DSSE Dependable Systems & Software Engineering

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Continuous Verification of Large Embedded Software using SMT-Based Bounded Model Checking

Lucas Cordeiro, Bernd Fischer, Joao Marques-Silva
lcc08r@ecs.soton.ac.uk

Agenda

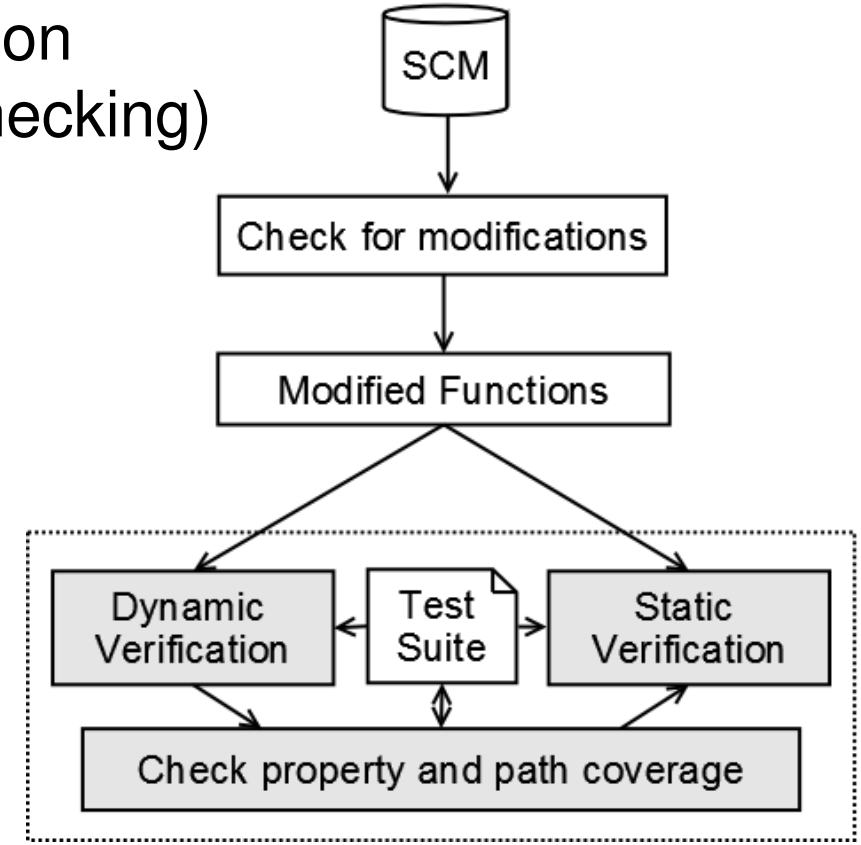
UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Continuous Verification
- SMT-based Bounded Model Checking (BMC)
- Case Study and Experimental Results
- Conclusions and Future Work

