CBMC by example

Maria João Frade

HASLab - INESC TEC
Dep. Informática, Univ. Minho

Assertions [ex1.c]

CBMC checks assertions as defined by the ANSI-C standard.
The assert statement takes a Boolean condition, and CBMC checks that this condition is true for all runs of the program.

```
void main (void)
{
  int x;
  int y=8, z=0, w=0;

  if (x)
    z = y - 1;
  else
    w = y + 1;

  assert (z == 7 || w == 9);
}
```

```
$ cbmc ex1.c
$ cbmc ex1.c --show-vcc
```

ex1.c outcome

```
$ cbmc ex1.c
```

```
CBMC version 4.9 64-bit macos
file ex1.c: Parsing
Converting
Type-checking ex1
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
size of program expression: 43 steps
simple slicing removed 2 assignments
Generated 1 VCC(s), 1 remaining after simplification
Passing problem to propositional reduction
Running propositional reduction
Post-processing
Solving with MiniSAT 2.2.0 with simplifier
147 variables, 65 clauses
SAT checker: negated claim is UNSATISFIABLE, i.e., holds
Runtime decision procedure: 0.007s
VERIFICATION SUCCESSFUL
```

ex1.c outcome

```
$ cbmc ex1.c --show-vcc
```

```
Generated 1 VCC(s), 1 remaining after simplification

VERIFICATION CONDITIONS:

file ex1.c line 11 function main
assertion z == 7 || w == 9
(…)
{-12} y!0@1#2 == 8
{-13} z!0@1#2 == 0
{-14} w!0@1#2 == 0
{-15} \guard#1 == !(x!0@1#1 == 0)
{-16} z!0@1#3 == 7
{-17} z!0@1#4 == 0
{-18} w!0@1#3 == 9
{-19} z!0@1#5 == (\guard#1 ? 7 : 0)
{-20} w!0@1#4 == (\guard#1 ? 0 : 9)
|--------------------------
{1} w!0@1#4 == 9 || z!0@1#5 == 7
```

ex2.c

```
void main (void)
{
  int x;
  int y=8, z=0, w=0;

  if (x)
    z = y - 1;
  else
    w = y + 1;

  assert (z == 5 || w == 9);
}
```

```
$ cbmc ex2.c
$ cbmc ex2.c --show-vcc
```

ex2.c outcome

```
$ cbmc ex2.c
```

```
Generated 1 VCC(s), 1 remaining after simplification
Passing problem to propositional reduction
Running propositional reduction
Post-processing
Solving with MiniSAT 2.2.0 with simplifier
147 variables, 65 clauses
SAT checker: negated claim is SATISFIABLE, i.e., does not hold
Runtime decision procedure: 0.006s
Building error trace

Counterexample:
(…)
State 17 file ex2.c line 3 function main thread 0
----------------------------------------------------
  x=32768 (00000000000000001000000000000000)

(…)
State 19 file ex2.c line 4 function main thread 0
----------------------------------------------------
  y=8 (00000000000000000000000000001000)
```

ex2.c outcome

```
$ cbmc ex2.c
```

```
(…)
State 23 file ex2.c line 4 function main thread 0
----------------------------------------------------
  w=0 (00000000000000000000000000000000)

State 25 file ex2.c line 7 function main thread 0
----------------------------------------------------
  z=7 (00000000000000000000000000000111)

Violated property:
  file ex2.c line 11 function main
  assertion z == 5 || w == 9
  z == 5 || w == 9

VERIFICATION FAILED
```

ex3.c

```
void main (void)
{
  int x, y;

  x = x + y;
  if (x != 3) x = 2;
  else  x++;

  assert (x <= 3);
}
```

```
$ cbmc ex3.c --show-vcc
$ cbmc ex3.c
$ cbmc ex3.c --no-assertions
$ cbmc ex3.c --no-assertions --show-vcc
```

Checking overflow

But the code can be automatically instrumented.

```
$ cbmc ex3.c --signed-overflow-check
             --no-assertions
```

```
(…)
State 17 file ex3.c line 3 function main thread 0
------------------------------------------------------
  x=-2146959360 (10000000000010000000000000000000)

State 18 file ex3.c line 3 function main thread 0
------------------------------------------------------
  y=-2147483648 (10000000000000000000000000000000)

Violated property:
  file ex3.c line 5 function main
  arithmetic overflow on signed +
  !overflow("+", signed int, x, y)
```

Workflow

- Internally CBMC runs `goto-cc` to produce a binary representation of the control flow graph of the program.

```
$ goto-cc ex3.c -o ex3.gb
```

- Then the instrumentation tool `goto-instrument` automatically add assertions to be checked.

```
$ goto-instrument --signed-overflow-check ex3.gb
ex3.instr.gb
```

