ECBS 2013

SMT-Bounded Model Checking of C++ Programs

Mikhail Ramalho, Mauro Freitas, Felipe Sousa, Hendrio Marques, Lucas Cordeiro, Bernd Fischer
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 $\psi$ such that
  $\psi$ satisfiable iff $\phi$ 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
Objective of this work

- **Extend BMC to support complex features of C++**

- exploit background theories of Satisfiability Modulo Theories (SMT) solvers

- provide suitable encodings for
  - template
  - exception handling
  - containers
  - 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)
  - standard 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
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;
}
```

```c
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
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)

![Diagram of container model and iterators]

- $pos$ contains the index value pointed by the iterator in the container
- `source` points to the underlying container
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

\[ C((c', i')) = c.\text{erase}(i) := \]
\[ \land c'.\text{size} = c.\text{size} - 1 \]
\[ \land c'.\text{array} = \text{store}(...(\text{store}(c.\text{array}, i.\text{pos}, \text{select}(c.\text{array}, i.\text{pos} + 1)), ...), c.\text{size} - 2, \text{select}(c.\text{array}, c.\text{size} - 1)) \]
\[ \land i'.\text{source} = c' \]
\[ \land i'.\text{pos} = i.\text{pos} \]

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 contrast 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 \(<C, <_s, <_r>\) where \(C\) is the set of classes
  - shared inheritance \(<_s \subseteq C \times C\)
  - replicated inheritance \(<_r \subseteq C \times C\)
- square class relation: \(<C, \emptyset, \{(Square, Rectangle, Shape), (Square, Rectangle, Display)\}>\n  - 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 constructor and area method

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

\[
C := \begin{cases} 
    j_1 = \text{store}(j_0, \text{vtable}, \text{Rectangle}) \\
    \land j_2 = \text{store}(j_1, \text{width}, 10) \\
    \land j_3 = \text{store}(j_2, \text{height}, 10) \\
    \land j_4 = \text{store}(j_3, \text{vtable}, \text{Square}) \\
    \land j_5 = \text{store}(j_4, \text{width}, 10) \\
    \land \text{return\_value} = \\
    (\text{select}(j_5, \text{width}) \times \text{select}(j_5, \text{width})) \\
\end{cases}
\]

\[
P := [\text{return\_value} = 100]
\]
Running Example (2)

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

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

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

Instantiation of square and area call

```
C :=
\[
\begin{align*}
  j_1 &= \text{store}(j_0, vtable, \text{Rectangle}) \\
  \land j_2 &= \text{store}(j_1, \text{width}, 10) \\
  \land j_3 &= \text{store}(j_2, \text{height}, 10) \\
  \land j_4 &= \text{store}(j_3, vtable, \text{Square}) \\
  \land j_5 &= \text{store}(j_4, \text{width}, 10) \\
  \land return\_value_1 = \\
      (\text{select}(j_5, \text{width}) \times \text{select}(j_5, \text{width})) \\
\end{align*}
\]
```

\[ P := [\text{return\_value}_1 = 100] \]
Running Example (2)

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

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

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

Internal SMT representation

\[
C := \begin{cases} 
  j_1 = \text{store}(j_0, \text{vtable}, \text{Rectangle}) \\
  j_2 = \text{store}(j_1, \text{width}, 10) \\
  j_3 = \text{store}(j_2, \text{height}, 10) \\
  j_4 = \text{store}(j_3, \text{vtable}, \text{Square}) \\
  j_5 = \text{store}(j_4, \text{width}, 10) \\
  \text{return\_value}_1 = (\text{select}(j_5, \text{width}) \times \text{select}(j_5, \text{width})) \\
\end{cases}
\]

\[ P := [\text{return\_value}_1 = 100] \]
Running Example (2)

Square \( (\text{int} \ w) \) : Rectangle \( (w,w) \)

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

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

C :=
\[
\begin{align*}
& j_1 = \text{store}(j_0, \text{vtable}, \text{Rectanle}) \\
& \land j_2 = \text{store}(j_1, \text{width}, 10) \\
& \land j_3 = \text{store}(j_2, \text{height}, 10) \\
& \land j_4 = \text{store}(j_3, \text{vtable}, \text{Square}) \\
& \land j_5 = \text{store}(j_4, \text{width}, 10) \\
& \land \text{return}\_\text{value}_1 = \\
& (\text{select}(j_5, \text{width}) \times \text{select}(j_5, \text{width})
\end{align*}
\]

\[P := [\text{return}\_\text{value}_1 = 100] \]
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