Continuous Verification

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

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

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

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

```
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

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

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

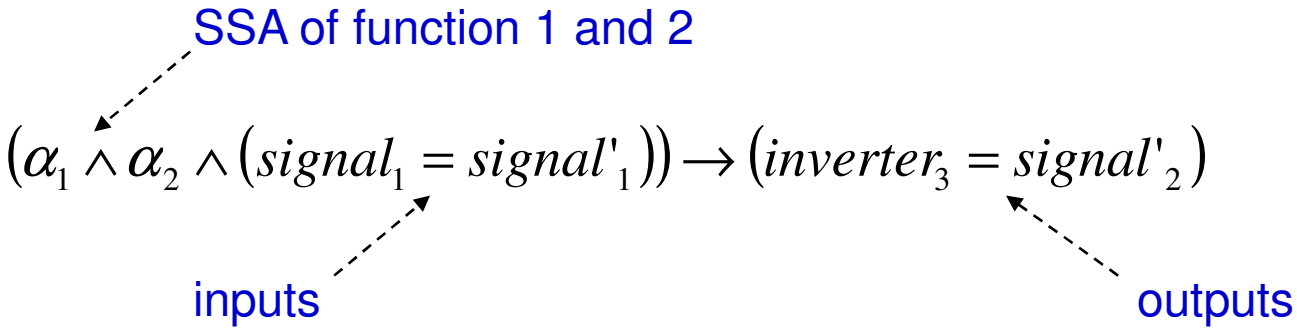
```
unsigned Inv(int signal) {
  unsigned inverter;
  if (signal >= 0)
    inverter = signal;
  else
    inverter = -1*signal;
  return inverter;
}
```

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

Functional Equivalence Checking

UNIVERSITY OF Southampton
School of Electronics and Computer Science

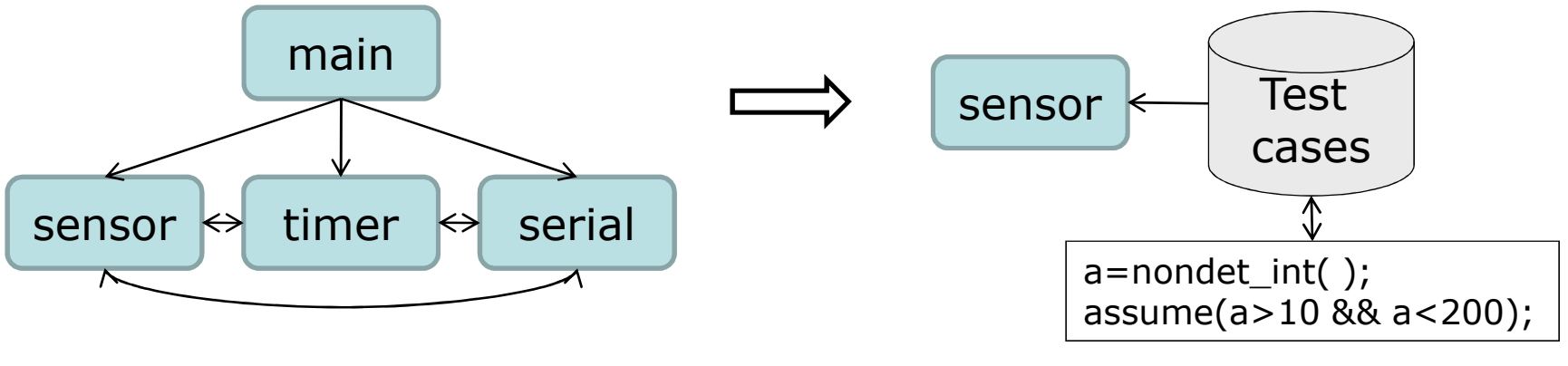
- determine whether modified functions need to be re-verified
  - no need to re-verify properties if functions are equivalent
  - **less expensive** than re-verifying the function
  - **undecidable** due to unbounded memory usage
- goal: compare input-output relation
  - remove variables and returns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



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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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 buffer over- and underflow

Agenda

