UNIVERSITY OF Southampton
School of Electronics and Computer Science

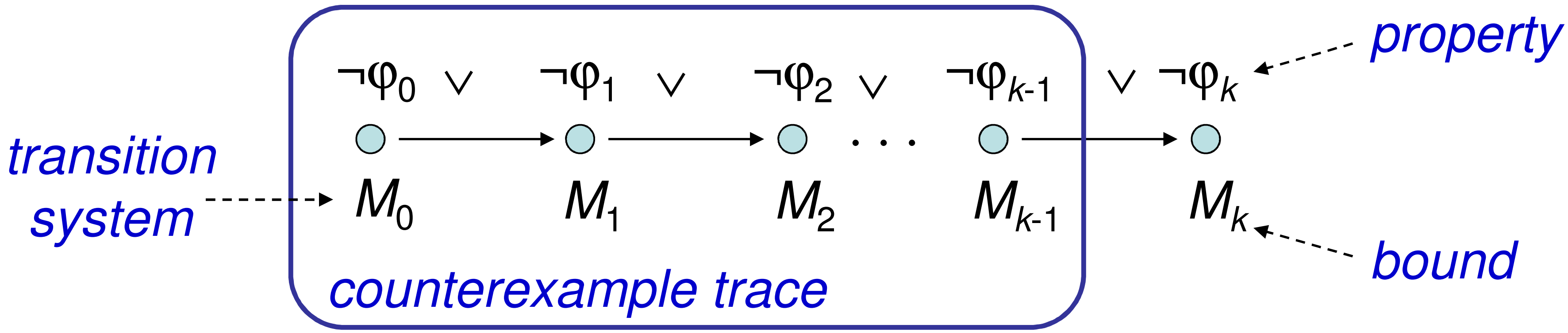
SMT-Based Bounded Model Checking for Embedded ANSI-C Software

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

Bounded Model Checking (BMC)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



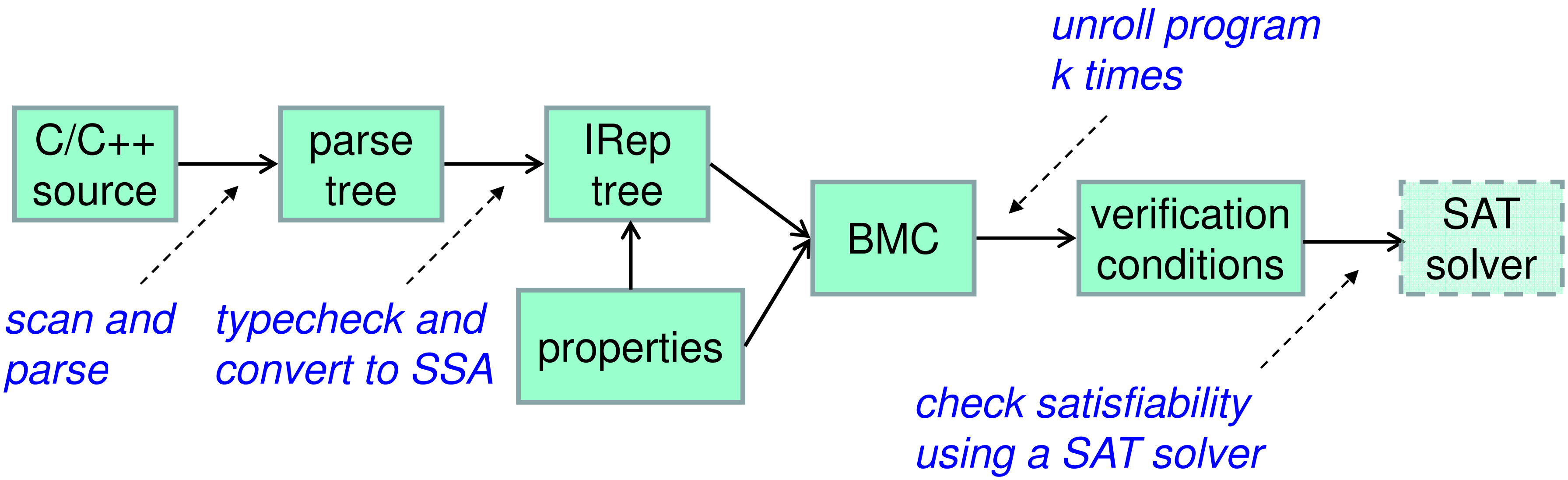
- transition system *M* unrolled *k* times
  - for programs: unroll loops, unfold arrays, …
- translated into verification condition ψ such that

