ECBS 2013

SMT-Bounded Model Checking of C++ Programs

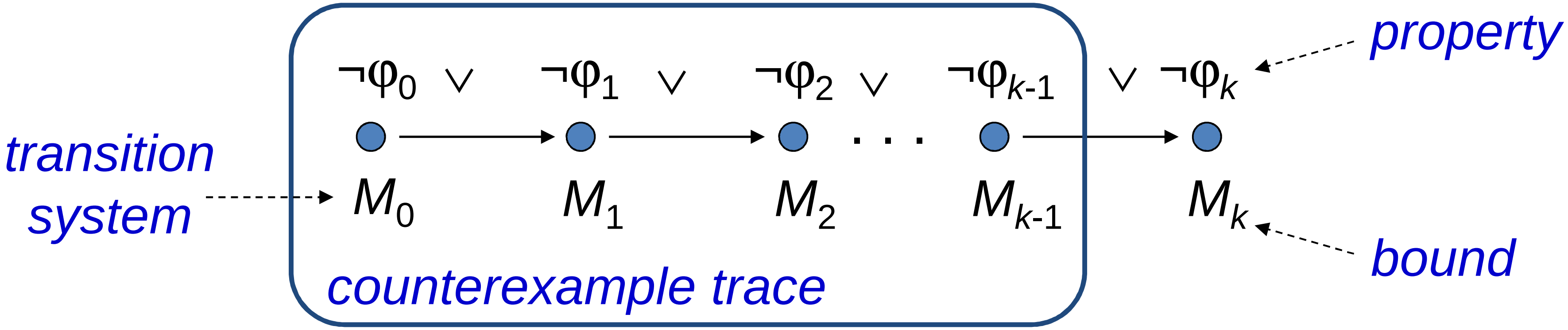
Mikhail Ramalho, **Mauro Freitas**, Felipe Sousa, Hendrio Marques, Lucas Cordeiro, Bernd Fischer

UFAM

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Bounded Model Checking (BMC)

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 C++ Programs

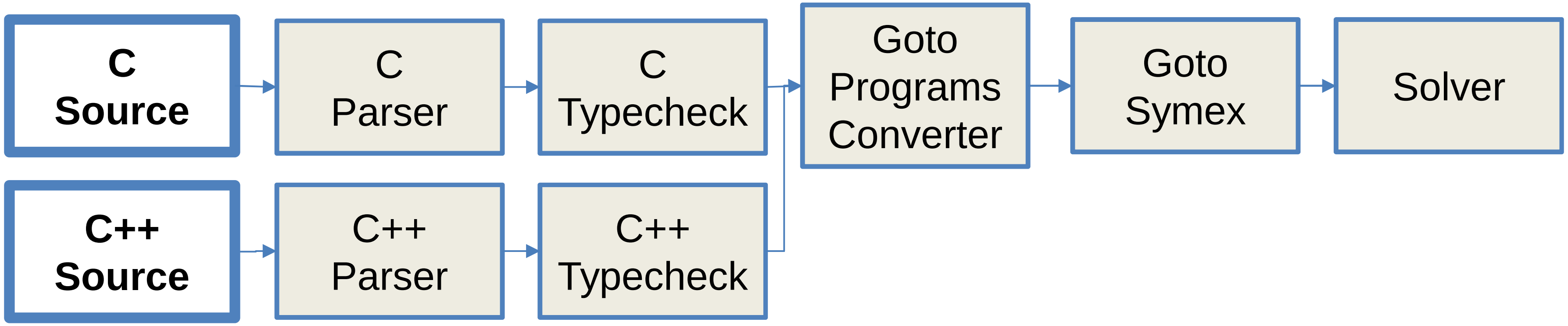
- there have been attempts to apply BMC to the verification of C++ programs but with **limited success**
  - handle **large programs** and **support complex features**
- problem: BMC of C++ programs presents greater challenges than that of C programs
  - more complex features such as **templates**, **containers**, and **exception handling** (contains and handles error situations in embedded systems)
- main insights:
  - optimized implementation of the standard C++ library complicates the VCs unnecessarily
  - abstract representation of the standard C++ libraries to conservatively approximate their semantics