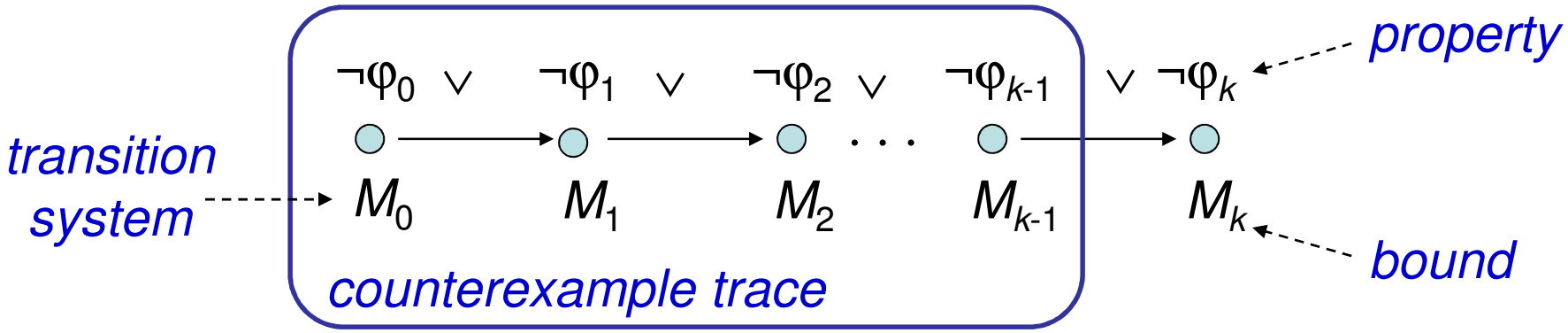
UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Continuous Verification
- SMT-based Bounded Model Checking (BMC)
- Case Study and Experimental Results
- Conclusions and Future Work

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 (embedded) software

Satisfiability Modulo Theories (1)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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

Satisfiability Modulo Theories (2)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

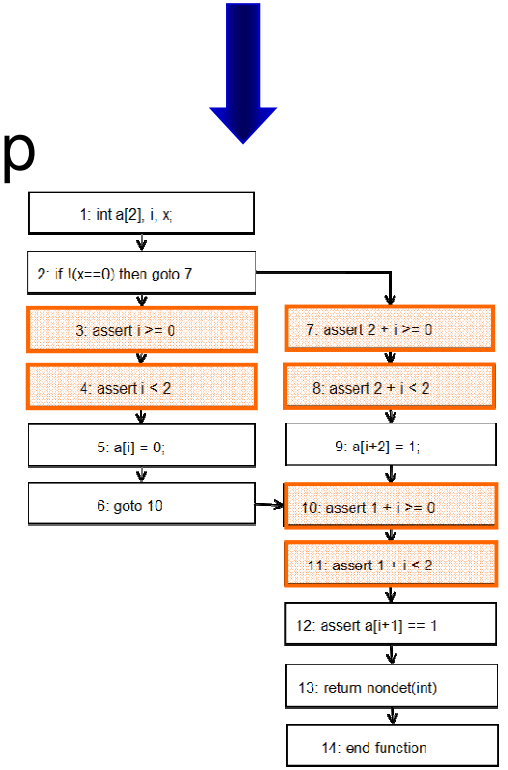
Software BMC using ESBMC

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 unrolled up to given bounds
  - number of loop iterations
  - size of arrays
- unrolled program optimized to reduce blow-up
  - constant folding
  - 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

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 unrolled up to given bounds
  - number of loop iterations
  - size of arrays
- unrolled program optimized to reduce blow-up
  - constant folding
  - 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

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 unrolled up to given bounds
  - number of loop iterations
  - size of arrays
- unrolled program optimized to reduce blow-up
  - constant folding
  - 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);
}
```