  ψ satisfiable iff φ has counterexample of max. depth k

- has been applied successfully to verify (embedded) software

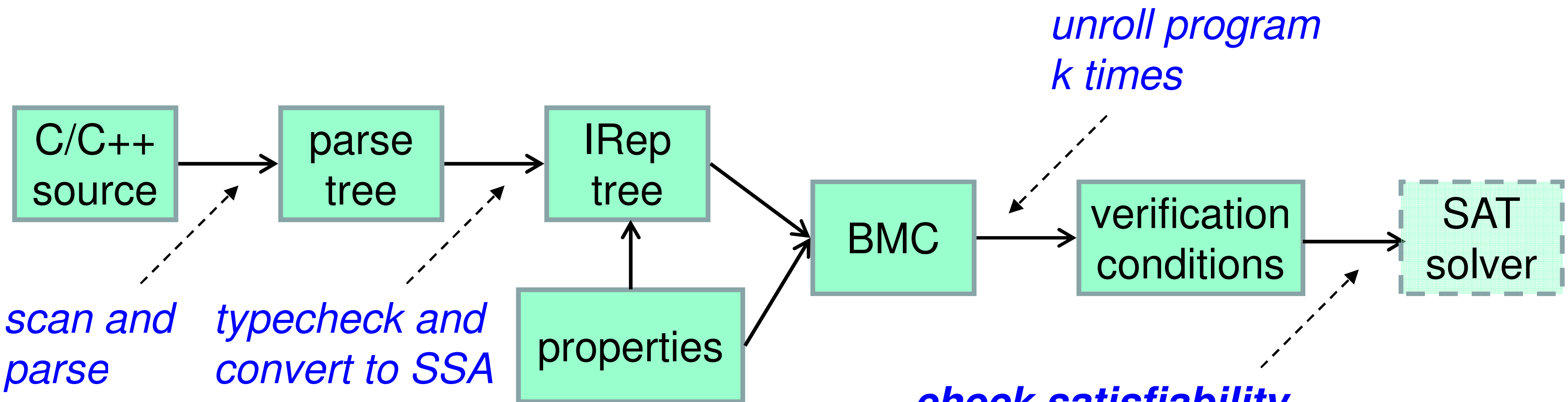
SAT-based CBMC [D. Kroening]

implements BMC for ANSI-C/C++ programs using SAT-solvers:



SAT-based CBMC [D. Kroening]

implements BMC for ANSI-C/C++ programs using SAT-solvers:



check satisfiability
using a SAT solver
⇒ conversion to propositional form

Problems (due to ***bit-blasting***):

- **complex expressions** lead to large propositional formulae
- **high-level information is lost**

Encoding of x == a + b

- *represent x, a, b by n independent propositional variables each*
- *represent addition by logical circuit*
- *represent equality by equivalences on propositional variables*

Objective of this work

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Exploit SMT to improve BMC of embedded software

- exploit background theories of SMT solvers
- provide suitable encodings for
  - pointers
  - unions
  - bit operations
  - arithmetic over- and underflow
- build an SMT-based BMC tool for full ANSI-C
  - build on top of CBMC front-end
  - use several third-party SMT solvers as back-ends
- evaluate ESBMC over embedded software applications

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	$(j \leq k \wedge a[j]=2) \Rightarrow a[i] < 3$

Satisfiability Modulo Theories (2)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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 ($\Gamma \models_T \varphi$) 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 $\Gamma \cup \{\neg\varphi\}$

