time	Boolector Error	Z3 Time	Z3 Error
B[illegible]	43	17	[illegible]	[illegible]	[illegible] 2)	0	**2** (3)	0
[illegible]	43	17	[illegible]	[illegible]	[illegible] 1)	0	**265** (269)	0
SelectionSort (n=35)	34	17	[illegible]	[illegible]	[illegible] (1)	0	**1 (1)**	0
(n=140)	34	17	M_b (209)	1	**161** (171)	0	165 (173)	0
InsertionSort (n=35)	86	17	4 (5)	0	**3** ([illegible]	[illegible]	[illegible]	0
(n=140)	86	17	**194** (283)	0	350 (21[illegible]	[illegible]	[illegible]	0
Prim	[illegible]	[illegible]	[illegible]	0	**<1 (<1)**	[illegible]	**<1 (<1)**	0
StrCmp	[illegible]	[illegible]	[illegible]	0	195 (257)	0	35 (46)	0
MinMax	19	9	T_b (Mb)	1	42[illegible]	0	**6** (7)	0
lms	258	23	**225** (324)	0	303[illegible]	[illegible]	[illegible] 7)	0
Bitwise	18	1	**3** (6)	0	7 (8)	0	30 (26)	0
adpcm_encode	149	12	6 (26)	0	6 (6)	0	6 (6)	0
adpcm_decode	111	10	3 (27)	0	3 (3)	0	3 (3)	0

lines of code

number of properties checked

SMT-LIB interface

size of arrays

native API

Comparison of SMT solvers

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#L	#P	CVC3		Boolector		Z3	
			Time	Error	Time	Error	Time	Error
BubbleSort (n=35)	43	17	17 (5)	0	**2 (2)**	0	**2** (3)	0
(n=140)	43	17	M_b(M_b)	1	282 (311)	0	**265** (269)	0
SelectionSort (n=35)	34	17	18 (3)	0	**1 (1)**	0	**1 (1)**	0
(n=140)	34	17	M_b (209)	1	**161** (171)	0	165 (173)	0
InsertionSort (n=35)	8[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	**3 (3)**	0
(n=140)	8[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	212 (222)	0
Prim	7[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	**<1 (<1)**	0
StrCmp	14	6	[illegible]34)	0	195 (257)	0	35 (46)	0
MinMax	19	[illegible]	T_b (Mb)	1	42 (7)	0	**6** (7)	0
lms	[illegible]38	23	**225** (324)	0	303 (307)	0	306 (307)	0
Bitwise	**18**	**1**	**3 (6)**	**0**	**7 (8)**	**0**	**30 (26)**	**0**
adpcm_encode	**149**	**12**	**6 (26)**	**0**	**6 (6)**	**0**	**6 (6)**	**0**
adpcm_decode	**111**	**10**	**3 (27)**	**0**	**3 (3)**	**0**	**3 (3)**	**0**

All SMT-solvers can handle the VCs from the embedded applications

Comparison of SMT solvers

Module	#L	#P	CVC3		Boolector		Z3	
			Time	Error	Time	Error	Time	Error
BubbleSort (n=35)	43	17	17 (5)	0	[illegible]	[illegible]	[illegible]	0
(n=140)	43	17	**M_b(M_b)**	1	28[illegible]	[illegible]	[illegible]	0
SelectionSort (n=35)	34	17	18 (3)	0	[illegible]	[illegible]	[illegible]	0
(n=140)	34	17	**M_b (209)**	1	**161** (171)	0	165 (173)	0
InsertionSort (n=35)	86	17	4 (5)	0	**3 (3)**	0	**3 (3)**	0
(n=140)	86	17	**194** (283)	0	350 (219)	0	212 (222)	0
Prim	79	30	5 (2)	0	**<1 (<1)**	0	**<1 (<1)**	0
StrCmp	14	6	**11** (454)	0	195 (257)	0	35 (46)	0
MinMax	19	9	**T_b (Mb)**	1	42 (7)	0	**6** (7)	0
lms	258	23	**225** (324)	0	303 (307)	0	306 (307)	0
Bitwise	18	1	3 (6)	0	7 (8)	0	30 (26)	0
adpcm_encode	149	12	6 (26)	0	6 (6)	0	6 (6)	0
adpcm_decode	111	10	3 (27)	0	3 (3)	0	3 (3)	0

CVC3 doesn't scale that well and runs out of memory and time

Comparison of SMT solvers

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Boolector and Z3 roughly comparable, with some advantages for Z3

Module	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		Z3	
					Time	Error	Time	Error
BubbleSort (n=[illegible])	[illegible]	[illegible]	[illegible]	[illegible]	**2 (2)**	0	**2 (3)**	0
(n=140)	43	17	$M_b(M_b)$	1	**282 (311)**	0	**265 (269)**	0
SelectionSort (n=35)	34	17	18 (3)	0	**1 (1)**	0	**1 (1)**	0
(n=140)	34	17	M_b (209)	1	**161 (171)**	0	**165 (173)**	0
InsertionSort (n=35)	86	17	4 (5)	0	**3 (3)**	0	**3 (3)**	0
(n=140)	86	17	**194** (283)	0	**350 (219)**	0	**212 (222)**	0
Prim	79	30	5 (2)	0	**<1 (<1)**	0	**<1 (<1)**	0
StrCmp	14	6	**11** (454)	0	**195 (257)**	0	**35 (46)**	0
MinMax	19	9	T_b (Mb)	1	**42 (7)**	0	**6 (7)**	0
lms	258	23	**225** (324)	0	**303 (307)**	0	**306 (307)**	0
Bitwise	18	1	**3** (6)	0	**7 (8)**	0	**30 (26)**	0
adpcm_encode	149	12	6 (26)	0	6 (6)	0	6 (6)	0
adpcm_decode	111	10	3 (27)	0	3 (3)	0	3 (3)	0

Comparison of SMT solvers

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

Mo			VC3		Boolector		Z3	
				Error	Time	Error	Time	Error
BubbleSort (n=35)	43	17	17 (5)	0	**2 (2)**	0	**2** (3)	0
(n=140)	43	17	$M_b(M_b)$	1	**282** (311)	0	**265** (269)	0
SelectionSort (n=35)	34	17	18 (3)	0	**1 (1)**	0	**1 (1)**	0
(n=140)	34	17	M_b (209)	1	**161** (171)	0	**165** (173)	0
InsertionSort (n=35)	86	17	**4** (5)	0	**3 (3)**	0	**3 (3)**	0
(n=140)	86	17	**194** (283)	0	350 (219)	0	**212** (222)	0
Prim	79	30	5 (2)	0	**<1 (<1)**	0	**<1 (<1)**	0
StrCmp	14	6	**11** (454)	0	**195** (257)	0	**35** (46)	0
MinMax	19	9	T_b (Mb)	1	42 (7)	0	**6** (7)	0
lms	258	23	**225** (324)	0	**303** (307)	0	**306** (307)	0
Bitwise	18	1	**3** (6)	0	**7** (8)	0	30 (26)	0
adpcm_encode	149	12	**6** (26)	0	6 (6)	0	6 (6)	0
adpcm_decode	111	10	**3** (27)	0	3 (3)	0	3 (3)	0

Comparison of SMT solvers

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

Mo			C3		Boolector		Z3	
				rror	Time	Error	Time	Error
BubbleSort (n=35) (n=140)	43 43	17 17	17 (5) M_b(M_b)	0 1	**2 (2)** 282 (311)	0 0	2 (3) 265 (269)	0 0
SelectionSort (n=35) (n=140)	34 34	17 17	18 (3) M_b (209)	0 1	**1 (1)** 161 (171)	0 0	**1 (1)** 165 (173)	0 0
InsertionSort (n=35) (n=140)	86 86	17 17	4 (5) 194 (283)	0 0	**3 (3)** 350 (219)	0 0	**3 (3)** 212 (222)	0 0
Prim	79	30	5 (2)	0	**<1 (<1)**	0	**<1 (<1)**	0
StrCmp	14	6	11 (454)	0	195 (257)	0	35 (46)	0
MinMax	19	9	T_b (Mb)	1	42 (7)	0	6 (7)	0
Ims	258	23	225 (324)	0	303 (307)	0	306 (307)	0
Bitwise	18	1	3 (6)	0	7 (8)	0	30 (26)	0
adpcm_encode	149	12	6 (26)	0	6 (6)	0	6 (6)	0
adpcm_decode	111	10	3 (27)	0	3 (3)	0	3 (3)	0

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Comparison to SMT-CBMC [A. Armando et al.]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

Module		ESBMC		SMT-CBMC
		Z3	CVC3	CVC3
BubbleSort	(n=35) (n=140)	<1 (<1) 259 (265)	2 (2) M_b (M_b)	100 MO
SelectionSort	(n=35) (n=140)	<1 (<1) 157 (162)	<1 (<1) 160 (193)	T T
BellmanFord		<1 (<1)	<1 (<1)	43
Prim		<1 (<1)	<1 (<1)	96
StrCmp		27 (38)	7 (261)	T
SumArray		25 (<1)	<1 (108)	98
MinMax		6 (6)	T_b (M_b)	65

Comparison to SMT-CBMC [A. Armando et al.]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

All benchmarks taken from SMT-CBMC suite

Module	Z3	CVC3	CVC3
BubbleSort **(n=35)**	<1 (<1)	2 (2)	100
(n=140)	259 (265)	M_b (M_b)	MO
SelectionSort **(n=35)**	<1 (<1)	<1 (<1)	T
(n=140)	157 (162)	160 (193)	T
BellmanFord	<1 (<1)	<1 (<1)	43
Prim	<1 (<1)	<1 (<1)	96
StrCmp	27 (38)	7 (261)	T
SumArray	25 (<1)	<1 (108)	98
MinMax	6 (6)	T_b (M_b)	65

Comparison to SMT-CBMC [A. Armando et al.]

