ESBMC 1.22
(Competition Contribution)

Jeremy Morse, Mikhail Ramalho, Lucas Cordeiro, Denis Nicole, Bernd Fischer
ESBMC: SMT-based BMC of single- and multi-threaded software

• exploits SMT solvers and their background theories:
  – optimized encodings for pointers, bit operations, unions and arithmetic over- and underflow
  – efficient search methods (non-chronological backtracking, conflict clauses learning)

• supports verifying multi-threaded software that uses pthreads threading library
  – interleaves only at “visible” instructions
  – lazy exploration of the reachability tree
  – optional context-bound

• derived from CBMC (v2.9) and has inherited its object-based memory model
ESBMC verification support

- built-in properties:
  - arithmetic under- and overflow
  - pointer safety
  - array bounds
  - division by zero
  - memory leaks
  - atomicity and order violations
  - deadlocks
  - data races

- user-specified assertions
  (\_\_ESBMC\_assume, \_\_ESBMC\_assert)

- built-in scheduling functions
  (\_\_ESBMC\_atomic\_begin, \_\_ESBMC\_atomic\_end, \_\_ESBMC\_yield)
Differences to ESBMC 1.20

• ESBMC 1.22 is largely a **bugfixing release**:
  – memory handling
  – replaced CBMC’s string-based accessor functions
    → increased ESBMC’s speed by 2x

• improved **memory model** for precision, performance, and stability

• produces a **smaller number of false results**
  – more errors detected (+109), fewer unexpected (-15) and missed (-157) errors
ESBMC’s memory model

• statically tracks possible pointer variable targets (objects)
  – dereferencing a pointer leads to the construction of guarded references to each potential target

• C is very liberal about permitted dereferences

```c
struct foo {
    uint16_t bar[2];
    uint8_t baz;
};
struct foo qux;
char *quux = &qux;
quux++;
*quux;  // pointer and object types do not match
```

• SAT: immediate access to bit-level representation
ESBMC’s memory model

- statically tracks possible pointer variable targets (objects)
  - dereferencing a pointer leads to the construction of guarded references to each potential target

- C is very liberal about permitted dereferences

```c
struct foo {
    uint16_t bar[2];
    uint8_t baz;
};
```

```c
struct foo qux;
char *quux = &qux;
quux++;
*quux; ← pointer and object types do not match
```

- SMT: sorts must be repeatedly unwrapped
Byte-level data extraction in SMT

• access to underlying data bytes is complicated
  – requires manipulation of arrays / tuples

• problem is magnified by nondeterministic offsets

```c
uint16_t *fuzz;
if (nondet_bool()) {
  fuzz = &qux.bar[0];
} else {
  fuzz = &qux.baz;
}
```

─ chooses accessed field nondeterministically
─ requires a `byte_extract` expression
─ handles the `tuple` that encoded the `struct`

• supporting all legal behaviors at SMT layer difficult
  – extract (unaligned) 16bit integer from *fuzz

• experiments showed significantly increased memory consumption
“Aligned” Memory Model

• framework cannot easily be changed to SMT-level byte representation (a la LLBMC)
• push unwrapping of SMT data structures to dereference
• enforce C alignment rules
  – static analysis of pointer alignment eliminates need to encode unaligned data accesses
    → reduces number of behaviors that must be modeled
  – add alignment assertions (if static analysis not conclusive)
  – extracting 16-bit integer from *fuzz:
    – offset = 0: project bar[0] out of foo
    – offset = 1: “unaligned memory access” failure
    – offset = 2: project bar[1] out of foo
    – offset = 3: “unaligned memory access” failure
    – offset = 4: “access to object out of bounds” failure
Strengths:

- robust context-bounded model checker for C programs
- improved memory model to handle pointer arithmetic
  - greater accuracy and faster verification

Weaknesses:

- all unexpected results are caused by
  - bounding the programs (**Recursive**)
  - differences in the memory models (**MemorySafety**)
    → ESBMC detects an unchecked dereference of a pointer to a freshly allocated memory chunk