Objetive of this work

Extend BMC to support complex features of C++

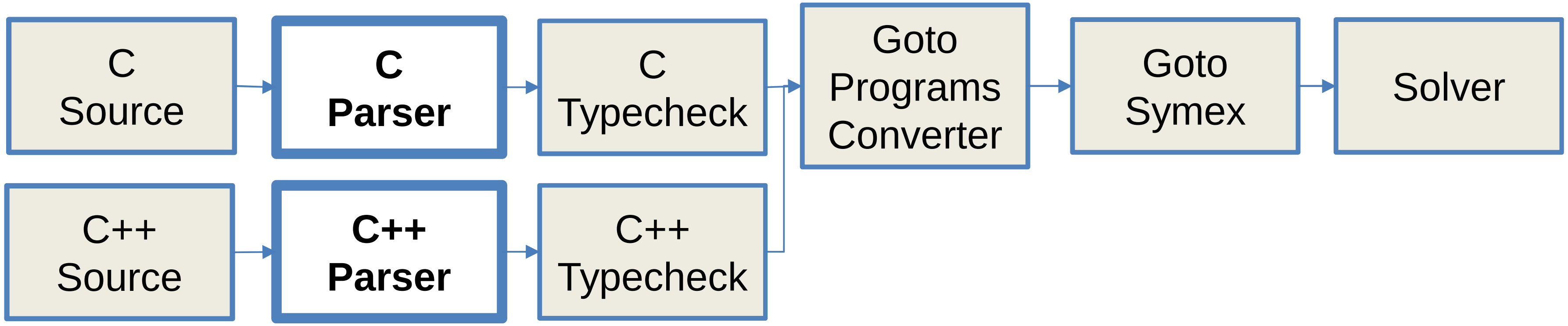
- exploit background theories of Satisfability Modulo Theories (SMT) solvers
- provide suitable encodings for
  - template
  - containers
  - exception handling
  - arithmetic over- and underflow
- build and evaluate an SMT-based BMC tool (ESBMC++)
  - build on top of CBMC front-end
  - use different SMT encodings as back-ends

ESBMC Architecture (1)



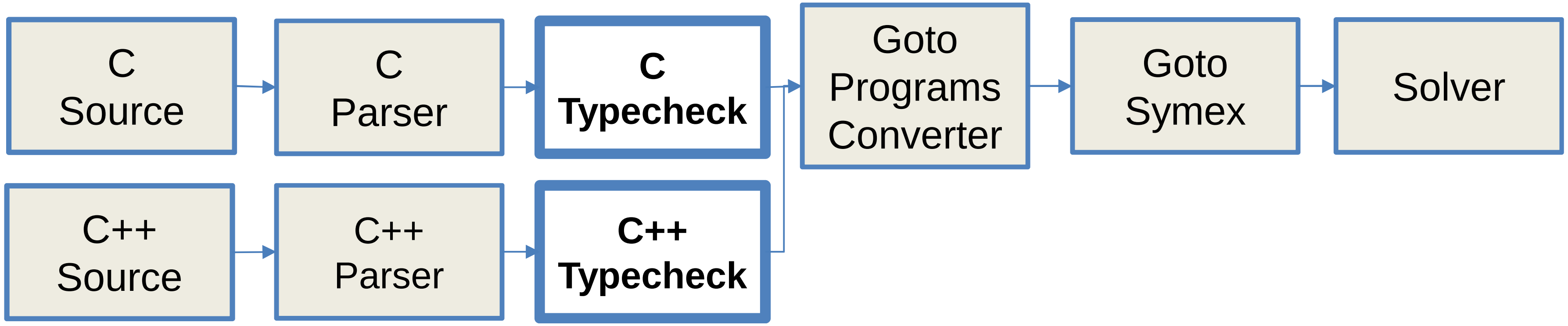
- originally only ANSI-C language was supported
- extend to support the verification of C++ programs with:
  - template (creation and instantiation)
  - exception handling (converted to goto functions)
  - standart template library (operational model)

ESBMC Architecture (2)



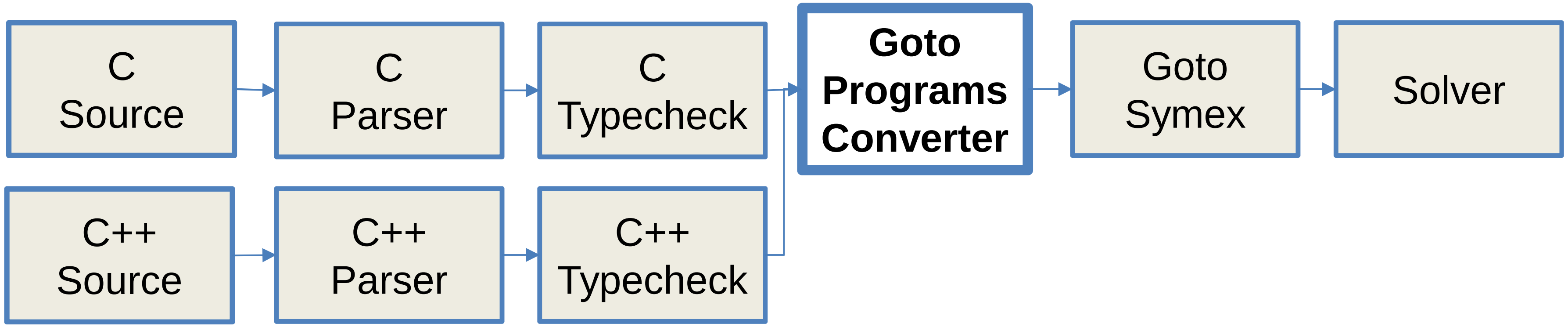
- lexer/parser based on the flex/bison
- most of the intermediate representation of the program (IRep) is created
  - this IRep is the base for the remaining phases of the verification