University of Southampton
School of Electronics and Computer Science

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

Module	ESBMC		SMT-CBMC
	Z3	CVC3	CVC3
BubbleSort (n=35)	[illegible] (<1) [illegible]	**2** (2) M_b (M_b)	**100** MO
[illegible]	[illegible]	**<1** (<1) **160** (193)	**T** **T**
[illegible]	[illegible] (<1)	**<1** (<1)	**43**
Prim	<1 (<1)	**<1** (<1)	**96**
StrCmp	27 (38)	**7** (261)	**T**
SumArray	25 (<1)	**<1** (108)	**98**
MinMax	6 (6)	**T_b** (M_b)	**65**

ESBMC substantially faster, even with identical solvers
⇒ probably better encoding

Comparison to SMT-CBMC [A. Armando et al.]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

Module	ESBMC		SMT-CBMC
	Z3	CVC3	CVC3
BubbleSort ([illegible]) ([illegible]0)	**<1 (<1)** **259 (265)**	**2 (2)** **M_b (M_b)**	100 MO
[illegible]) [illegible]0)	<1 (<1) **157 (162)**	<1 (<1) **160 (193)**	T T
BellmanFord	<1 (<1)	<1 (<1)	43
Prim	<1 (<1)	<1 (<1)	96
StrCmp	**27 (38)**	**7 (261)**	T
SumArray	**25 (<1)**	**<1 (108)**	98
MinMax	**6 (6)**	**T_b (M_b)**	65

Z3 uniformly better than CVC3

Agenda

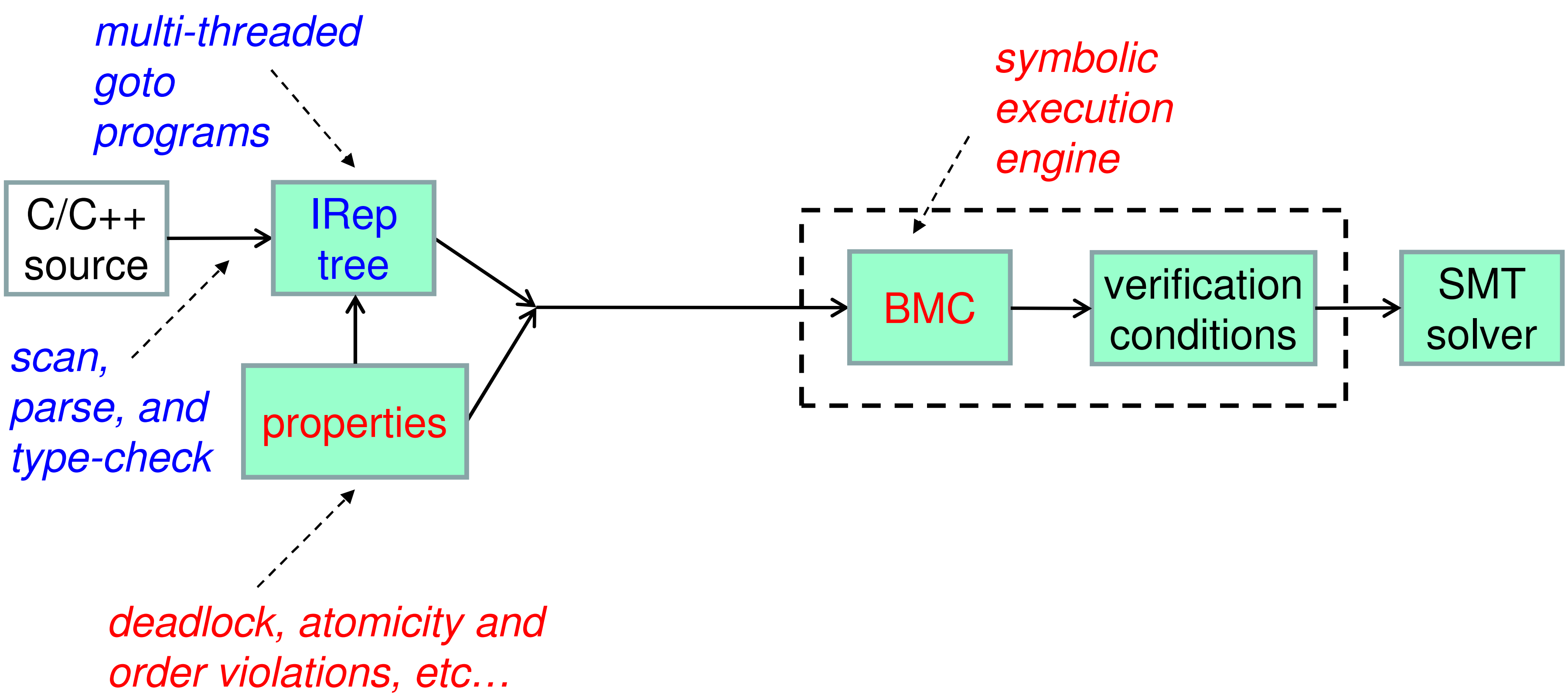
University of Southampton
School of Electronics and Computer Science

- 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

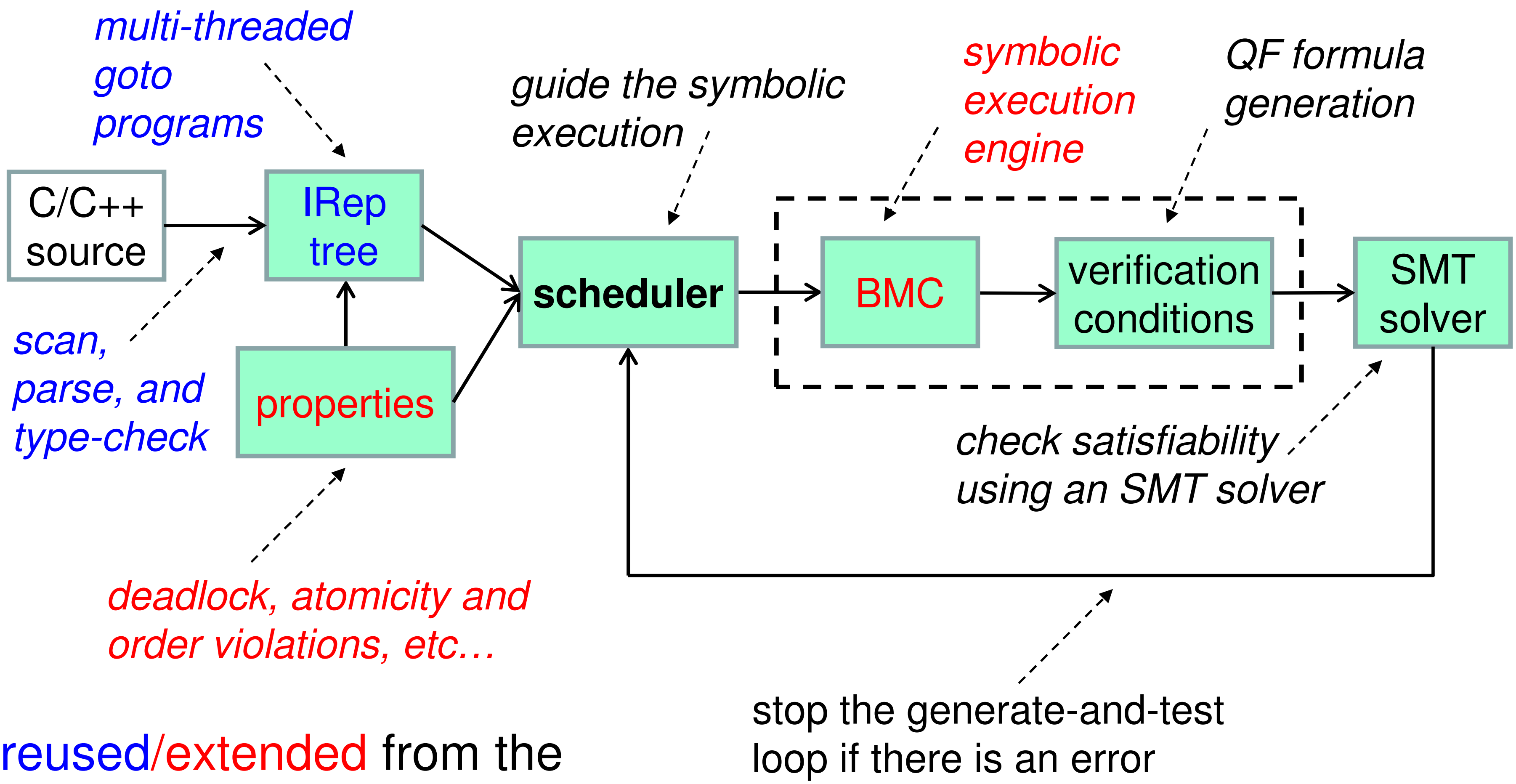


reused/extended from the Cprover framework

Lazy exploration of interleavings

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



reused/extended from the Cprover framework

Lazy exploration of interleavings

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Main steps of the algorithm:

1. Initialize the stack with the initial node ν_0 and the initial path $\pi_0 = \langle \upsilon_0 \rangle$
2. If the stack is empty, terminate with “no error”.
3. Pop the current node υ and current path π off the stack and compute the set υ' of successors of υ using rules R1-R8.
4. If υ' is empty, derive the VC φ_k^{π} for π and call the SMT solver on it. If φ_k^{π} is satisfiable, terminate with “error”; otherwise, goto step 2.
5. If υ' is not empty, then for each node $\upsilon \in \upsilon'$, add ν to π, and push node and extended path on the stack. goto step 3.

computation path

$\pi = \{\upsilon_1, \ldots \upsilon_n\}$

$$\varphi_k^{\pi} = \overbrace{I(s_0) \wedge R(s_0, s_1) \wedge \ldots \wedge R(s_{k-1}, s_k)}^{\text{constraints}} \wedge \overbrace{\neg \phi_k}^{\text{property}}$$

bound

Running Example

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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);
```

```
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

A state s ∈ S consists of the value of the program counter *pc* and the values of all program variables

program counter: 0
mutexes: m1=0; m2=0;
global variables: val1=0; val2=0;
local variabes: 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: 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 variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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 variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

statements: 1-2

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

val2-access:

> write access to the shared variable *val1* in statement *2* of the thread *twoStage*

```
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 variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

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: 3

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

global variables: val1=1; val2=0;

local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7

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);
```