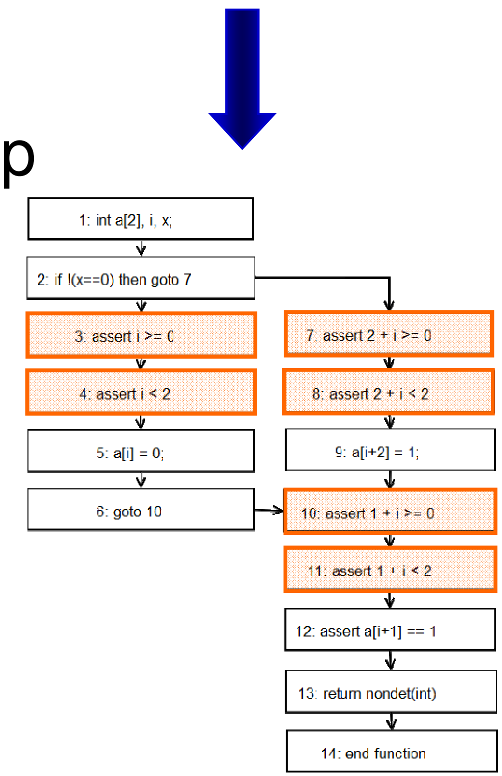
Software BMC using ESBMC

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);
}
```

$$
\begin{aligned}
g_1 &= x_1 == 0 \\
a_1 &= a_0 \text{ WITH } [i_0 := 0] \\
a_2 &= a_0 \\
a_3 &= a_2 \text{ WITH } [2+i_0 := 1] \\
a_4 &= g_1 \ ?\ a_1 : a_3 \\
t_1 &= a_4[1+i_0] == 1
\end{aligned}
$$

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 } crucial
  - forward substitutions
- 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

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

valid in abstract domains such as $\mathbb{Z}$ or $\mathbb{R}$

doesn't hold for bitvectors, due to possible overflows

  - majority of VCs solved faster if numeric types are modelled by abstract domains but possible loss of precision
  - ESBMC supports both encodings

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: extractBits, …)
    - ▹ different conversions for every pair of types
    - ▹ uses type information provided by front-end
  - conversion to / from bool via if-then-else operator
- arithmetic over- / underflow
  - standard requires modulo-arithmetic for unsigned integers
  - define error literals to detect over- / underflow for other types

$$res\_ok \Leftrightarrow \neg\, overflow(x, y) \wedge \neg\, underflow(x, y)$$

    - ▹ similar to conversions
- floating-point numbers
  - approximated by fixed-point numbers, integral part only
  - represented by fixed-width bitvector

Encoding of Structured Datatypes

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

$$
\begin{pmatrix}
p_1 := \mathrm{store}(p_0,\ 0,\ \&a[0]) \\
\wedge\ p_2 := \mathrm{store}(p_1,\ 1,\ 0) \\
\wedge\ g_2 := (x_2 == 0) \\
\wedge\ a_1 := \mathrm{store}(a_0,\ i_0 \ldots \\
\ldots (a_2,\ 1+ i_0,\ 1) \\
\wedge\ a_4 := \mathrm{ite}(\ldots_1,\ a_1,\ a_3) \\
\wedge\ p_3 := \mathrm{store}(p_2,\ 1,\ \mathrm{select}(p_2\ ,\ 1)+2)
\end{pmatrix}
$$

Store object at position 0

Store index at position 1

Update index

Encoding of Structured Datatypes

- 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);
}
```

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

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

Evaluation

Comparison of SMT solvers

- Goal: compare efficiency of different SMT-solvers
  - CVC3 (1.5)
  - Boolector (1.0)
  - Z3 (2.0)
- 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		Boolector		Z3	
				Time	Error	Time	Error	Time	Error
Bu[illegible]	[illegible]	43	17	[illegible]	[illegible]	[illegible]	0	2	0
	[illegible]	43	17	[illegible]	[illegible]	[illegible]	0	163.2	0
SelectionSort	(n=35)	34	17	8.5	0	0.8	0	0.8	0
	(n=140)	34	17	MO	1	74.6	0	74.4	0
InsertionSort	(n=35)	86	17	35.6	0	2.4	0	2.5	0
	(n=140)	86	17	MO	1	TO	1	143	0
Prim		[illegible]	[illegible]	[illegible]	0	0.5	0	0.5	0
StrCmp		[illegible]	[illegible]	[illegible]	0	91.2	0	38.8	0
MinMax		19	9	MO	1	947.6	0	6.2	0
lms		258	23	1011.9	0	138.7	0	138.6	0
Bitwise		18	1	272.4	0	7.5	0	28.4	0
adpcm_encode		149	12	211.8	0	738.9	0	5.5	0
adpcm_decode		111	10	43.8	0	20.2	0	14.3	0