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

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

Extending ESBMC

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- SMT solvers provide different encodings for numbers:
  - abstract domains ($\mathbf{Z}$, $\mathbf{R}$)
  - fixed-width bit vectors (`unsigned int`, ...)
- majority of VCs solved faster if numeric types are modelled by abstract domains but possible loss of precision
  - default solver: Z3 (using AUFLIRA logic)
  - switch to Boolector and encode as bit-vectors (when using bit operations or typecasts but no pointers)
- encoding of floating-point arithmetic leads to large formulae
  - approximate by fixed-point arithmetic
- we check two properties for dynamic memory allocation
  - whether argument to malloc / free is a dynamic object
  - whether argument to free is still a valid object

Agenda

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Continuous Verification
- SMT-based Bounded Model Checking (BMC)
- Case Study and Experimental Results
- Conclusions and Future Work

Experimental Evaluation

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- goal: check efficiency and effectiveness of ESBMC
  - check error-detection capability on different benchmarks
  - evaluate ESBMC's performance relative to CBMC
  - evaluate scalability of the CV approach

- set-up:
  - Intel Pentium Dual CPU, 2GHz / 4GB RAM, Linux OS
  - time limit 3600 seconds / individual property

Error-Detection Capability

UNIVERSITY OF Southampton
School of Electronics and Computer Science

encoding time

SMT solver time

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

Module	#L	#P	Time			#P		
			Enc.	Solver	Time	Passed	Fail	Error
VERISEC	090	214	190.9	228.4	419.2	1949	202	0
NE					.2	188	20	0
SNU-RT	3102				79	762	28	0
WCET	3430	726	72.7	8.3			4	0
POWERSTONE	2957	2053	127.4	913.6	1(		0	0

lines of code

number of properties checked

*error detected in module – **GOOD THING***

Error-Detection Capability

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#L	#P	Time			#P		
			Enc.	Solver	Time	Passed	Fail	Error
VERISEC	**9090**	**2148**	**190.9**	**228.4**	**419.2**	**1949**	**202**	**0**
NECLA	**1011**	**208**	**59.3**	**89**	**148.2**	**188**	**20**	**0**
SNU-RT	2102	790	3167	12.2	3179	762	28	0
WCET				8.3	81	722	4	0
POWERST				913.6	1041	2043	10	0

VERISEC and NECLA are not specialized to embedded software

Error-Detection Capability

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#L	#P	Time			#P		
			Enc.	Solver	Time	Passed	Fail	Error
VERISEC	9090	2148	190.9	228.4	419.2	1949	202	0
NECLA	1011	208	59.3	89	148.2	188	20	0
SNU-RT	**3102**	**790**	**3167**	**12.2**	**3179**	**762**	**28**	**0**
WCET	**3430**	**726**	**72.7**	**8.3**	**81**	**722**	**4**	**0**
POWERSTONE	**2[illegible]**	**2053**	**127.4**	**913.6**	**1041**	**2043**	**10**	**0**

Contain ANSI-C constructs commonly found in embedded software

Error-Detection Capability

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#L	#P	Time			#P		
			Enc.	Solver	Time	Passed	Fail	Error
VERISEC	9090	2148	190.9	228.4	419.2	1949	**202**	0
NECLA	1011	208	5			88	**20**	0
SNU-RT	3102	790	31			62	**28**	0
WCET	3430	726	7			22	**4**	0
POWERSTONE	2957	2053	12			43	**10**	0

string manipulation, aliasing, dynamic memory allocation, interprocedural dataflow

Comparison to CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- SAT-based BMC for full ANSI-C
  - SMT-based version does not seem to support full ANSI-C
  - mature tool (V3.3.2)
  - ***not*** recent SMT-based version
- goal: compare efficiency of CBMC vs. ESBMC
  - on identical verification problems

Comparison to SAT-CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module			SAT-CBMC			ESBMC		
			Time	#P		Time	#P	
				Fail	Error		Fail	Error
exStbKey	558	22	3.1	0	0	1.1	0	0
exStbHDMI	1508	41	SF	0	41	211.1	0	0
exStbLE		59	MO	0	59	1817.6	0	0
exStbHwAcc	1432	115	0.2	0	0	1	0	0
exStbResolution	353	32	TO	0	32	1596.6	0	0
exStbFb	689	48	MO	0	48	138.1	0	0
exStbCc		5	174.5	0	0	46.1	0	0
exStbDem		267	MO	0	267	MO	0	267

Segmentation fault

Memory out

Time out

Comparison to SAT-CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#L	#P	SAT-CBMC			ESBMC		
			Time	#P		Time	#P	
				Fail	Error		Fail	Error
exStbKey	558	22	3.1	0	0	1.1	0	0
exStbHDMI	1508	41	SF	0	41	211.1	0	0
exStbLED				0	59	1817.6	0	0
exStbHwAcc				0	0	1	0	0
exStbResolution				0	32	1596.6	0	0
exStbFb				0	48	138.1	0	0
exStbCc	331	5	174.5	0	0	46.1	0	0
exStbDemo	30902	267	MO	0	267	MO	0	267