CS1

```
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 variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

read access to the shared variable *val1* in statement *8* of the thread *reader*

statements: 1-2-3-7-8

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

val2-access:

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

CS1

```
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 variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);
```

CS1

```
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 variabes: **t1= 1**; t2= -1;*

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);
```

CS1

```
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 variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);
```

CS1 (3 → 7)

CS2 (12 → 4)

```
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 variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

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

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);
```

CS1: 3: unlock(m1); → 7: lock(m1);

CS2: 12: unlock(m1); → 4: lock(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 variabes: 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);
```

CS1

CS2

```
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 variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);
```

CS1

CS2

```
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 variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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));
```

CS1: 3 → 7
CS2: 12 → 4
CS3: 6 → 13

program counter: 13
mutexes: m1=0; m2=0;
global variables: val1=1; val2=2;
local variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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);
```

CS1

CS2

CS3

```
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 variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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

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

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

CS1 (3 → 7), CS2 (12 → 4), CS3 (6 → 13)

```
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 variabes: t1= 1; ***t2= 2****;*

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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

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

```
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));
```

CS1: 3 → 7

CS2: 12 → 4

CS3: 6 → 13

program counter: 15

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

global variables: val1=1; val2=2;

local variabes: 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: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

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

```
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));
```

CS1 (3 → 7), CS2 (12 → 4), CS3 (6 → 13)

program counter: 16

mutexes: m1=0; m2=0;

global variables: val1=1; val2=2;

local variabes: 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: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

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

```
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));
```

CS1: 3 → 7

CS2: 12 → 4

CS3: 6 → 13

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 variabes: 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: 3
mutexes: m1=0; m2=0;
global variables: ***val1=1****; val2=0;*
local variabes: 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);                CS1
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 variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_f

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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

val2-access: $R_{reader,14}$

```
Thread twoStage
1:  lock(m1);                CS1
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 variabes: **t1= 1**; **t2= 0**;*

Lazy exploration: interleaving I_f

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

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

val2-access: $R_{reader,14}$

```
Thread twoStage
1:  lock(m1);                CS1
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

CS2

```
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 variabes: 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: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

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

```
Thread twoStage
1:  lock(m1);                CS1
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));
```

CS2

program counter: 6

mutexes: m1=0; m2=0;

global variables: val1=1; ***val2=2****;*

local variabes: t1= 1; t2= 0;

Lazy exploration: interleaving I_f

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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

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

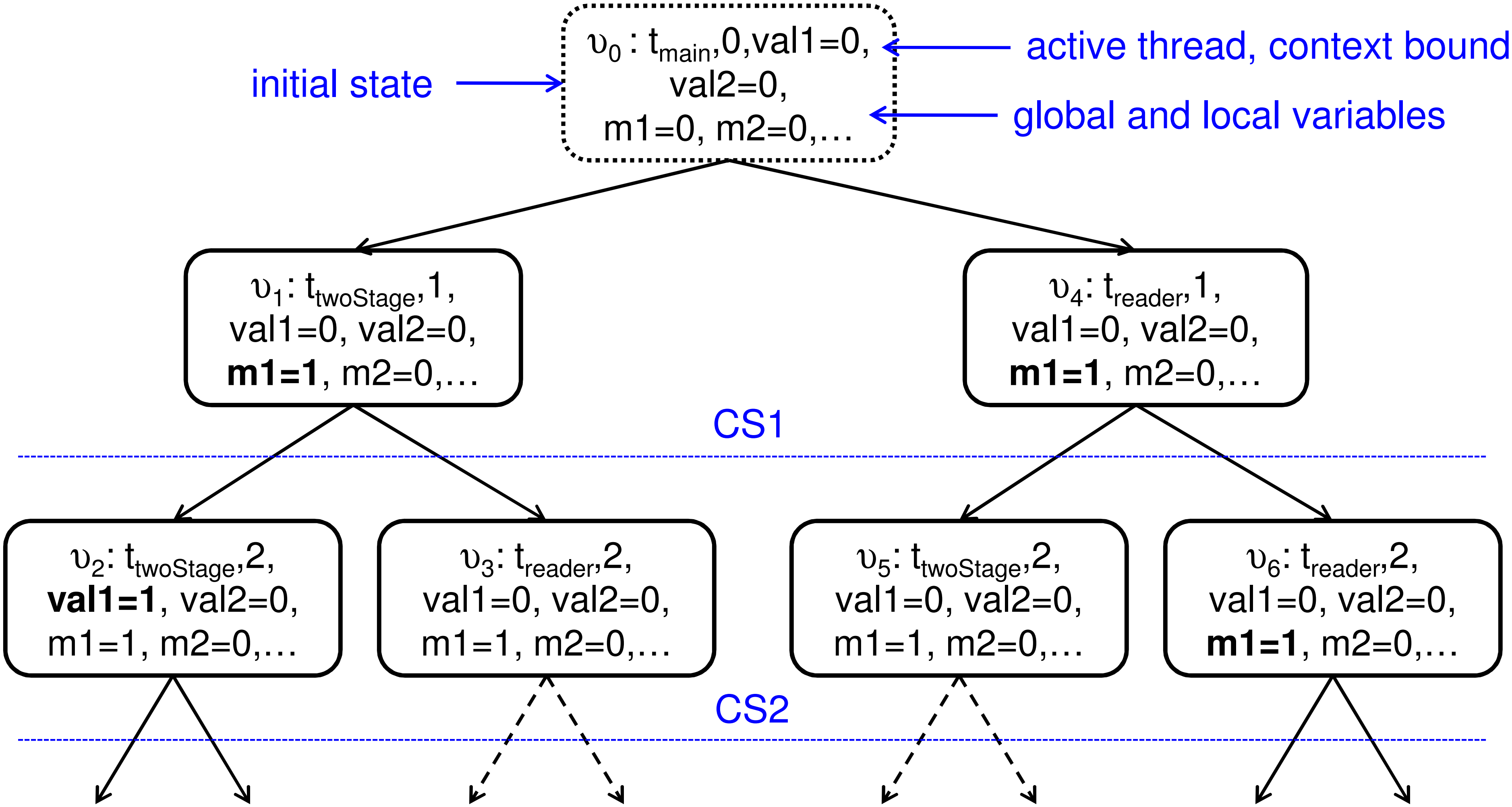
```
Thread twoStage
1:  lock(m1);                CS1
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

CS2

```
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



→ execution paths

- - - -> blocked execution paths (*eliminate*)

Exploring the Reachability Tree

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- use a reachability tree (RT) to describe reachable states of a multi-threaded program
- each node in the RT is a tuple $\upsilon = \left(A_i, C_i, s_i, \left\langle l_i^j, G_i^j \right\rangle_{j=1}^{n} \right)_i$ 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

$$l_{i+1}^{A_i} = l_i^{A_i} + 1$$

$$\upsilon' = \left(A_i, C_i, s_{i+1}, \left\langle l_{i+1}^{j}, G_i^{j} \right\rangle\right)_{i+1}$$

- we have fully expanded υ if
  - I within an atomic block; or
  - *l* contains no global variable; or
  - the upper bound of context switches ($C_i = C$) is reached
- if υ 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

$$\upsilon_j' = \left(j, C_i + 1, s_{i+1}, \left\langle l_i^{j}, G_i^{j} \right\rangle \right)_{i+1}$$

Expansion Rules of the RT

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

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

R3 (unconditional goto): If *I* 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:

$$v' = \left(A_i, C_i, s_i, \left\langle l_{i+1}^j, G_i^j \right\rangle\right)_{i+1} \qquad l_{i+1}^j = \begin{cases} l & : \quad j = A_i \\ l_i^j & : \quad otherwise \end{cases}$$

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 υ' and υ''.

- for υ' , we assume that *c* is *true* and proceed with the target instruction of the jump:

$$l_{i+1}^{j} = \begin{cases} l & : \quad j = A_i \\ l_i^j & : \quad otherwise \end{cases}$$

$$\upsilon' = \left(A_i, C_i, s_i, \left\langle l_{i+1}^{j}, c \wedge G_i^{j} \right\rangle\right)_{i+1}$$

- for υ'', we add ¬*c* to the guards and continue with the next instruction in the current thread

$$l_{i+1}^{j} = \begin{cases} l_i^j + 1 & : \quad j = A_i \\ l_i^j & : \quad otherwise \end{cases}$$

$$\upsilon'' = \left(A_i, C_i, s_i, \left\langle l_{i+1}^{j}, \neg c \wedge G_i^{j} \right\rangle\right)_{i+1}$$

- prune one of the nodes if the condition is determined statically

Expansion Rules of the RT

R5 (assume): If *I* is an *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 $c \wedge G_i^j$ evaluates to *false*, we prune the execution path

R6 (assert): If *I* is an *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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

$$\upsilon' = \left(A_i, C_i, s_i, \underline{\left\langle l_{i+1}^{j}, G_{i+1}^{j} \right\rangle}_{j=1}^{n+1} \right)_{i+1}$$

- 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 *I* is a *join_thread* instruction with argument *Id*, we add a child node:

$$\upsilon' = \left(A_i, C_i, s_i, \left\langle \underline{l_{i+1}^{j}}, G_i^{j} \right\rangle \right)_{i+1}$$

- where $l_{i+1}^{j} = l_i^{A_i} + 1$ only if the joining thread Id 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