lines of code

number of properties checked

size of arrays

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	28.3	0	1.9	0	2	0
	(n=140)	43	17	MO	1	182.7	0	163.2	0
SelectionSort	(n=35)	34	17	8.5	0	0.8	0	0.8	0
	(n=140)	[illegible]	[illegible]	[illegible]	[illegible]	74.6	0	74.4	0
InsertionSort	(n=35)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]2.4	0	2.5	0
	(n=140	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]TO	1	143	0
Prim		[illegible]	[illegible]	[illegible]	[illegible]	0.5	0	0.5	0
StrCmp		[illegible]	6	9.9	0	91.2	0	38.8	0
MinMax		19	9	MO	1	947.6	0	6.2	0
lms		**258**	**23**	**1011.9**	**0**	**138.7**	**0**	**138.6**	**0**
Bitwise		**18**	**1**	**272.4**	**0**	**7.5**	**0**	**28.4**	**0**
adpcm_encode		**149**	**12**	**211.8**	**0**	**738.9**	**0**	**5.5**	**0**
adpcm_decode		**111**	**10**	**43.8**	**0**	**20.2**	**0**	**14.3**	**0**

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

Comparison of SMT solvers

Module		#L	#P	CVC3 Time	CVC3 Error	Boolector Time	Boolector Error	Z3 Time	Z3 Error
BubbleSort	(n=35)	43	17	28.3	0	[illegible]	[illegible]	[illegible]	0
	(n=140)	43	17	MO	1	[illegible]	[illegible]	[illegible]	0
SelectionSort	(n=35)	34	17	8.5	0	[illegible]	[illegible]	[illegible]	0
	(n=140)	34	17	MO	1	74.6	0	74.4	0
InsertionSort	(n=35)	86	17	35.6	0	2.4	0	2.5	0
	(n=140)	86	17	MO	1	TO	1	143	0
Prim		79	30	16.9	0	0.5	0	0.5	0
StrCmp		14	6	9.9	0	91.2	0	38.8	0
MinMax		19	9	MO	1	947.6	0	6.2	0
lms		258	23	1011.9	0	138.7	0	138.6	0
Bitwise		18	1	272.4	0	7.5	0	28.4	0
adpcm_encode		149	12	211.8	0	738.9	0	5.5	0
adpcm_decode		111	10	43.8	0	20.2	0	14.3	0

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

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
BubbleSort [illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.9	0	2	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	82.7	0	163.2	0
SelectionSort [illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.8	0	0.8	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	74.6	0	74.4	0
InsertionSort (n=35)	86	17	35.6	0	2.4	0	2.5	0
(n=140)	86	17	MO	1	TO	1	143	0
Prim	79	30	16.9	0	0.5	0	0.5	0
StrCmp	14	6	9.9	0	91.2	0	38.8	0
MinMax	19	9	MO	1	947.6	0	6.2	0
lms	258	23	1011.9	0	138.7	0	138.6	0
Bitwise	18	1	272.4	0	7.5	0	28.4	0
adpcm_encode	149	12	211.8	0	738.9	0	5.5	0
adpcm_decode	111	10	43.8	0	20.2	0	14.3	0

Boolector and Z3 roughly comparable, with some advantages for Z3

Comparison of SMT solvers

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Goal: compare efficiency of different SMT-solvers
  - CVC3 (1.5)
  - Boolector (1.0)
  - Z3 (2.0)
- Set-up:
  - identical ESBMC front-end, individual back-ends
  - unsupported operations axiomatized
  - standard desktop PC, time-out 3600 seconds