ESBMC Architecture (3)



- some checks are made in this step:
  - assignment check
  - typecast check
  - pointer initialization check
  - function call check
  - template instantiation

ESBMC Architecture (4)

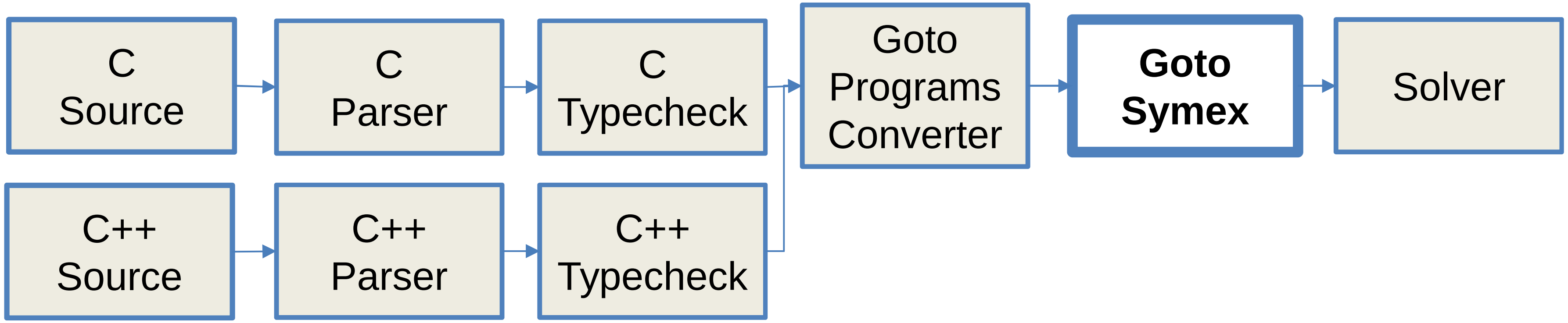


- conversion from **IRep** to **goto programs**:

```c
int main()
{
   int x=5;

   if(x==5)
      return 0;

   return -1;
}
```

```
main() (c::main):
  int x;
  x = 5;

  IF !(x == 5) THEN GOTO 1
    return 0;
1: return -1;

END_FUNCTION
```

ESBMC Architecture (5)



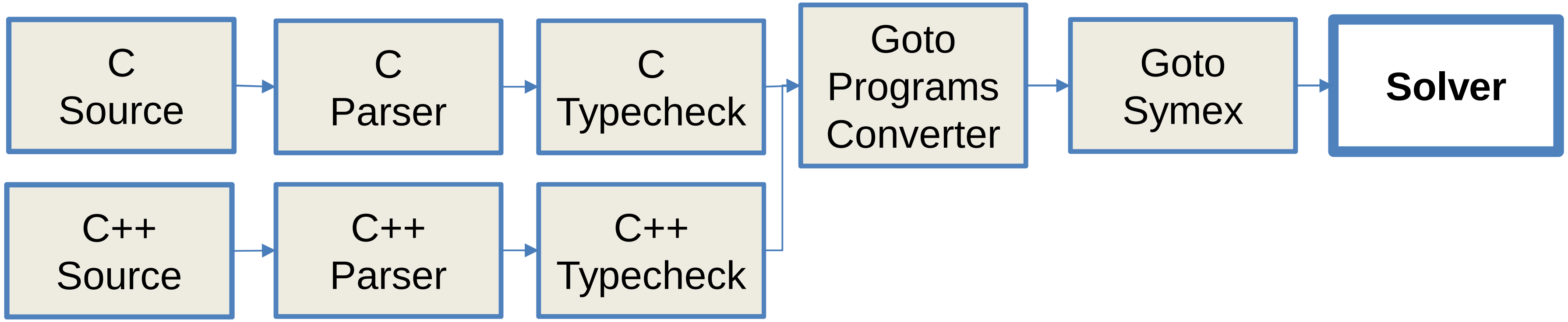
- creation of SSA expressions from goto programs:
  - assertions are inserted to check for pointer safety, memory-leak, division by zero, etc
  - **jump** instructions are inserted for **exception handling**

x = 5;
x = 6;
y = x;