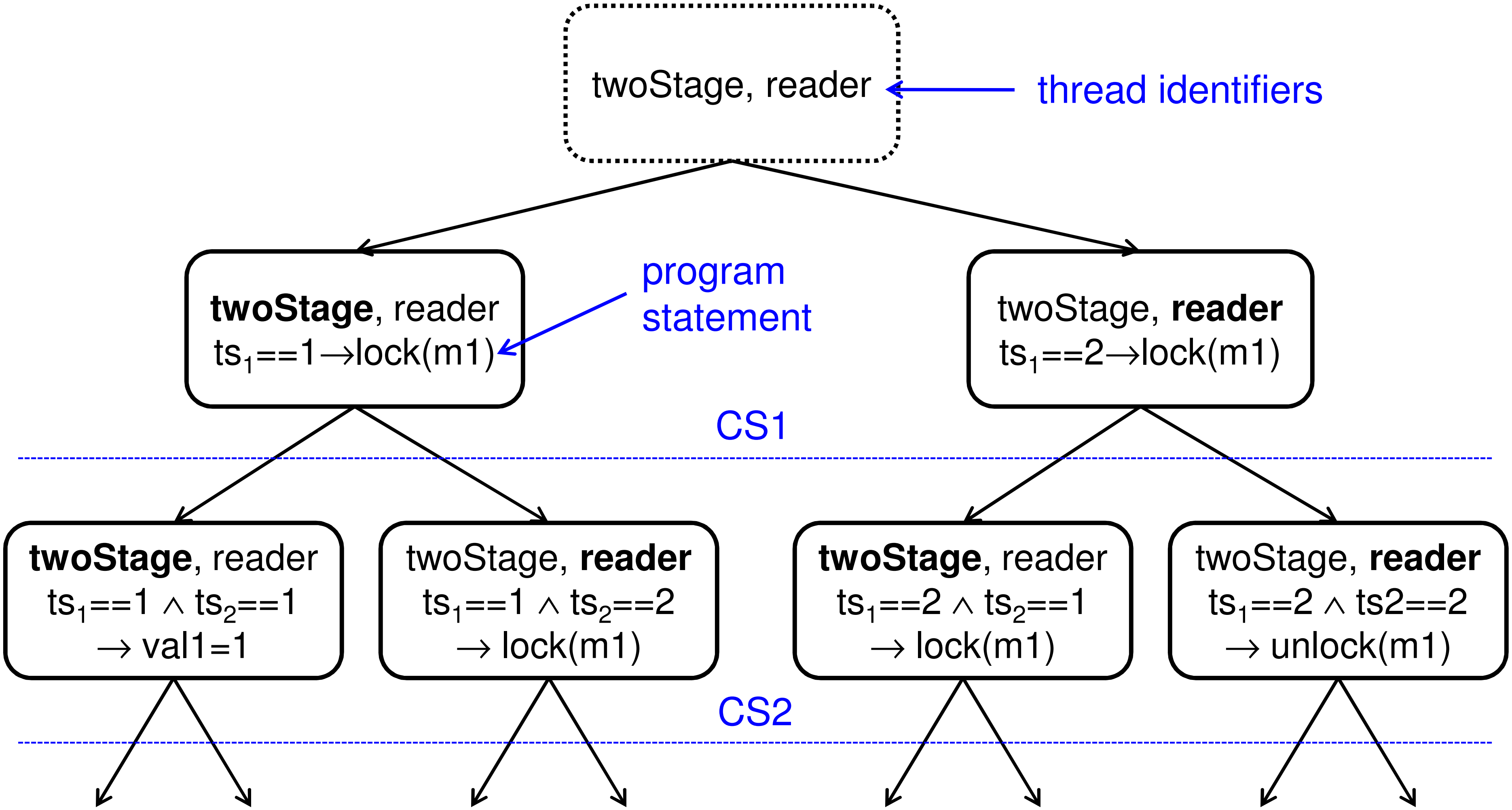
Schedule Recording

University of Southampton School of Electronics and Computer Science

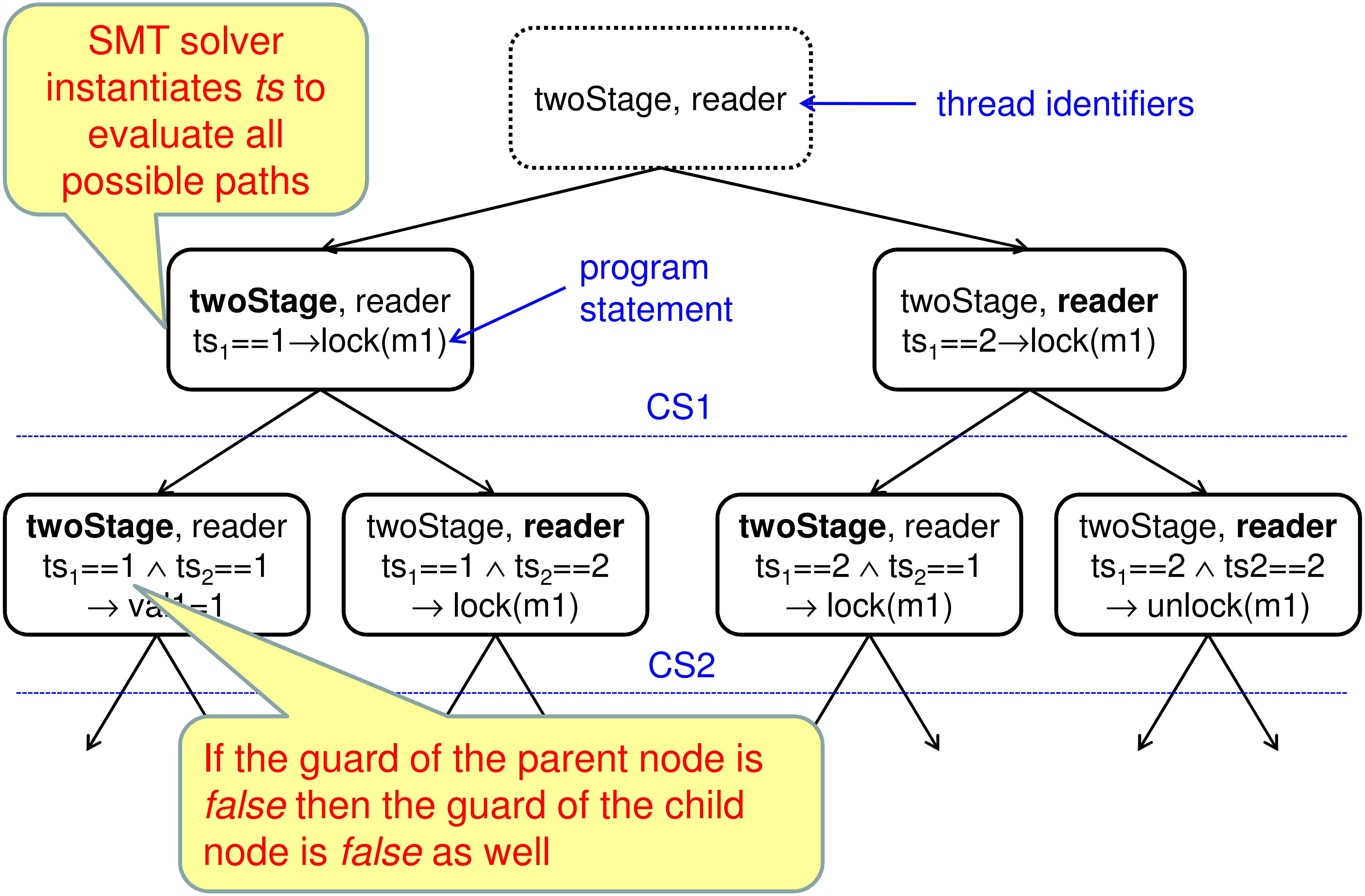
Idea: systematically encode all possible interleavings into one formula

- 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



Schedule Recording: Execution Paths



Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

statements: 1

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

reader-ECS:

guarded statement can only be executed if *statement 1* is scheduled in the *ECS block 1*

```
Thread twoStage
1: lock(m1);                ts1 == 1
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

```
Thread reader
7: lock(m1);
...
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

University of Southampton
School of Electronics and Computer Science

statements: 1-2

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

reader-ECS:

```
Thread twoStage
1: lock(m1);              ts1 == 1
2: val1 = 1;              ts2 == 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);            ts1 == 1
2: val1 = 1;            ts2 == 1
3: unlock(m1);          ts3 == 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);            ts1 == 1
2: val1 = 1;            ts2 == 1
3: unlock(m1);          ts3 == 1
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

CS

```
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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);              ts1 == 1
2: val1 = 1;              ts2 == 1
3: unlock(m1);            ts3 == 1
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

CS

```
Thread reader
7:  lock(m1);                     ts4 == 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);              ts1 == 1
2: val1 = 1;              ts2 == 1
3: unlock(m1);            ts3 == 1
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

CS

```
Thread reader
7:  lock(m1);                 ts4 == 2
8:  if (val1 == 0) {          ts5 == 2
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

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);              ts1 == 1
2: val1 = 1;              ts2 == 1
3: unlock(m1);            ts3 == 1
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

CS

```
Thread reader
7:  lock(m1);               ts4 == 2
8:  if (val1 == 0) {        ts5 == 2
9:     unlock(m1);
10:  return NULL; }
11: t1 = val1;              ts6 == 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);              ts1 == 1
2: val1 = 1;              ts2 == 1
3: unlock(m1);            ts3 == 1
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

CS

```
Thread reader
7:  lock(m1);                 ts4 == 2
8:  if (val1 == 0) {          ts5 == 2
9:    unlock(m1);
10:   return NULL; }
11: t1 = val1;                ts6 == 2
12: unlock(m1);               ts7 == 2
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

Schedule Recording: Interleaving I_s

University of Southampton
School of Electronics and Computer Science

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);              ts1 == 1
2: val1 = 1;              ts2 == 1
3: unlock(m1);            ts3 == 1
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

CS

CS

```
Thread reader
7:  lock(m1);                    ts4 == 2
8:  if (val1 == 0) {             ts5 == 2
9:     unlock(m1);
10:   return NULL; }
11: t1 = val1;                   ts6 == 2
12: unlock(m1);                  ts7 == 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

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);             ts1 == 1
2: val1 = 1;             ts2 == 1
3: unlock(m1);           ts3 == 1
4: lock(m2);             ts8 == 1
5: val2 = val1 + 1;
6: unlock(m2);
```

CS

CS

```
Thread reader
7:  lock(m1);                  ts4 == 2
8:  if (val1 == 0) {           ts5 == 2
9:     unlock(m1);
10:   return NULL; }
11: t1 = val1;                 ts6 == 2
12: unlock(m1);                ts7 == 2
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

Schedule Recording: Interleaving I_s

University of Southampton
School of Electronics and Computer Science

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);              ts1 == 1
2: val1 = 1;              ts2 == 1
3: unlock(m1);            ts3 == 1
4: lock(m2);              ts8 == 1
5: val2 = val1 + 1;       ts9 == 1
6: unlock(m2);
```