Both tools fail to model check the module exStbDemo

Comparison to SMT-CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#L	#P	SMT-CBMC			ESBMC		
			Time	#P		Time	#P	
				Fail	Error		Fail	Error
exStbKey	**558**	**22**	**3.1**	**0**	**0**	**1.1**	**0**	**0**
exStbHDMI	1508	41	SF	0	41	211.1	0	0
exStbLED	430	59	TO	0	59	1817.6	0	0
exStbHwAcc	**1432**	**115**	**0.7**	**0**	**0**	**1**	**0**	**0**
exStbResolution	353	32	TO	0	32	1596.6	0	0
exStbFb	689	48	SF	0	48	138.1	0	0
exStbCc	331	5	TO	0	5	46.1	0	0
exStbDemo	30902	267	MO	0	267	MO	0	267

Scalability

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- To model check the exStbDemo module, we apply the continuous verification approach
  - we use EmbUnit test framework for dynamic verification
  - we use subversion as SCM system
- goal: apply the CV approach to large embedded software used in a commercial product

Scalability

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#TC	#P	SMT-CBMC			Subversion			
			Time	#P		PR10	PR11	PR12	PR13
				Fail	Error				
getCommand	18	237	4.4	1	0	X	X	X	
commandLoop	26	237	128.4	0	0	X			X
c	4	229	161.2	0	0	X	X	X	X
c	3	228	4.4	0	0	X			X
checkEndOfIPStream	3	228	4	0	0	X			
checkEndOfMediaStream	4	228	4	0	0	X			
setupFrameBuffers	2	228	4.2	0	0	X	X		X
setupFBResolution	2	228	4	0	0	X		X	
Total verification time in seconds for each PR						314.4	169.8	169.5	298.1

Number of test cases

Scalability

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#TC	#P	SMT-CBMC			Subversion			
			Time	#P					
				Fail	E			R12	PR13
getCommand	**18**	**237**	**4.4**	**1**	**0**	**X**	**X**	**X**	
commandLoop	26	237	128.4	0	0	X			X
checkCommandParams	4	229	161.2	0	0	X	X	X	X
checkEndOfPvrStream	3	228	4.4	0	0	X			X
checkEndOfIPStream	3	228	4	0	0	X			
checkEndOfMediaStream	4	228	4	0	0	X			
setupFrameBuffers	2	228	4.2	0	0	X	X		X
setupFBResolution	2	228	4	0	0	X		X	
Total verification time in seconds for each PR						314.4	169.8	169.5	298.1

Invalid pointer

Scalability

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Module	#TC	#P	SMT-CBMC			Subversion			
			Time	#P		PR10	PR11	PR12	PR13
				Fail	Error				
getCommand	18	237	4.4	1	0	X	X	X	
commandLoop	26	237	128.4	0	0	X			X
checkCommandParams	4	229	161.2	0	0	X	X	X	X
checkEndOfPvrStream	3	228	4.4	0	0	X			X
checkEndOfIPStream	3	228	4	0					
checkEndOfMediaStream	4	228	4	0					
setupFrameBuffers	2	228	4.2	0			X		X
setupFBResolution	2	228	4	0	0	X		X	
Total verification time in seconds for each PR						**314.4**	**169.8**	**169.5**	**298.1**

reduces verification time by up to 50%

But not always

Agenda

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Continuous Verification
- SMT-based Bounded Model Checking
- Case Study and Experimental Results
- Conclusions and Future Work

Conclusions

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- introduced **continuous verification** approach
- evaluated on large embedded software
- described a new set of encodings that allow us to reason accurately about embedded software.
  - provided encodings for typical ANSI-C constructs not directly supported by SMT-solvers
- available at `users.ecs.soton.ac.uk/lcc08r/esbmc/`

Future work

- concurrency (based on Pthread library)
- termination analysis