⇓

$x_1 = 5;$
$x_2 = 6;$
$y_1 = x_2;$

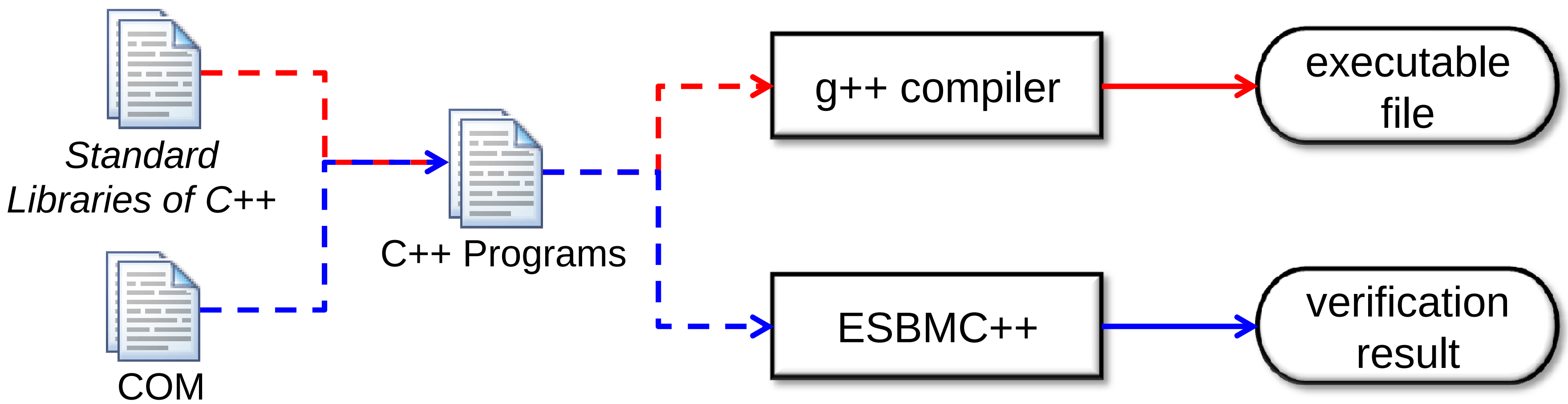
ESBMC Architecture (6)



- encoding to bit-vector or integer/real arithmetic
- verification results can depend on encodings:
  - majority of VCs solved faster if numeric types are modelled by abstract domains but possible loss of precision