CS

CS

```
Thread reader
7:  lock(m1);             ts4 == 2
8:  if (val1 == 0) {      ts5 == 2
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;            ts6 == 2
12: unlock(m1);           ts7 == 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);            ts1 == 1
2: val1 = 1;            ts2 == 1
3: unlock(m1);          ts3 == 1
4: lock(m2);            ts8 == 1
5: val2 = val1 + 1;     ts9 == 1
6: unlock(m2);          ts10== 1
```

CS

CS

```
Thread reader
7:  lock(m1);                 ts4 == 2
8:  if (val1 == 0) {          ts5 == 2
9:    unlock(m1);
10:   return NULL; }
11: t1 = val1;                ts6 == 2
12: unlock(m1);               ts7 == 2
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);            ts1 == 1
2: val1 = 1;            ts2 == 1
3: unlock(m1);          ts3 == 1
4: lock(m2);            ts8 == 1
5: val2 = val1 + 1;     ts9 == 1
6: unlock(m2);          ts10== 1
```

CS

CS

CS

```
Thread reader
7:  lock(m1);                ts4 == 2
8:  if (val1 == 0) {         ts5 == 2
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;               ts6 == 2
12: unlock(m1);              ts7 == 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);          ts1 == 1
2: val1 = 1;          ts2 == 1
3: unlock(m1);        ts3 == 1
4: lock(m2);          ts8 == 1
5: val2 = val1 + 1;   ts9 == 1
6: unlock(m2);        ts10== 1
```

CS

CS

CS

```
Thread reader
7:  lock(m1);                ts4 == 2
8:  if (val1 == 0) {         ts5 == 2
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;               ts6 == 2
12: unlock(m1);              ts7 == 2
13: lock(m2);                ts11== 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);	$ts_{10} == 1$

CS

CS

CS

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);	$ts_{11} == 2$
14: t2 = val2;	$ts_{12} == 2$
15: unlock(m2);	
16: assert(t2==(t1+1));	

Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

```
Thread twoStage
1: lock(m1);            ts1 == 1
2: val1 = 1;            ts2 == 1
3: unlock(m1);          ts3 == 1
4: lock(m2);            ts8 == 1
5: val2 = val1 + 1;     ts9 == 1
6: unlock(m2);          ts10== 1
```

CS

CS

CS

```
Thread reader
7:  lock(m1);                ts4 == 2
8:  if (val1 == 0) {         ts5 == 2
9:     unlock(m1);
10:   return NULL; }
11: t1 = val1;               ts6 == 2
12: unlock(m1);              ts7 == 2
13: lock(m2);                ts11== 2
14: t2 = val2;               ts12== 2
15: unlock(m2);              ts13== 2
16: assert(t2==(t1+1));
```

Schedule Recording: Interleaving I_s

Southampton School of Electronics and Computer Science

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);	$ts_{10} == 1$

CS

CS

CS

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);	$ts_{11} == 2$
14: t2 = val2;	$ts_{12} == 2$
15: unlock(m2);	$ts_{13} == 2$
16: assert(t2==(t1+1));	$ts_{14} == 2$

Schedule Recording: Interleaving I_f

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);              ts1 == 1
2: val1 = 1;              ts2 == 1
3: unlock(m1);            ts3 == 1
4: lock(m2);              ts12 == 1
5: val2 = val1 + 1;       ts13 == 1
6: unlock(m2);            ts14 == 1
```