⇒ SMT-solver of choice: Z3
  - best coverage of domain
  - overall fastest

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

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

Module	ESBMC		SMT-CBMC
	Z3	CVC3	CVC3
BubbleSort (n=35)	2.0	28.7	94.5
(n=140)	163.1	MO	*
SelectionSort (n=35)	0.8	8.5	66.5
(n=140)	74.4	MO	MO
BellmanFord	0.3	0.5	13.6
Prim	0.5	16.9	18.4
StrCmp	38.8	9.9	TO
SumArray	4.7	1.2	113.8
MinMax	6.2	MO	MO

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

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

All benchmarks taken from SMT-CBMC suite

Module			SMT-CBMC
	Z3	CVC3	CVC3
BubbleSort (n=35)	2.0	28.7	94.5
(n=140)	163.1	MO	*
SelectionSort (n=35)	0.8	8.5	66.5
(n=140)	74.4	MO	MO
BellmanFord	0.3	0.5	13.6
Prim	0.5	16.9	18.4
StrCmp	38.8	9.9	TO
SumArray	4.7	1.2	113.8
MinMax	6.2	MO	MO

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

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

Module	ESBMC		SMT-CBMC
	Z3	CVC3	CVC3
BubbleSort (n=35)	[illegible]	28.7	94.5
[illegible]	[illegible]	MO	*
[illegible]	[illegible]	8.5	66.5
[illegible]	[illegible]	MO	MO
BellmanFord	0.3	0.5	13.6
Prim	0.5	16.9	18.4
StrCmp	38.8	9.9	TO
SumArray	4.7	1.2	113.8
MinMax	6.2	MO	MO

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 (n=35)	2.0	28.7	94.5
(n=140)	163.1	MO	*
SelectionSort (n=35)	0.8	8.5	
(n=140)	74.4	MO	
BellmanFord	0.3	0.5	13.6
Prim	0.5	16.9	18.4
StrCmp	38.8	9.9	TO
SumArray	4.7	1.2	113.8
MinMax	6.2	MO	MO

Z3 not uniformly better than CVC3

Comparison to SAT-CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- SAT-based BMC for full ANSI-C
  - ***not*** recent SMT-based version
  - mature tool (V 2.9)
  - front-end and overall structure shared with ESBMC
- Goal: compare efficiency of SAT vs. SMT
  - on identical verification problems

Comparison to SAT-CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

			SAT-CBM[illegible]				[illegible]SBMC			
			Time		[illegible]		[illegible]		#P	
Module	#L	#P	Enc.	Solver	Fail	Error	Enc.	Solver	Fail	Error
fft1	218	72	0.4	<0.1	0	0	0.4	<0.1	0	0
fft1k	155	39	MO	-	0	39	[illegible]37.8	<0.1	0	0
jfdctint	374	331	1.2	[illegible]	1	0	[illegible]	2.4	1	0
lms	25[illegible]	[illegible]	[illegible]	[illegible]	0	35	[illegible]	[illegible]	[illegible]	0
ludcmp	14[illegible]	[illegible]	[illegible]	[illegible]	0	1	[illegible]	[illegible]	[illegible]	0
qurt	16[illegible]	[illegible]	[illegible]	[illegible]	0	1	[illegible]	[illegible]	[illegible]	0
pocsag	521	183	15.3	0.1	1	0	12.3	5.8	1	0
adpcm	473	553	74.3	3.5	0	0	45.7	9.2	0	0
laplace	110	76	30.8	TO	0	76	12.3	0.3	0	0
exStbKey	558	18	1.2	<0.1	0	0	1.2	<0.1	0	0
exStbHDMI	1045	25	167.9	78.9	0	0	164.4	33.5	0	0
exStbLED	430	6	195.9	130.0	0	0	165.6	44.5	0	0
exStbHwAcc	1432	113	0.7	<0.1	0	0	0.7	<0.1	0	0
exStbRes	353	40	271.8	319.0	0	0	269.3	1161.0	0	0

encoding time

SMT / SAT solver time

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

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

Comparison to SAT-CBMC [D. Kroening]