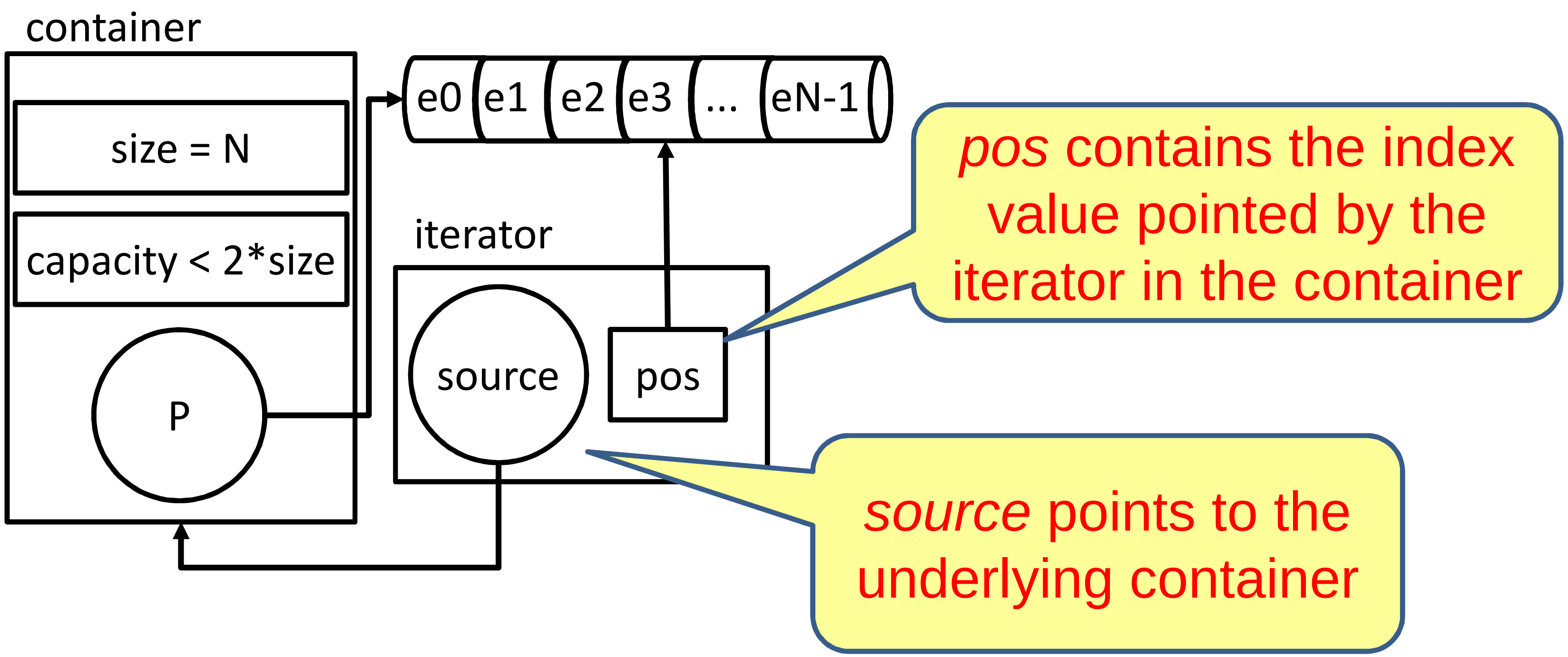
SMT-Based BMC of C++ Programs

- there have been attempts to apply BMC to the verification of C++ programs but with **limited success**
  - handle **large programs** and **support complex features**
- standard C++ libraries contain complex (and low-level) data structures (complicates the VCs unnecessarily)
  - provide a C++ operational model (COM) which is an **abstract representation** of the standard C++ libraries that **conservatively approximates** their semantics



Container Model (1)

- the **container model** uses three variables:
  - **P** that points to the first element of the array
  - **size** that stores the quantity of elements in the container
  - **capacity** that stores the total capacity of a container
- **iterators** are modelled using two variables (*source* and *pos*)



Container Model (2)

- the core container model only supports the **insert**, **erase**, and **search** methods
  - push_back, pop_back, front, back, push_front ,and pop_front are variation of these basic methods

$$
\begin{aligned}
& C((c',i') = c.erase(i)) := \\
& \wedge c'.size = c.size - 1 \\
& \wedge c'.array = store(\ldots(store(c.array, \\
& \qquad\qquad i.pos, select(c.array, i.pos+1)), \\
& \qquad\qquad \ldots, \\
& \qquad\qquad c.size-2, select(c.array, c.size-1)) \\
& \wedge i'.source = c' \\
& \wedge i'.pos = i.pos
\end{aligned}
$$

decrement the size of the container

the exclusion is made by a given position, regardless the value

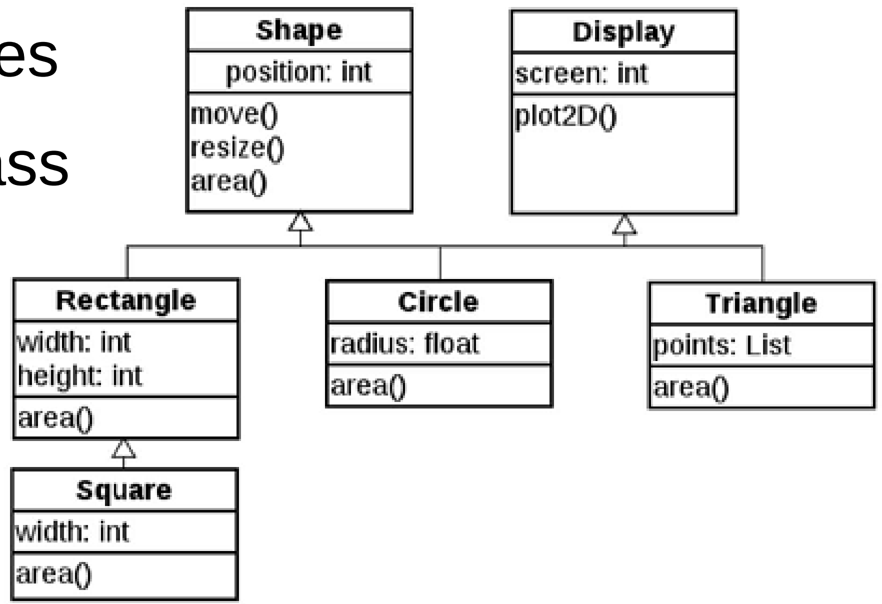
points to the position next to the previously erased part of the container

Inheritance and Polymorphism

- polymorphism allows the creation of reusable code by changing only specific methods from the base class
  - in constrast to Java, C++ allows **multiple inheritance** which increase the complexity of the static analysis
- in ESBMC++, each new class instantiation replicate all the methods and attributes from the base classes
  - this feature allows base classes pointers to keep reference to derived classes
  - during verification time decides which method is being called from such pointer

Running Example (1)

- triple <C, $\prec_s$, $\prec_r$ > where C is the set of classes
  - shared inheritance $\prec_s \subseteq$ C x C
  - replicated inheritance $\prec_r \subseteq$ C x C
- square class relation: <*C, ∅, {(Square, Rectangle, Shape), (Square, Rectangle, Display)}*>
  - direct access to the attributes and methods of the derived class
  - replicate information to any new class