CS

CS

```
Thread reader
7:  lock(m1);                   ts4 == 2
8:  if (val1 == 0) {            ts5 == 2
9:    unlock(m1);
10:   return NULL; }
11: t1 = val1;                  ts6 == 2
12: unlock(m1);                 ts7 == 2
13: lock(m2);                   ts8 == 2
14: t2 = val2;                  ts9 == 2
15: unlock(m2);                 ts10 == 2
16: assert(t2==(t1+1));         ts11 == 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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 ($cl_{i,j}$) into the verification condition ψ
   - ▹ $cl_{i,j}$ where *i* is the ECS block number and *j* is the thread identifier
2. check the satisfiability of ψ (stop if ψ 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

Thread twoStage	
1: lock(m1);	$cl_{1,twoStage} \rightarrow ts_1 == 1$
2: val1 = 1;	$cl_{2,twoStage} \rightarrow ts_2 == 1$
3: unlock(m1);	$cl_{3,twoStage} \rightarrow ts_3 == 1$
4: lock(m2);	$cl_{8,twoStage} \rightarrow ts_8 == 1$
5: val2 = val1 + 1;	$cl_{9,twoStage} \rightarrow ts_9 == 1$
6: unlock(m2);	$cl_{10,twoStage} \rightarrow ts_{10} == 1$

- reduce the number of control points from *m* x *n* to *e* x *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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

	Module	#L	#T	#P	B	#C	
		81	26	7	26	2	Fra
2	fsbench_bad		27		27	2	Fr... file system with array
3	in				29	4	...nto a hash table concurrently
4	aget-0.4_bad	1233					...eaded download ...tor
5	bzip2smp_ok	6366					...mpressor
6	reorder_bad	84	10	7	10	11	Contains a data race
7	twostage_bad	128	100	13	100	4	Contains an atomicity violation
8	wronglock_bad	110	8	8	8	8	Contains wrong lock acquisition ordering
9	exStbHDMI_ok	1060	2	24	16	20	Configures the HDMI device
10	exStbLED_ok	425	2	45	10	10	Front panel LED display
11	exStbThumb_bad	1109	2	249	2	1	Demonstrate how thumbnail images can be manipulated
12	micro_10_ok	1171	10	10	1	17	synthetic micro-benchmark

lines of code

number of properties checked

the number of BMC unrolling steps

number of threads

number of context switches

About the benchmarks

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Inspect benchmark suite

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

About the benchmarks

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

VV-lab benchmark suite

About the benchmarks

UNIVERSITY OF Southampton
School of Electronics and Computer Science

	Module	#L	#T	#P	B	#C	Description
1	fsbench_ok	81	26	47	26	2	Frangipani file system
2	fsbench_bad	80	27	48	27	2	Frangipani file system with array out of bounds
3	indexer_ok	77	13	21	129	4	Insert messages into a hash table concurrently
4	aget-0.4_bad	1233	3	279	200	2	Multi-threaded download accelerator
5	bzip2smp_ok	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	ata compressor
6	reorder_bad	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	ontains a data race
7	twostage_bad	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	ontains an atomicity violation
8	wronglock_bad	[illegible]	[illegible]	8	8	8	ontains wrong lock acquisition ordering
9	**exStbHDMI_ok**	[illegible]	**2**	**24**	**16**	**20**	**Configures the HDMI device**
10	**exStbLED_ok**	**425**	**2**	**45**	**10**	**10**	**Front panel LED display**
11	**exStbThumb_bad**	**1109**	**2**	**249**	**2**	**1**	**Demonstrate how thumbnail images can be manipulated**
12	micro_10_ok	1171	10	10	1	17	synthetic micro-benchmark

Set-top box applications from NXP semiconductors

About the benchmarks

	Module	#L	#T	#P	B	#C	Description
1	fsbench_ok	81	26	47	26	2	Frangipani file system
2	fsbench_bad	80	27	48	27	2	Frangipani file system with array out of bounds
3	indexer_ok	77	13	21	129	4	Insert messages into a hash table concurrently
4	aget-0.4_bad	1233	3	279	200	2	Multi-threaded download accelerator
5	bzip2smp_ok	6366	3	8568	1	9	Data compressor
6	reorder_bad	84	10	7	10	11	Contains a data race
7	twostage_bad	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...n atomicity violation
8	wronglock_bad	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...rong lock acquisition
9	exStbHDMI_ok	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	... the HDMI device
10	exStbLED_ok	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...l LED display
11	exStbThumb_bad	11...	...2	249	2	1	Demonstrate how thumbnail images can be manipulated
12	**micro_10_ok**	**1171**	**10**	**10**	**1**	**17**	**synthetic micro-benchmark**

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

Comparison of the appro[illegible]

University of Southampton
School of Electronics and Computer Science

encoding and solver time

number of generated and failed interleavings

Module	Lazy			[illegible]		[illegible]W		
	Time	Result	#FI/#I	Time	Result	Time	Result	Iter
fsbench_ok	[illegible]	[illegible]	[illegible]	[illegible]	+	301	[illegible]	1
fsbench_bad	[illegible]	[illegible]	[illegible]	[illegible]	+	[illegible]	[illegible]	2
indexer_ok	[illegible]	[illegible]	[illegible]	[illegible]	+	[illegible]	[illegible]	1
aget-0.4_bad	[illegible]	[illegible]	[illegible]	[illegible]	+	125	+	1
bzip2smp_ok	1800	+	0/1294	MO	-	MO	-	1
reorder_bad	<1	+	1/154574	MO	-	MO	-	1
twostage_bad	88	+	1/139	93	+	195	+	5
wronglock_bad	90	+	6/104015	MO	-	MO	-	1
exStbHDMI_ok	229	+	0/1	226	+	213	+	1
exStbLED_ok	73	+	0/11	73	+	787	+	1
exStbThumb_bad	95	+	3/3	14	+	12	+	1
micro_10_ok	254	+	0/29260	MO	-	MO	-	1

error detected in module "+" **GOOD THING**

error occurred in tool "–" **BAD THING**

number of iterations

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Comparison of the approaches (1)

lazy encoding often more efficient than schedule recording and UW

	Lazy			Schedule		UW		
	…ne	Result	#FI/#I	Time	Result	Time	Result	Iter
fsben…k	**282**	+	0/676	**304**	+	**301**	+	1
fsbench_bad	**<1**	+	729/729	**360**	+	**786**	+	2
indexer_ok	595	+	0/17160	220	+	218	+	1
aget-0.4_bad	137	+	1/1	127	+	125	+	1
bzip2smp_ok	**1800**	+	0/1294	**MO**	-	**MO**	-	1
reorder_bad	**<1**	+	1/154574	**MO**	-	**MO**	-	1
twostage_bad	**88**	+	1/139	**93**	+	**195**	+	5
wronglock_bad	**90**	+	6/104015	**MO**	-	**MO**	-	1
exStbHDMI_ok	229	+	0/1	226	+	213	+	1
exStbLED_ok	73	+	0/11	73	+	787	+	1
exStbThumb_bad	95	+	3/3	14	+	12	+	1
micro_10_ok	**254**	+	0/29260	**MO**	-	**MO**	-	1

Comparison of the approaches (2)

University of Southampton
School of Electronics and Computer Science

lazy encoding often more efficient than schedule recording and UW, ***but not always***

	azy			Schedule		UW		
	[illegible]	sult	#FI/#I	Time	Result	Time	Result	Iter
[illegible]	[illegible]	+	0/676	304	+	301	+	1
fsbench_[illegible]	<1	+	729/729	360	+	786	+	2
indexer_ok	**595**	+	0/17160	**220**	+	**218**	+	1
aget-0.4_bad	**137**	+	1/1	**127**	+	**125**	+	1
bzip2smp_ok	1800	+	0/1294	MO	-	MO	-	1
reorder_bad	<1	+	1/154574	MO	-	MO	-	1
twostage_bad	88	+	1/139	93	+	195	+	5
wronglock_bad	90	+	6/104015	MO	-	MO	-	1
exStbHDMI_ok	**229**	+	0/1	**226**	+	**213**	+	1
exStbLED_ok	**73**	+	0/11	**73**	+	**787**	+	1
exStbThumb_bad	**95**	+	3/3	**14**	+	**12**	+	1
micro_10_ok	254	+	0/29260	MO	-	MO	-	1

... the approaches (3)

lazy encoding is extremely fast for ***satisfiable instances***

Module	Lazy			Schedule		UW		
	Time	Result	#FI/#I	Time	Result	Time	Result	Iter
fsbench_ok	282	+	0/676	304	+	301	+	1
fsbench_bad	**<1**	+	729/729	**360**	+	**786**	+	2
indexer_ok	595	+	0/17160	220	+	218	+	1
aget-0.4_bad	137	+	1/1	127	+	125	+	1
bzip2smp_ok	1800	+	0/1294	MO	-	MO	-	1
reorder_bad	**<1**	+	1/154574	**MO**	-	**MO**	-	1
twostage_bad	**88**	+	1/139	**93**	+	**195**	+	5
wronglock_bad	**90**	+	6/104015	**MO**	-	**MO**	-	1
exStbHDMI_ok	229	+	0/1	226	+	213	+	1
exStbLED_ok	73	+	0/11	73	+	787	+	1
exStbThumb_bad	95	+	3/3	14	+	12	+	1
micro_10_ok	254	+	0/29260	MO	-	MO	-	1

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Comparison to CHESS [Musuvathi and Qadeer]

UNIVERSITY OF Southampton School of Electronics and Computer Science

- 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

ESS [Musuvathi and Qadeer]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

		B	C	CHESS		Lazy	
				Time	Tests	Time	#FI/#I
reorder_4_bad (3,1)	4	4	5	**98**	130000	**<1**	1/82
reorder_5_bad (4,1)	5	5	6	TO	429000	<1	1/277
reorder_6_bad (5,1)	6	6	7	TO	396000	<1	1/853
reorder_6_bad (5,1)	6	6	8	TO	371000	<1	1/2810
reorder_6_bad (5,1)	6	6	9	TO	367000	<1	1/8124
twostage_4_bad (3,1)	4	4	4	**215**	27000	**2**	1/42
twostage_5_bad (4,1)	5	5	4	TO	384000	2	1/44
twostage_6_bad (5,1)	6	6	4	TO	366000	2	1/45
wronglock_4_bad (1,3)	4	4	8	**21**	3000	**5**	2/489
wronglock_5_bad (1,4)	5	5	8	**724**	93000	**10**	3/2869
wronglock_6_bad (1,5)	6	6	8	TO	356000	18	4/12106
micro_2_ok (100)	2	1	2	**316**	35855	**<1**	0/4
micro_2_ok (100)	2	1	17	TO	40000	1095	0/131072

…SS [Musuvathi and Qadeer]

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**

				CHESS		Lazy	
				Time	Tests	Time	#FI/#I
			5	98	130000	<1	1/82
reorder_5_bad (4,1)	5	5	6	TO	429000	<1	1/277
reorder_6_bad (5,1)	6	6	7	TO	396000	<1	1/853
reorder_6_bad (5,1)	6	6	8	TO	371000	<1	1/2810
reorder_6_bad (5,1)	6	6	9	TO	367000	<1	1/8124
twostage_4_bad (3,1)	4	4	4	215	27000	2	1/42
twostage_5_bad (4,1)	5	5	4	TO	384000	2	1/44
twostage_6_bad (5,1)	6	6	4	TO	366000	2	1/45
wronglock_4_bad (1,3)	4	4	8	21	3000	5	2/489
wronglock_5_bad (1,4)	5	5	8	724	93000	10	3/2869
wronglock_6_bad (1,5)	6	6	8	TO	356000	18	4/12106
micro_2_ok (100)	2	1	2	316	35855	<1	0/4
micro_2_ok (100)	2	1	17	TO	40000	1095	0/131072

Comparison to SATABS [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Comparison to SATABS [D. Kroening]

failed to validate the counterexample

	SATABS		Lazy		
	Time	Result	Time	Result	#FI/#I
fsbench_ok	†	-	**282**	+	0/676
fsbench_bad	†	-	**<1**	+	729/729
indexer_ok	TO	-	595	+	0/17160
aget-0.4_bad	3346	+	137	+	1/1
bzip2smp_ok	TO	-	1800	+	0/1294
[illegible]	1	-	<1	+	1/154574
[illegible]	2	-	88	+	1/139
wrongloc[illegible]d	2	-	90	+	6/104015
exStbHDMI_ok	TO	-	229	+	0/1
exStbLED_ok	RF	-	**73**	+	0/11
exStbThumb_bad	317	+	95	+	3/3
micro_10_ok	TO	-	254	+	0/29260

failed to refine the predicate

Comparison to SATABS [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	SATABS		Lazy		
	Time	Result	Time	Result	#FI/#I
fsbench_ok	†	-	282	+	0/676
fsbench_bad	†	-	<1	+	729/729
[illegible]	TO	-	595	+	0/17160
[illegible]	3346	+	137	+	1/1
bzip2smp_[illegible]	TO	-	1800	+	0/1294
reorder_bad	1	-	**<1**	+	1/154574
twostage_bad	2	-	**88**	+	1/139
wronglock_bad	2	-	**90**	+	6/104015
exStbHDMI_ok	TO	-	229	+	0/1
exStbLED_ok	RF	-	73	+	0/11
exStbThumb_bad	317	+	95	+	3/3
micro_10_ok	TO	-	254	+	0/29260

false positives answers

TABS [D. Kroening]

SATABS uses predicate abstraction and refinement and tries to solve a harder problem than ESBMC

		TABS		Lazy	
		Result	Time	Result	#FI/#I
fsbench_ok	†	-	282	+	0/676
fsbench_bad	†	-	<1	+	729/729
indexer_ok	TO	-	595	+	0/17160
aget-0.4_bad	3346	+	**137**	+	1/1
bzip2smp_ok	TO	-	1800	+	0/1294
reorder_bad	1	-	<1	+	1/154574
twostage_bad	2	-	88	+	1/139
wronglock_bad	2	-	90	+	6/104015
exStbHDMI_ok	TO	-	229	+	0/1
exStbLED_ok	RF	-	73	+	0/11
exStbThumb_bad	317	+	**95**	+	3/3
micro_10_ok	TO	-	254	+	0/29260

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**

. Kroening]

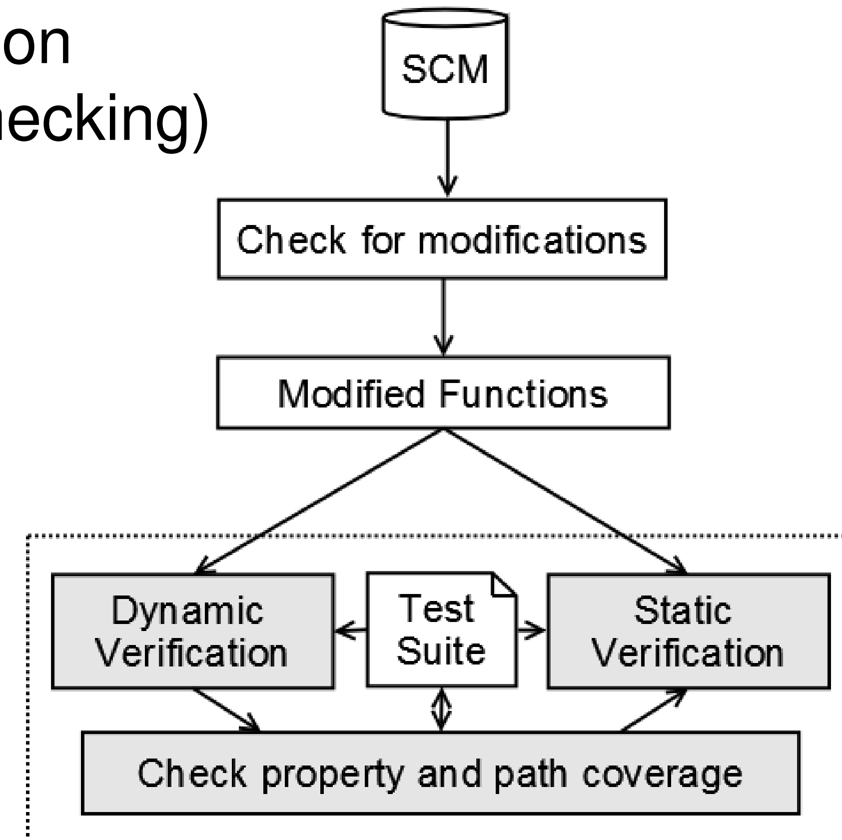
			Lazy		
			Time	Result	#FI/#I
fsbench_ok		-	282	+	0/676
fsbench_bad	1	-	<1	+	729/729
indexer_ok	TO	-	**595**	+	0/17160
aget-0.4_bad	3346	+	137	+	1/1
bzip2smp_ok	TO	-	**1800**	+	0/1294
reorder_bad	1	-	<1	+	1/154574
twostage_bad	2	-	88	+	1/139
wronglock_bad	2	-	90	+	6/104015
exStbHDMI_ok	TO	-	**229**	**+**	0/1
exStbLED_ok	RF	-	73	+	0/11
exStbThumb_bad	317	+	95	+	3/3
micro_10_ok	TO	-	**254**	+	0/29260

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