University of Southampton
School of Electronics and Computer Science

			SAT-CBMC				[illegible]SBMC			
			Time		#P		[illegible]		#P	
Module	#L	#P	Enc.	Solver	Fail	Error	Enc.	Solver	Fail	Error
fft1	218	72	0.4	<0.1	0	[illegible]	0.4	<0.1	0	0
fft1k	155	39	MO	[illegible]	0	39	[illegible]	<0.1	0	0
jfdctint	374	331	1.2	[illegible]	1	0	[illegible]	2.4	1	0
lms	[illegible]	[illegible]	[illegible]	[illegible]	0	35	[illegible]	[illegible]	[illegible]	0
ludcmp	[illegible]	[illegible]	[illegible]	[illegible]	0	1	[illegible]	[illegible]	[illegible]	0
qurt	[illegible]	[illegible]	[illegible]	[illegible]	0	1	[illegible]	[illegible]	[illegible]	0
pocsag	521	183	15.3	0.1	1	0	12.3	5.8	1	0
adpcm	[illegible]	553	74.3	3.5	0	0	45.7	9.2	0	0
laplace	[illegible]	[illegible]	30.8	TO	0	76	12.3	0.3	0	0
exStbKey	558	[illegible]	1.2	<0.1	0	0	1.2	<0.1	0	0
exStbHDMI	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	164.4	33.5	0	0
exStbLED	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	165.6	44.5	0	0
exStbHwAcc	1432	113	0.7	<0.1	0	0	0.7	<0.1	0	0
exStbRes	353	40	271.8	319.0	0	0	269.3	1161.0	0	0

encoding time

SMT / SAT solver time

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

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

all embedded systems applicatons

Comparison to SAT-CBMC [D. Kroening]

University of Southampton
School of Electronics and Computer Science

			SAT-CBMC				ESBMC			
			Time		#P		Time		#P	
Module	#L	#P	Enc.	Solver	Fail	Error	Enc.	Solver	Fail	Error
fft1	218	72	0.4	<0.1	0	0	0.4	<0.1	0	0
fft1k	155	39	MO	-	0	39	2337.8	<0.1	0	0
jfdctint	374	331	1.2	<0.1	1	0	0.5	2.4	1	0
lms	258	35	MO	-	0	35	132.6	0.2	0	0
ludcmp	144	88	4.5	TO	0	1	<0.1	1.44	0	0
qurt	164	8	18.8	TO	0	1	1.2	7.7	0	0
pocsag	521	183	15.3	0.1	1	0	12.3	5.8	1	0
adpcm	473	553	74.3	3.5	0	0	45.7	9.2	0	0
laplace	110	76	30.8	TO	0	76	12.3	0.3	0	0
exStbKey	558	18	1.2	<0.1	0	0	1.2	<0.1	0	0
exStbHDMI	1045	25	167.9	78.9	0	0	164.4	33.5	0	0
exStbLED	430	6	195.9	130.0	0	0	165.6	44.5	0	0
exStbHwAcc	1432	113	0.7	<0.1	0	0	0.7	<0.1	0	0
exStbRes	353	40	271.8	319.0	0	0	269.3	1161.0	0	0

Comparison to SAT-CBMC [D. Kroening]

SMT-encoding often more efficient than bit-blasting

			SAT-CBMC				ESBMC			
			Time		#P		Time		#P	
			Enc.	Solver	Fail	Error	**Enc.**	Solver	Fail	Error
[illegible]	[illegible]	[illegible]	0.4	<0.1	0	0	0.4	<0.1	0	0
fft1k	[illegible]	39	**MO**	-	0	39	**2337.8**	<0.1	0	0
jfdctint	37[illegible]	331	1.2	<0.1	1	0	0.5	2.4	1	0
lms	258	35	**MO**	-	0	35	**132.6**	0.2	0	0
ludcmp	144	88	**4.5**	TO	0	1	**<0.1**	1.44	0	0
qurt	164	8	**18.8**	TO	0	1	**1.2**	7.7	0	0
pocsag	521	183	15.3	0.1	1	0	12.3	5.8	1	0
adpcm	473	553	**74.3**	3.5	0	0	**45.7**	9.2	0	0
laplace	110	76	**30.8**	TO	0	76	**12.3**	0.3	0	0
exStbKey	558	18	1.2	<0.1	0	0	1.2	<0.1	0	0
exStbHDMI	1045	25	167.9	78.9	0	0	164.4	33.5	0	0
exStbLED	430	6	195.9	130.0	0	0	165.6	44.5	0	0
exStbHwAcc	1432	113	0.7	<0.1	0	0	0.7	<0.1	0	0
exStbRes	353	40	271.8	319.0	0	0	269.3	1161.0	0	0