Running Example (2)

```
Square (int w) : Rectangle(w,w)
{ width = w; }

int area(void) { return width*width; }
```

```
Square(10);
== 100);
```

Square constructor and area method

$$C := \begin{bmatrix} j_1 = store(j_0, vtable, \text{Rectanle}) \\ \wedge\ j_2 = store(j_1, width, 10) \\ \wedge\ j_3 = store(j_2, height, 10) \\ \wedge\ j_4 = store(j_3, vtable, \text{Square}) \\ \wedge\ j_5 = store(j_4, width, 10) \\ \wedge\ return\_value_1 = \\ (select(j_5, width) \times select(j_5, width) \end{bmatrix}$$

$$P := [return\_value_1 = 100]$$

Running Example (2)

```
Square (int w) :
{ width = w; }

int area(void) { return width*width; }
```

Instantiation of square and area call

```
Shape *sqre = new Square(10);
assert (sqre->area() == 100);
```

$$C := \left[\begin{array}{l} j_1 = store(j_0, vtable, \mathrm{Rectanle}) \\ \wedge\ j_2 = store(j_1, width, 10) \\ \wedge\ j_3 = store(j_2, height, 10) \\ \wedge\ j_4 = store(j_3, vtable, \mathrm{Square}) \\ \wedge\ j_5 = store(j_4, width, 10) \\ \wedge\ return\_value_1 = \\ (select(j_5, width) \times select(j_5, width) \end{array}\right]$$

$$P := [return\_value_1 = 100]$$

Running Example (2)

```
Square (int w) : Rectangle(w,w)
{ width = w; }

int area(void) { return width*width; }
```

```
Shape *sqre = new Square(10);
assert (sq
```

Internal SMT representation

$$C := \begin{bmatrix} j_1 = store(j_0, vtable, \text{Rectanle}) \\ \wedge\ j_2 = store(j_1, width, 10) \\ \wedge\ j_3 = store(j_2, height, 10) \\ \wedge\ j_4 = store(j_3, vtable, \text{Square}) \\ \wedge\ j_5 = store(j_4, width, 10) \\ \wedge\ return\_value_1 = \\ (select(j_5, width) \times select(j_5, width) \end{bmatrix}$$

$$P := [return\_value_1 = 100]$$

Running Example (2)

```
Square (int w) : Rectangle(w,w)
...
... *width; }
```

contain the address of the object's bound methods

```
Shape *sqre = new Square(10);
assert (sqre->area() == 100);
```

$$C := \begin{bmatrix} j_1 = store(j_0, vtable, \text{Rectanle}) \\ \wedge\, j_2 = store(j_1, width, 10) \\ \wedge\, j_3 = store(j_2, height, 10) \\ \wedge\, j_4 = store(j_3, vtable, \text{Square}) \\ \wedge\, j_5 = store(j_4, width, 10) \\ \wedge\, return\_value_1 = \\ (select(j_5, width) \times select(j_5, width) \end{bmatrix}$$

$$P := [return\_value_1 = 100]$$

Exception Handling (1)

- exceptions are unexpected situations within a C++ programs
  - access an invalid position in a vector throws an *out_of_range* exception
- exception handling is divided into three elements: a **try** block, a **catch** block, and a **throw** statement

```
int main (void) {
  try {
    throw 1;
  }
  catch (int) { return 1; }
  catch (char) { return 2; }
  return 0;
}
```

try block

throw statement

catch block

Exception Handling (2)

try-catch conversion to **goto** functions (internal flow)

```
main():
        CATCH signed_int->1, char->2
        THROW signed_int: 1
        CATCH
        GOTO 3

 1:     int #anon;
        return 1;
        GOTO 3

2:      char #anon;
        return 2;

3:      return 0;
        END_FUNCTION
```

jump when the type is *signed int*

jump when the type is *char*

This goto instruction is modified if an exception is thrown

Exception Handling (2)

try-catch conversion to **goto** functions (internal flow)

```
main():
        CATCH signed_int->1, char->2
        THROW signed_int: 1
        CATCH
        GOTO 1

1:      int #anon;
        return 1;
        GOTO 3

2:      char #anon;
        return 2;

3:      return 0;
        END_FUNCTION
```

This goto instruction is modified if an exception is thrown

Experimental Results

- Goal: compare the efficiency of C++ verification on 1165 C++ programs using ESBMC and LLBMC

- Setup:
  - ESBMC v1.20 with SMT Solver Z3 3.2
  - LLBMC 2012.2a
  - Intel Core i7-2600, 3.40 GHz with 24 GB of RAM running Ubuntu 64-bits