```
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

```
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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

```
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;
}
```

$$\alpha_1 = \begin{bmatrix} inverter_1 = signal_1 \\ \wedge\, inverter_2 = -1 * signal_1 \\ \wedge\, inverter_3 = \left(signal_1 \geq 0\, ?\, inverter_1 : inverter_2\right) \end{bmatrix}$$

$$\alpha_2 = \left[signal'_2 = \left(signal'_1 < 0\, ?\, -signal'_1 : signal'_1\right)\right]$$

Functional Equivalence Checking

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

SSA of function 1 and 2

$$(\alpha_1 \wedge \alpha_2 \wedge (signal_1 = signal'_1)) \rightarrow (inverter_3 = signal'_2)$$

inputs

outputs

Functional Equivalence Checking

University of Southampton
School of Electronics and Computer Science

- 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

SSA of function 1 and 2

global variables

$$(\alpha_1 \wedge \alpha_2 \wedge (signal_1 = signal'_1)) \rightarrow (inverter_3 = signal'_2) \wedge (g_1 = g_1^{'})$$

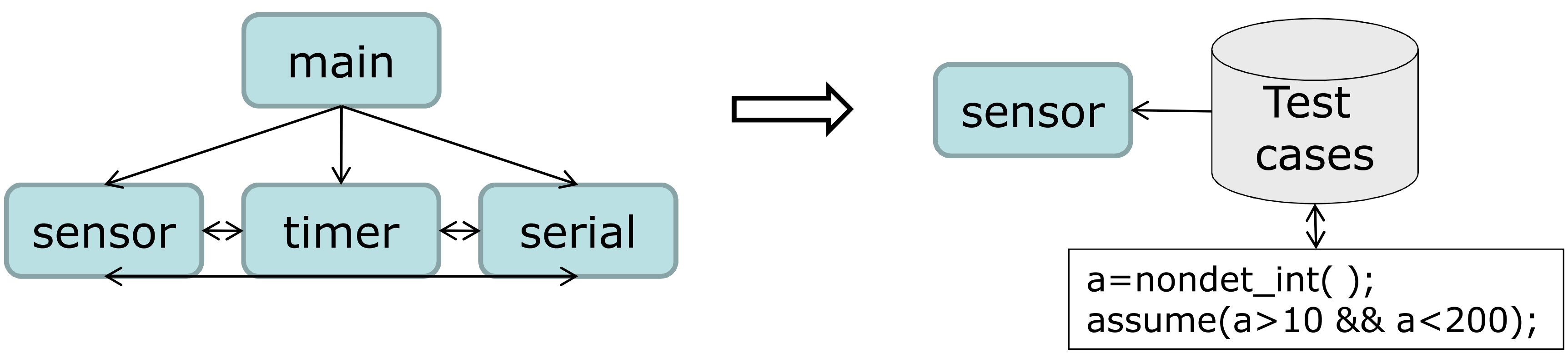
inputs

outputs

Generalizing Test Cases

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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



Generalizing Test Cases: Example

Simple circular FIFO buffer:

```
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

```
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:

```
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:

```
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!

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 && sendData[0] >= 42);
  assume(senData[1]<=-128 && senData[1]>=-28);
  ...
  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());
}
```

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

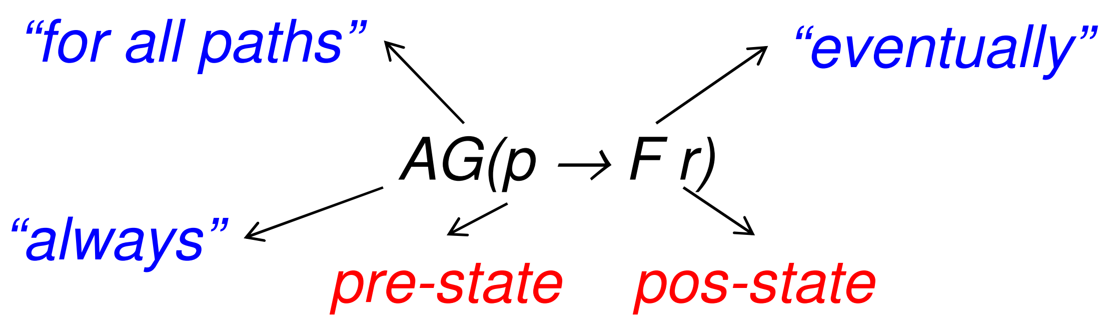
- force the model checker towards the “unobvious” errors

⇒ detects two bugs related to arithmetic over- and underflow

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Specifying Temporal Properties

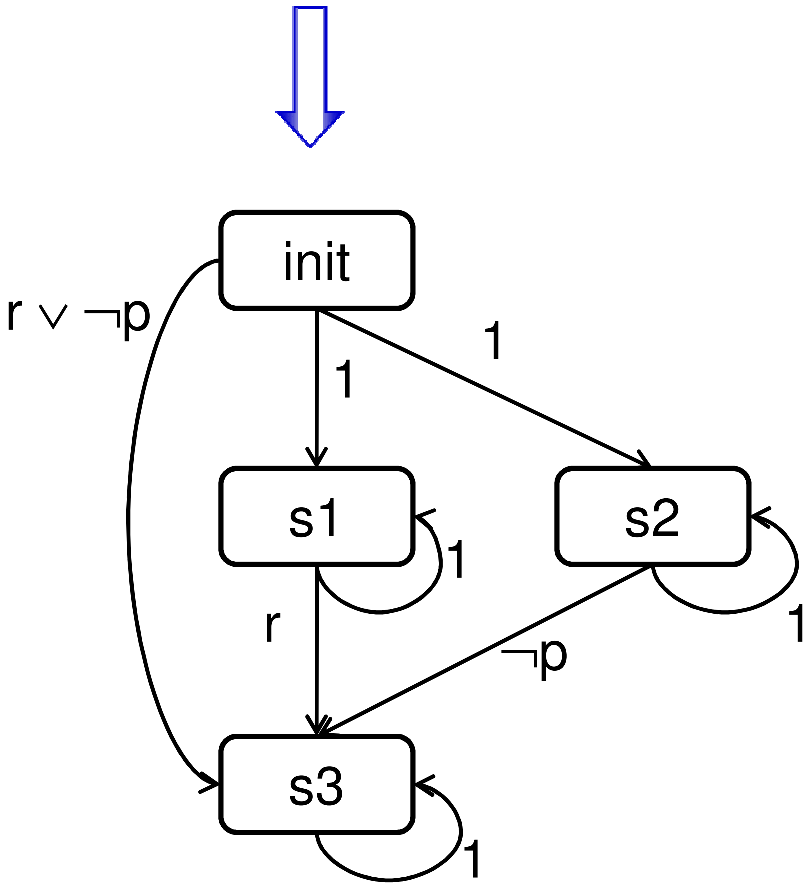
- we translate the LTL formulae into Buechi Automata (BA) and further into ANSI-C
  - monitor the design's progress and watch out for violations
- we extract two properties of the pulse oximeter device:
  a) verify the data flow to compute the HR value that is provided by the sensor
  b) verify whether 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

University of Southampton
School of Electronics and Computer Science

$AG(p \rightarrow F\ r)$




```
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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

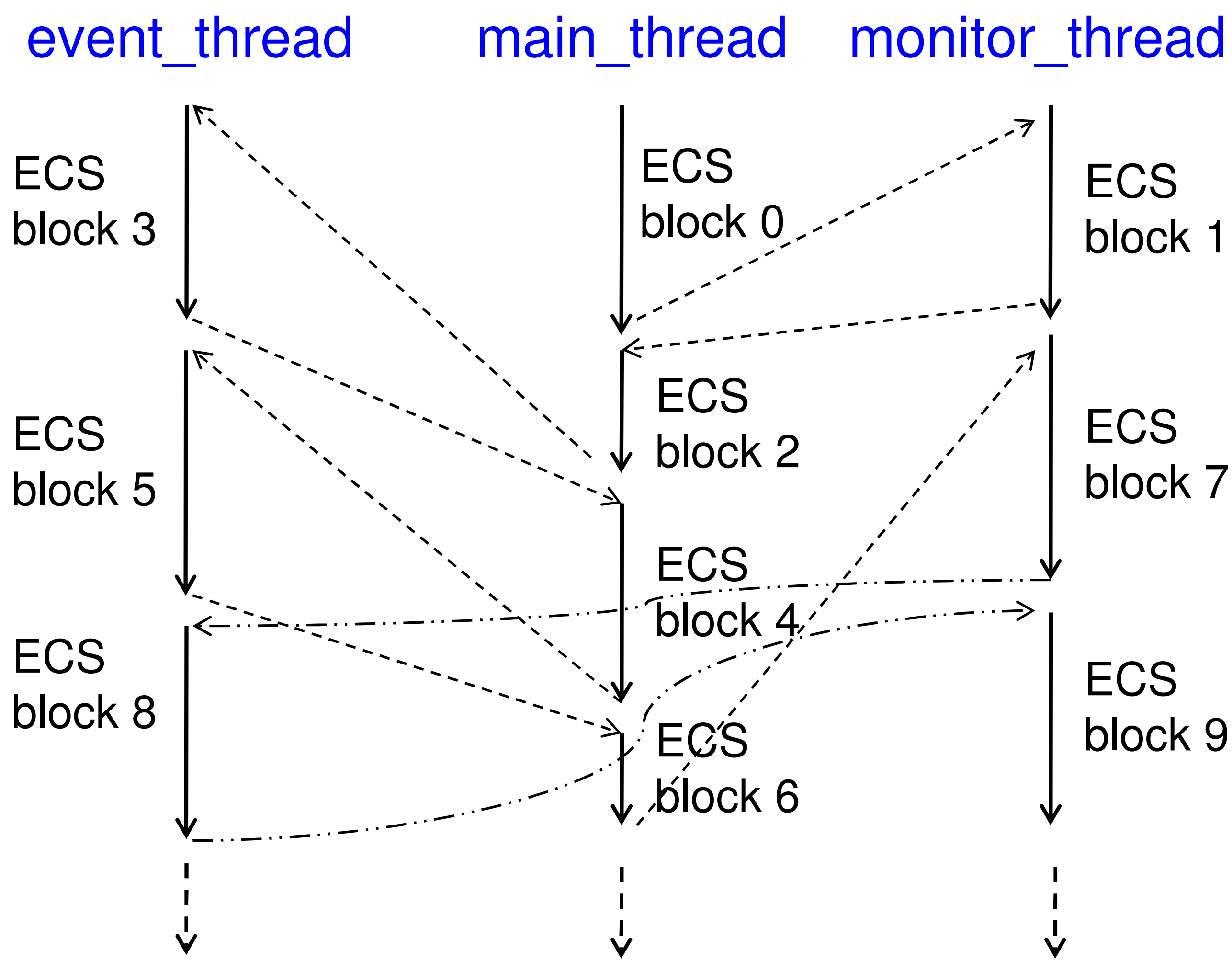
```
void monitor_thread(void* arg)
  …
  while(1) {
    choice = nondet_bool();
     if (p) flag=true;
     switch (state) {
      case init:
        if (r || !p) state=s3;
        …
        break;
      case s1:
        …
        break;
      …
    }
    if (flag && !is_processing)
      assert(state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