- And finally the assertions are checked.

```
$ cbmc ex3.instr.gb
```

Seeing the properties

```
$ cbmc ex3.c --signed-overflow-check --show-properties
```

```
(…)
Generic Property Instrumentation
Property main.1:
  file ex3.c line 5 function main
  arithmetic overflow on signed +
  !overflow("+", signed int, x, y)
  in "x + y"

Property main.2:
  file ex3.c line 7 function main
  arithmetic overflow on signed +
  !overflow("+", signed int, x, 1)
  in "x + 1"

Property main.3:
  file ex3.c line 9 function main
  assertion x <= 3
  x <= 3
```

Seeing the instrumented code

```
$ cbmc ex3.c --signed-overflow-check
             --show-goto-functions
```

```
(…)
main /* c::main */
        // 15 file ex3.c line 3 function main
        signed int x;
        // 16 file ex3.c line 3 function main
        signed int y;
        // 17 file ex3.c line 5 function main
        ASSERT !overflow("+", signed int, x, y) // arithmetic overflow on signed +
        // 18 file ex3.c line 5 function main
        x = x + y;
        // 19 file ex3.c line 6 function main
        IF !(x != 3) THEN GOTO 1
        // 20 file ex3.c line 6 function main
        x = 2;
        // 21 file ex3.c line 6 function main
        GOTO 2
        // 22 file ex3.c line 7 function main
    1: ASSERT !overflow("+", signed int, x, 1) // arithmetic overflow on signed +
        // 23 file ex3.c line 7 function main
        x = x + 1;
        // 24 file ex3.c line 9 function main
    2: ASSERT x <= 3 // assertion x <= 3
        // 25 file ex3.c line 10 function main
        dead y;
        // 26 file ex3.c line 10 function main
        dead x;
        // 27 file ex3.c line 10 function main
        END_FUNCTION
```

Entrypoints [ex4.c]

```
int fun (int a, int b)
{
  int c = a+b;

  if (a>0 || b>0)
    c = 1/(a+b);
  return c;
}
```

```
$ cbmc ex4.c
$ cbmc ex4.c --function fun
$ cbmc ex4.c --function fun --div-by-zero-check
```

Checking division by zero

```
$ cbmc ex4.c --function fun --div-by-zero-check
```

```
(…)
State 17 file ex4.c line 1 thread 0
------------------------------------------------------------
  a=2147475456 (01111111111111111111000000000000)

State 18 file ex4.c line 1 thread 0
------------------------------------------------------------
  b=-2147475456 (10000000000000000010000000000000)

State 19 file ex4.c line 3 function fun thread 0
------------------------------------------------------------
  c=0 (00000000000000000000000000000000)

State 20 file ex4.c line 3 function fun thread 0
------------------------------------------------------------
  c=0 (00000000000000000000000000000000)

Violated property:
  file ex4.c line 6 function fun
  division by zero
  a + b != 0

VERIFICATION FAILED
```

ex5.c

```
void main ()
{
  char c;
  long l;
  int i;

  l = c = i;
  assert (l==i);
}
```

```
$ cbmc ex5.c
```

ex5.c outcome

```
$ cbmc ex5.c
```

```
(…)
State 17 file ex5.c line 3 function main thread 0
------------------------------------------------------------
  c=0 (00000000)

State 18 file ex5.c line 4 function main thread 0
------------------------------------------------------------
  l=0 (0000000000000000000000000000000000000000000000000000000000000000)

State 19 file ex5.c line 5 function main thread 0
------------------------------------------------------------
  i=262144 (00000000000010000000000000000000)

State 20 file ex5.c line 7 function main thread 0
------------------------------------------------------------
  c=0 (00000000)

State 21 file ex5.c line 7 function main thread 0
------------------------------------------------------------
  l=0 (0000000000000000000000000000000000000000000000000000000000000000)

Violated property:
  file ex5.c line 8 function main
  assertion l == (signed long int)i
  l == (signed long int)i

VERIFICATION FAILED
```