About the benchmarks

		N	L	Time	P	N	FP	FN	FAIL	TO	MO
[illegible]	[illegible]	[illegible]	3376	996	63	38	[illegible]	[illegible]	[illegible]	0	0
2	Deque	43	[illegible]	238	19	20	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	6853	[illegible]	95	37	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	2292	[illegible]	25	25	[illegible]	[illegible]	0	0	0
[illegible]	[illegible]	14	328	177	7	7	[illegible]	[illegible]	[illegible]	0	0
[illegible]	[illegible]	12	[illegible]	82	[illegible]	6	[illegible]	[illegible]	[illegible]	0	0
[illegible]	[illegible]	51	[illegible]	311	[illegible]	17	[illegible]	[illegible]	[illegible]	0	0
[illegible]	[illegible]	67	[illegible]	45	[illegible]	41	[illegible]	[illegible]	[illegible]	0	0
9	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0
10	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0
11	Cpp	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4	0	0
		1165	**55953**	**58386**	**680**	**354**	**42**	**92**	**7**	**0**	**0**

Number of programs

Lines of code

Verification time of the modules (s)

Positive verification **GOOD THING**

Negative verification **GOOD THING**

Negative verification **BAD THING**

Negative verification **BAD THING**

Crash **BAD THING**

Time out **BAD THING**

Memory out **BAD THING**

Experimental Results with ESBMC

	Testsuite	N	L	Time	P	N	FP	FN	FAIL	TO	MO
1	**Algorithm**	**130**	**3376**	**996**	**63**	**38**	[illegible]	[illegible]	[illegible]	**0**	**0**
2	**Deque**	**43**	**1239**	**238**	**19**	[illegible]	[illegible]	[illegible]	[illegible]	**0**	**0**
3	**Vector**	**146**	**6853**	**2714**	**95**	[illegible]	**3**	**11**	**0**	**0**	**0**
4	**List**	**670**	**2292**	**3928**	**25**	**25**	**3**	**17**	**0**	**0**	**0**
5	**Queue**	**14**	**328**	**177**	**7**	**7**	**0**	**0**	**0**	**0**	**0**
6	**Stack**	**12**	**286**	**82**	**6**	**6**	**0**	**0**	**0**	**0**	**0**
7	Inheritance	51	3460	311	28	17	1	2	3	0	0
8	Try catch	67	4743	45	17	41	7	2	0	0	0
9	Stream	66	1831	1892	51	13	0	2	0	0	0
10	String	233	4921	48186	100	112	5	16	0	0	0
11	Cpp	343	26624	1817	269	38	7	25	4	0	0
		1165	**55953**	**58386**	**680**	**354**	**42**	**92**	**7**	**0**	**0**

STL modules

Experimental Results with ESBMC

	Testsuite	N	L	Time	P	N	FP	FN	FAIL	TO	MO
1	Algorithm	130	3376	996	63	38	16	13	0	0	0
2	Deque	43	1239	238	19	20	0	4	0	0	0
3	Vector	146	6853	2714	95	37	3	11	0	0	0
4	List	670	2292	3928	25	25	3	17	0	0	0
5	Queue	14	328	177	7	7	[illegible]	[illegible]	[illegible]	0	0
6	Stack	12	286	82	6	6	[illegible]	[illegible]	[illegible]	0	0
7	**Inheritance**	**51**	**3460**	**311**	**28**	**17**	**1**	**2**	**3**	**0**	**0**
8	**Try catch**	**67**	**4743**	**45**	**17**	**41**	**7**	**2**	**0**	**0**	**0**
9	Stream	66	1831	1892	51	13	0	2	0	0	0
10	String	233	4921	48186	100	112	5	16	0	0	0
11	Cpp	343	26624	1817	269	38	7	25	4	0	0
		1165	**55953**	**58386**	**680**	**354**	**42**	**92**	**7**	**0**	**0**

Inheritance and exception handling

Experimental Results with ESBMC

	Testsuite	N	L	Time	P	N	FP	FN	FAIL	TO	MO
1	Algorithm	130	3376	996	63	38	16	13	0	0	0
2	Deque	43	1239	238	19	20	0	4	0	0	0
3	Vector	146	6853	2714	95	37	3	11	0	0	0
4	List	670	2292	3928	25	25	3	17	0	0	0
5	Queue	14	328	177	7	7	0	0	0	0	0
6	Stack	12	286	82	6	6	0	0	0	0	0
7	Inheritance	51	3460	311	[illegible]	[illegible]	[illegible]	[illegible]	3	0	0
8	Try catch	67	4743	45	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0
9	**Stream**	**66**	**1831**	**1892**	**51**	**13**	**0**	**2**	**0**	**0**	**0**
10	**String**	**233**	**4921**	**48186**	**100**	**112**	**5**	**16**	**0**	**0**	**0**
11	Cpp	343	26624	1817	269	38	7	25	4	0	0
		1165	**55953**	**58386**	**680**	**354**	**42**	**92**	**7**	**0**	**0**