model the hardware interrupt and interacts with the pulse oximeter

```
bool is_processing = false;
…
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 occured

Concurrent Execution of Main, Monitor and event Threads



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

Test Program	L	B	P	VC	Time
commandLoop.TC1	545	-	18	0	4
commandLoop.TC2	545	500*	18	3	29
commandLoop.TC3	545	500*	18	3	29
commandLoop.TC4	545	17	18	5	14
commandLoop.TC5	545	-	18	1	4
commandLoop.TC6	545	-	18	0	4
commandLoop.TC7	545	1	18	15	19
checkCommandParams.TC1	238	17	17	56	9
checkCommandParams.TC2	238	17	17	36	5
checkCommandParams.TC3	238	17	17	37	5
checkCommandParams.TC4	238	17	17	36	30
checkCommandParams.TC5	238	17	17	80	50
checkCommandParams.TC6	238	17	17	664	44
checkCommandParams.TC7	238	20*	17	1117	215

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Verification of the Test Cases

ESBMC fails to verify these functions due to memory limitations and time-outs

Test Program	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
commandLoop.TC1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
commandLoop.TC2	[illegible]	500*	18	3	29
commandLoop.TC3	545	500*	18	3	29
commandLoop.TC4	545	17	18	5	14
commandLoop.TC5	545	-	18	1	4
commandLoop.TC6	545	-	18	0	4
commandLoop.TC7	545	1	18	15	19
checkCommandParams.TC1	238	17	17	56	9
checkCommandParams.TC2	238	17	17	36	5
checkCommandParams.TC3	238	17	17	37	5
checkCommandParams.TC4	238	17	17	36	30
checkCommandParams.TC5	238	17	17	80	50
checkCommandParams.TC6	238	17	17	664	44
checkCommandParams.TC7	238	20*	17	1117	215

Verification of the Test Cases

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

[illegible]	L	B	P	VC	Time
[illegible]	545	-	18	0	4
[illegible]	545	**500***	18	3	29
[illegible]	545	**500***	18	3	29
commandLoop.TC4	545	**17**	18	5	14
commandLoop.TC5	545	**-**	18	1	4
commandLoop.TC6	545	**-**	18	0	4
commandLoop.TC7	545	**1**	18	15	19
checkCommandParams.TC1	238	**17**	17	56	9
checkCommandParams.TC2	238	**17**	17	36	5
checkCommandParams.TC3	238	**17**	17	37	5
checkCommandParams.TC4	238	**17**	17	36	30
checkCommandParams.TC5	238	**17**	17	80	50
checkCommandParams.TC6	238	**17**	17	664	44
checkCommandParams.TC7	238	**20***	17	1117	215

Verification of the Test Cases

ESBMC is not able to prove or falsify some of the properties due to unwinding violations

	L	B	P	VC	Time
[illegible]	545	-	18	0	4
[illegible]	545	**500***	18	3	29
[illegible]	545	**500***	18	3	29
commandLoop.TC4	545	17	18	5	14
commandLoop.TC5	545	-	18	1	4
commandLoop.TC6	545	-	18	0	4
commandLoop.TC7	545	1	18	15	19
checkCommandParams.TC1	238	17	17	56	9
checkCommandParams.TC2	238	17	17	36	5
checkCommandParams.TC3	238	17	17	37	5
checkCommandParams.TC4	238	17	17	36	30
checkCommandParams.TC5	238	17	17	80	50
checkCommandParams.TC6	238	17	17	664	44
checkCommandParams.TC7	238	**20***	17	1117	215

Equivalence Checking

UNIVERSITY OF Southampton
School of Electronics and Computer Science

					Product Releases			
Test Program	L	B	P	Time	PR10	PR11	PR12	PR13
threadRename	6	17	0	3	X			
fileExists	19	17	0	3	X			
readLine	27	17	11	3	X			
getCommand	269	17	61	3	X	N/3		N/3
powerDown	9	17	0	2	X			
digitStart	12	17	0	2	X	Y/2		
difgitAdd	34	17	2	2	X	Y/2		
checkEndOfPvrStream	32	17	13	2	X			Y/2
checkEndOfMediaStream	28	17	1	2	X			
commandLoop	545	17	53	M_f	X	M_f	M_f	
checkCommandParams	238	17	269	T_b	X	T_b	T_b	T_b
singal_handler	13	17	0	2	X			
setupFBResolution	29	17	0	2	X	Y/3	Y/3	Y/3
setupFramebuffers	115	17	8	3	X	N/3	N/2	N/2
main_Thread	68	17	4	4	X		Y/3	Y/2

Equivalence Checking

				[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Test Program	L	B	P	Ti[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
threadRename	6	17	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
fileExists	19	17	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
readLine	27	17	11	3	X			
getCommand	**269**	**17**	**61**	**3**	**X**	**N/3**		**N/3**
powerDown	9	17	0	2	X			
digitStart	**12**	**17**	**0**	**2**	**X**	**Y/2**		
difgitAdd	**34**	**17**	**2**	**2**	**X**	**Y/2**		
checkEndOfPvrStream	**32**	**17**	**13**	**2**	**X**			**Y/2**
checkEndOfMediaStream	28	17	1	2	X			
commandLoop	545	17	53	M_f	X	M_f	M_f	
checkCommandParams	238	17	269	T_b	X	T_b	T_b	T_b
singal_handler	13	17	0	2	X			
setupFBResolution	**29**	**17**	**0**	**2**	**X**	**Y/3**	**Y/3**	**Y/3**
setupFramebuffers	**115**	**17**	**8**	**3**	**X**	**N/3**	**N/2**	**N/2**
main_Thread	**68**	**17**	**4**	**4**	**X**		**Y/3**	**Y/2**

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

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

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

UNIVERSITY OF Southampton School of Electronics and Computer Science

- **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) % buffer_size

fault: next %= buffer_size
next+=1

Verification of the LTL Properties

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

Verification of the LTL Properties

reactive system: ESBMC can check the LTL properties up to a certain unwinding bound

		T	B	C	Time	#FI/#I
		3	2	-	7	0/120
			3	-	80	0/1001
			4	-	107	0/8568
keyboard†	49	3	2	-	1	2/6
			3	-	1	3/8
			4	-	1	4/10
menu_app	847	3	2	-	16	0/3003
			3	20	271	0/50456
			4	20	625	0/87386
menu_app†	847	3	2	-	9	663/3003
			3	20	121	7584/50456
			4	20	218	12548/87386
log	135	3	2	-	12	0/12
			3	-	820	0/22
			4	10	1149	0/8
log†	135	3	2	-	1	12/16
			3	-	3	27/31
			4	-	5	48/52

Verification of the LTL Properties

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

		T	B	C	Time	#FI/#I
		3	2	-	7	0/120
			3	-	80	0/1001
			4	-	107	0/8568
			2	-	1	2/6
		3	3	-	1	3/8
			4	-	1	4/10
			2	-	16	0/3003
menu_app	847	3	3	20	271	0/50456
			4	20	625	0/87386
			2	-	9	663/3003
menu_app†	847	3	3	20	121	7584/50456
			4	20	218	12548/87386
			2	-	12	0/12
log	135	3	3	-	820	0/22
			4	10	1149	0/8
			2	-	1	12/16
log†	135	3	3	-	3	27/31
			4	-	5	48/52

Verification of the LTL Properties

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

Test Pr					ime	#FI/#I
keyboa					7	0/120
					80	0/1001
					107	0/8568
					1	**2/6**
keyboard†	49	3	3	-	1	**3/8**
			4	-	1	**4/10**
			2	-	16	0/3003
menu_app	847	3	3	20	271	0/50456
			4	20	625	0/87386
			2	-	9	**663/3003**
menu_app†	847	3	3	20	121	**7584/50456**
			4	20	218	**12548/87386**
			2	-	12	0/12
log	135	3	3	-	820	0/22
			4	10	1149	0/8
			2	-	1	**12/16**
log†	135	3	3	-	3	**27/31**
			4	-	5	**48/52**

Agenda

University of Southampton
School of Electronics and Computer Science

- 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

University of Southampton
School of Electronics and Computer Science

- 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)

    $\Rightarrow$ *combines symbolic with explicit state space exploration*
  - *UW:* memory overhead and slowdowns to extract the *unsat core*

    $\Rightarrow$ *it has not been used for BMC of multi-threaded software*

    $\Rightarrow$ *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