Comparison to SAT-CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

			SAT-CBMC				ESBMC			
			Time		#P		Time		#P	
Module	#L	#P	Enc.	**Solver**	Fail	Error	Enc.	**Solver**	Fail	Error
fft1	218	72	0.4	<0.1	0	0	0.4	<0.1	0	0
fft[illegible]	[illegible]	[illegible]	MO	-	0	39	2337.8	<0.1	0	0
jf[illegible]	[illegible]	[illegible]	1.2	<0.1	1	0	0.5	2.4	1	0
lr[illegible]	[illegible]	[illegible]	MO	-	0	35	132.6	0.2	0	0
lu[illegible]	[illegible]	[illegible]	4.5	**TO**	0	1	<0.1	**1.44**	0	0
c[illegible]	[illegible]	[illegible]	18.8	**TO**	0	1	1.2	**7.7**	0	0
p[illegible]	[illegible]	[illegible]	[illegible]5.3	**0.1**	1	0	12.3	**5.8**	1	0
ad[illegible]	[illegible]	[illegible]	7[illegible].3	**3.5**	0	0	45.7	**9.2**	0	0
laplace	110	76	30.8	**TO**	0	76	12.3	**0.3**	0	0
exStbKey	558	18	1.2	<0.1	0	0	1.2	<0.1	0	0
exStbHDMI	1045	25	167.9	**78.9**	0	0	164.4	**33.5**	0	0
exStbLED	430	6	195.9	**130.0**	0	0	165.6	**44.5**	0	0
exStbHwAcc	1432	113	0.7	<0.1	0	0	0.7	<0.1	0	0
exStbRes	353	40	271.8	**319.0**	0	0	269.3	**1161.0**	0	0

SMT-solver often significantly faster than SAT-solver

Comparison to SAT-CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

			SAT-CBMC				ESBMC			
			Time		#P		Time		#P	
Module	#L	#P	Enc.	Solver	Fail	Error	Enc.	Solver	Fail	Error
fft1	218	72	0.4	<0.1	0	0	0.4	<0.1	0	0
fft[illegible]	[illegible]	[illegible]	MO	-	0	39	2337.8	<0.1	0	0
jf[illegible]	[illegible]	[illegible]	1.2	<0.1	1	0	0.5	2.4	1	0
lr[illegible]	[illegible]	[illegible]	MO	-	0	35	132.6	0.2	0	0
lu[illegible]	[illegible]	[illegible]	4.5	**TO**	0	1	<0.1	**1.44**	0	0
c[illegible]	[illegible]	[illegible]	18.8	**TO**	0	1	1.2	**7.7**	0	0
p[illegible]	[illegible]	[illegible]	15.3	**0.1**	1	0	12.3	**5.8**	1	0
ad[illegible]	[illegible]	[illegible]	74.3	**3.5**	0	0	45.7	**9.2**	0	0
laplace	11[illegible]	[illegible]6	30.8	**TO**	0	76	12.3	**0.3**	0	0
exStbKey	558	[illegible]	1.2	<0.1	0	0	1.2	<0.1	0	0
exStbHDMI	1045	2[illegible]	167.9	**78.9**	0	0	164.4	**33.5**	0	0
exStbLED	430	6	[illegible]5.9	**130.0**	0	0	165.6	**44.5**	0	0
exStbHwAcc	1432	113	0[illegible]7	<0.1	0	0	0.7	<0.1	0	0
exStbRes	353	40	271.8	**319.0**	0	0	269.3	**1161.0**	0	0

SMT-solver often significantly faster than SAT-solver, but not always

Conclusions

- SMT-based BMC is more efficient than SAT-based BMC
  – but not uniformly
- described and evaluated first SMT-based BMC for ANSI-C
  – provided encodings for typical ANSI-C constructs not directly supported by SMT-solvers
- available at `users.ecs.soton.ac.uk/lcc08r/esbmc/`

Future work:

- better handling of floating-point numbers
- concurrency (based on Pthread library)
- termination analysis