I/O Streams
Strings

Experimental Results with ESBMC

	Testsuite	N	L	Time	P	N	FP	FN	FAIL	TO	MO
1	Algorithm	130	3376	996	63	38	16	13	0	0	0
2	Deque	43	1239	238	19	20	0	4	0	0	0
3	Vector	146	6853	2714	95	37	3	11	0	0	0
4	List	670	2292	3928	25	25	3	17	0	0	0
5	Queue	14	328	177	7	7	0	0	0	0	0
6	Stack	12	286	82	6	6	0	0	0	0	0
7	Inheritance	51	3460	311	28	17	1	2	3	0	0
8	Try catch	67	4743	45	17	41	7	2	0	0	0
9	Stream	66	1831	[illegible]	[illegible]	[illegible]	0	2	0	0	0
10	String	233	4921	[illegible]	100	112	5	16	0	0	0
11	**Cpp**	**343**	**26624**	**1817**	**269**	**38**	**7**	**25**	**4**	**0**	**0**
		1165	**55953**	**58386**	**680**	**354**	**42**	**92**	**7**	**0**	**0**

Generic programs from Deitel

Comparison between ESBMC and LLBMC

	Testsuite	Time	P	N	FP	FN	FAIL	TO	MO
1	Algorithm	996	63	38	16	13	0	0	0
2	Deque	238	19	20	0	4	0	0	0
3	Vector	2714	95	37	3	11	0	0	0
4	List	3928	25	25	3	17	0	0	0
5	Queue	177	7	7	0	0	0	0	0
6	Stack	82	6	6	0	0	0	0	0
		8135	**215**	**133**	**22**	**45**	**0**	**0**	**0**
1	Algorithm	22964	53	45	1	5	0	24	2
2	Deque	8585	16	17	0	0	1	9	0
3	Vector	7234	91	38	1	3	4	6	3
4	List	2562	5	26	5	30	0	0	4
5	Queue	45	6	7	0	1	0	0	0
6	Stack	45	6	6	0	0	0	0	0
		41435	**177**	**139**	**7**	**39**	**5**	**39**	**9**

ESBMC

LLBMC

Comparison between ESBMC and LLBMC

	Testsuite	Time	P	N	FP	FN	FAIL	TO	MO
1	Inheritance	311	28	17	1	2	3	0	0
2	Try catch	45	17	41	7	2	0	0	0
		356	**45**	**58**	**8**	**4**	**3**	**0**	**0**
1	Inheritance	122	32	12	1	3	3	0	0
2	Try catch	4	0	1	0	0	66	0	0
		126	**32**	**13**	**1**	**3**	**69**	**0**	**0**

ESBMC

LLBMC

Comparison between ESBMC and LLBMC

	Testsuite	Time	P	N	FP	FN	FAIL	TO	MO
ESBMC									
1	Stream	1892	51	13	0	2	0	0	0
2	String	46186	100	112	5	16	0	0	0
		48078	**151**	**125**	**5**	**18**	**0**	**0**	**0**
LLBMC									
1	Stream	11	17	13	0	35	1	0	0
2	String	37	6	121	4	102	0	0	0
		48	**23**	**134**	**4**	**137**	**1**	**0**	**0**

Comparison between ESBMC and LLBMC

	Testsuite	Time	P	N	FP	FN	FAIL	TO	MO
1	Cpp	1817	269	38	7	25	4	0	0
		58386	**680**	**354**	**42**	**92**	**7**	**0**	**0**
1	Cpp	3260	235	24	10	56	15	2	1
		44869	**467**	**310**	**22**	**235**	**90**	**41**	**10**

ESBMC

LLBMC

- ESBMC++ took approximately 16 hours and successfully verified 1046 out of 1165 (**89%**)
- LLBMC took approximately 12 hours and successfully verified 777 out of 1165 (**66%**)

Experimental Results Sniffer Code

- ESBMC++ was used to verify a **commercial** application provided by Nokia Institute of Technology (INdT)

- The sniffer code contains 20 classes, 85 methods, and approximately 2839 lines of C++ code

- Five bugs were identified that were related to arithmetic under- and over-flow. The bugs were later confirmed by the developers

Conclusions

- SMT-based verification of C++ programs by focusing on the major features of the language
- Described the implementation of STL containers, inheritance, polymorphism and exception handling
  - in particular, exception specification, which is a feature that is not supported by others BMC tools
- ESBMC++ outperforms LLBMC if we consider the verification of C++ programs
  - with increased accuracy (i.e. exception enabled verification)
- Also, ESBMC++ was able to find undiscovered bugs in the sniffer code, a commercial application