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

try-catch conversion to goto functions (internal flow)

```plaintext
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
```

This goto instruction is modified if an exception is thrown.
**Exception Handling (2)**

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

```plaintext
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

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>L</th>
<th>Time</th>
<th>P</th>
<th>N</th>
<th>FP</th>
<th>FN</th>
<th>FAIL</th>
<th>TO</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Algorithm</td>
<td>130</td>
<td>3376</td>
<td>996</td>
<td>63</td>
<td>38</td>
<td>16</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2. Deque</td>
<td>43</td>
<td>1239</td>
<td>238</td>
<td>19</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3. Vector</td>
<td>146</td>
<td>6853</td>
<td>274</td>
<td>95</td>
<td>37</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4. List</td>
<td>670</td>
<td>2292</td>
<td>928</td>
<td>25</td>
<td>25</td>
<td>3</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5. Queue</td>
<td>14</td>
<td>328</td>
<td>177</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6. Stack</td>
<td>12</td>
<td>286</td>
<td>82</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7. Inheritance</td>
<td>51</td>
<td>3460</td>
<td>311</td>
<td>28</td>
<td>17</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8. Try catch</td>
<td>67</td>
<td>743</td>
<td>45</td>
<td>41</td>
<td>41</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9. Stream</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10. String</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11. Cpp</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Number of programs
- **Deque**: 2 programs
- **Vector**: 3 programs
- **List**: 4 programs
- **Queue**: 5 programs
- **Stack**: 6 programs
- **Inheritance**: 7 programs
- **Try catch**: 8 programs
- **Stream**: 9 programs
- **String**: 10 programs
- **Cpp**: 11 programs

### Lines of code
- **Deque**: 43 lines
- **Vector**: 146 lines
- **List**: 670 lines
- **Queue**: 14 lines
- **Stack**: 12 lines
- **Inheritance**: 51 lines
- **Try catch**: 67 lines
- **Stream**: 3 lines
- **String**: 2 lines
- **Cpp**: 4 lines

### Verification time of the modules (s)
- **Positive verification**: GOOD THING
- **Negative verification**: BAD THING

### Time out
- **BAD THING**

### Memory out
- **BAD THING**

### Crash
- **BAD THING**
# Experimental Results with ESBMC

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

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

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

<table>
<thead>
<tr>
<th>Testsuite</th>
<th>N</th>
<th>L</th>
<th>Time</th>
<th>P</th>
<th>N</th>
<th>FP</th>
<th>FN</th>
<th>FAIL</th>
<th>TO</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Algorithm</td>
<td>130</td>
<td>3376</td>
<td>996</td>
<td>63</td>
<td>38</td>
<td>16</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 Deque</td>
<td>43</td>
<td>1239</td>
<td>238</td>
<td>19</td>
<td>20</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3 Vector</td>
<td>146</td>
<td>6853</td>
<td>2714</td>
<td>95</td>
<td>37</td>
<td>3</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4 List</td>
<td>670</td>
<td>2292</td>
<td>3928</td>
<td>25</td>
<td>25</td>
<td>3</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5 Queue</td>
<td>14</td>
<td>328</td>
<td>177</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6 Stack</td>
<td>12</td>
<td>286</td>
<td>82</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7 Inheritance</td>
<td>51</td>
<td>3460</td>
<td>311</td>
<td>28</td>
<td>17</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8 Try catch</td>
<td>67</td>
<td>4743</td>
<td>451</td>
<td>17</td>
<td>11</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9 Stream</td>
<td>66</td>
<td>1831</td>
<td>13</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10 String</td>
<td>233</td>
<td>4921</td>
<td>1569</td>
<td>38</td>
<td>38</td>
<td>5</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11 Cpp</td>
<td>343</td>
<td>2662</td>
<td>1817</td>
<td>269</td>
<td>38</td>
<td>7</td>
<td>25</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1165</td>
<td>55953</td>
<td>58386</td>
<td>680</td>
<td>354</td>
<td>42</td>
<td>92</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Comparison between ESBMC and LLBMC

<table>
<thead>
<tr>
<th>Testsuite</th>
<th>Time</th>
<th>P</th>
<th>N</th>
<th>FP</th>
<th>FN</th>
<th>FAIL</th>
<th>TO</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Algorithm</td>
<td>996</td>
<td>63</td>
<td>38</td>
<td>16</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 Deque</td>
<td>238</td>
<td>19</td>
<td>20</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3 Vector</td>
<td>2714</td>
<td>95</td>
<td>37</td>
<td>3</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4 List</td>
<td>3928</td>
<td>25</td>
<td>25</td>
<td>3</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5 Queue</td>
<td>177</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6 Stack</td>
<td>82</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>8135</strong></td>
<td><strong>215</strong></td>
<td><strong>133</strong></td>
<td><strong>22</strong></td>
<td><strong>45</strong></td>
<td><strong>0</strong></td>
<td><strong>0</strong></td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testsuite</th>
<th>Time</th>
<th>P</th>
<th>N</th>
<th>FP</th>
<th>FN</th>
<th>FAIL</th>
<th>TO</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Algorithm</td>
<td>22964</td>
<td>53</td>
<td>45</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>2 Deque</td>
<td>8585</td>
<td>16</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>3 Vector</td>
<td>7234</td>
<td>91</td>
<td>38</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4 List</td>
<td>2562</td>
<td>5</td>
<td>26</td>
<td>5</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5 Queue</td>
<td>45</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6 Stack</td>
<td>45</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>41435</strong></td>
<td><strong>177</strong></td>
<td><strong>139</strong></td>
<td><strong>7</strong></td>
<td><strong>39</strong></td>
<td><strong>5</strong></td>
<td><strong>39</strong></td>
<td><strong>9</strong></td>
</tr>
</tbody>
</table>
## Comparison between ESBMC and LLBMC

<table>
<thead>
<tr>
<th>Testsuite</th>
<th>Time</th>
<th>P</th>
<th>N</th>
<th>FP</th>
<th>FN</th>
<th>FAIL</th>
<th>TO</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Inheritance</td>
<td>311</td>
<td>28</td>
<td>17</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 Try catch</td>
<td>45</td>
<td>17</td>
<td>41</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>356</td>
<td>45</td>
<td>58</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 Inheritance</td>
<td>122</td>
<td>32</td>
<td>12</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 Try catch</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>66</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>126</td>
<td>32</td>
<td>13</td>
<td>1</td>
<td>3</td>
<td>69</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Comparison between ESBMC and LLBMC

<table>
<thead>
<tr>
<th>Testsuite</th>
<th>Time</th>
<th>P</th>
<th>N</th>
<th>FP</th>
<th>FN</th>
<th>FAIL</th>
<th>TO</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream</td>
<td>1892</td>
<td>51</td>
<td>13</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>String</td>
<td>46186</td>
<td>100</td>
<td>112</td>
<td>5</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Stream</td>
<td>11</td>
<td>17</td>
<td>13</td>
<td>0</td>
<td>35</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>String</td>
<td>37</td>
<td>6</td>
<td>121</td>
<td>4</td>
<td>102</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>23</td>
<td>134</td>
<td>4</td>
<td>137</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Comparison between ESBMC and LLBMC

<table>
<thead>
<tr>
<th>Testsuite</th>
<th>Time</th>
<th>P</th>
<th>N</th>
<th>FP</th>
<th>FN</th>
<th>FAIL</th>
<th>TO</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cpp</td>
<td>1817</td>
<td>269</td>
<td>38</td>
<td>7</td>
<td>25</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 Cpp</td>
<td>58386</td>
<td>680</td>
<td>354</td>
<td>42</td>
<td>92</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 Cpp</td>
<td>3260</td>
<td>235</td>
<td>24</td>
<td>10</td>
<td>56</td>
<td>15</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1 Cpp</td>
<td>44869</td>
<td>467</td>
<td>310</td>
<td>22</td>
<td>235</td>
<td>90</td>
<td>41</td>
<td>10</td>
</tr>
</tbody>
</table>

- 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