Array bounds [ex6.c]

```
int puts (const char *s);

int main (int argc, char **argv)
{
  int i;

  if (argc >= 1)
    puts (argv[2]);
}
```

```
$ cbmc ex6.c
$ cbmc ex6.c --bounds-check --pointer-check
```

ex6.c outcome

```
$ cbmc ex6.c --bounds-check --pointer-check
```

```
(…)
State 17  thread 0
---------------------------------------------------
  argv'[1]=irep("(\"nil\")")[1] (?)

State 20 file ex6.c line 3 thread 0
---------------------------------------------------
  argc=1 (00000000000000000000000000000001)

State 21 file ex6.c line 3 thread 0
---------------------------------------------------
  argv=argv' (00000010000000000000000000000000000000000000000000000000000000)

State 22 file ex6.c line 5 function main thread 0
---------------------------------------------------
  i=0 (00000000000000000000000000000000)

State 25 file ex6.c line 8 function main thread 0
---------------------------------------------------
  s=((signed char *)NULL) (0000000000000000000000000000000000000000000000000000000000000000)

Violated property:
  file ex6.c line 8 function main
  dereference failure: object bounds
  !(16l + POINTER_OFFSET(argv) < 0) && OBJECT_SIZE(argv) >= 24 + POINTER_OFFSET(argv) ||
DYNAMIC_OBJECT(argv)

VERIFICATION FAILED
```

Array bounds [ex7.c]

```
int puts (const char *s);

int main (int argc, char **argv)
{
  int i;

  if (argc >= 2)
    puts (argv[2]);
}
```

```
$ cbmc ex7.c --bounds-check --pointer-check
```

```
(…)
Generated 6 VCC(s), 5 remaining after simplification
Passing problem to propositional reduction
Running propositional reduction
Post-processing
Solving with MiniSAT 2.2.0 with simplifier
951 variables, 2462 clauses
SAT checker: negated claim is UNSATISFIABLE, i.e., holds
Runtime decision procedure: 0.01s
VERIFICATION SUCCESSFUL
```

ex8.c

```
int array[10];

int sum ()
{
  unsigned i, sum;

  sum = 0;
  for (i = 0; i <= 10; i++)
    sum += array [i];
}
```

```
$ cbmc ex8.c --function sum
$ cbmc ex8.c --function sum --bounds-check
```

ex8.c outcome

```
$ cbmc ex8.c --function sum --bounds-check
```

```
(…)
State 10 file ex8.c line 1 thread 0
------------------------------------------------------
  array={ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

(…)

State 50 file ex8.c line 8 function sum thread 0
------------------------------------------------------
  i=10 (00000000000000000000000000001010)

Violated property:
  file ex8.c line 9 function sum
  array `array' upper bound
  (signed long int)i < 10l

VERIFICATION FAILED
```

Loop unwinding [ex9.c]

```
int binsearch (int x)
{
  int a[16];
  signed low = 0, high = 16;

  while (low < high) {
    signed middle = low + ((high - low) >> 1);
    if (a[middle]<x)  high = middle;
    else if (a [middle] > x)  low = middle + 1;
    else return middle;
  }
 return -1;
}
```

```
$ cbmc ex9.c --function binsearch
             --bounds-check --pointer-check
```

ex9.c outcome

```
$ cbmc ex9.c --function binsearch
             --bounds-check --pointer-check
```

CBMC does not stop! The loop is being infinitely unwound.
We must provide the number of iterations to be unwound.

```
$ cbmc ex9.c --function binsearch
             --bounds-check --pointer-check
             --unwind 4
```

```
(…)
Generated 17 VCC(s), 13 remaining after simplification
Passing problem to propositional reduction
(…)
Violated property:
  file ex9.c line 6 function binsearch
  unwinding assertion loop 0

VERIFICATION FAILED
```

Unwinding assertion

The failure of the "unwinding assertion" means that it is not guaranteed that the number k of iterations given as parameter will be sufficient, i.e. some execution path may run through n>k iterations.

In this case it suffices to increase k.

```
$ cbmc ex9.c --function binsearch
             --bounds-check --pointer-check
             --unwind 6
```

```
Generated 25 VCC(s), 21 remaining after simplification
Passing problem to propositional reduction
Running propositional reduction
Post-processing
Solving with MiniSAT 2.2.0 with simplifier
9291 variables, 37235 clauses
SAT checker: negated claim is UNSATISFIABLE, i.e., holds
Runtime decision procedure: 0.12s
VERIFICATION SUCCESSFUL
```

Bounded loops [ex10.c]

CBMC checks if enough unwinding is done.

```
int sumq (void)
{
  short int i, s;

  s = 0;
  for (i = 0; i <= 10; i++)
    s *= i*i;
  return s;
}
```

```
$ cbmc ex10.c --function sumq --signed-overflow-check
```

```
Generated 44 VCC(s), 33 remaining after simplification
(…)
Runtime decision procedure: 0.003s
VERIFICATION SUCCESSFUL
```

Unbounded loops [ex11.c]

```
$ cbmc ex11.c --function sumqq
              --signed-overflow-check --unwind 100
```

```
int sumqq (int x)
{
  short int i, s;

  s = 0;
  for (i = 0; i <= x; i++)
    s *= i*i;
  return s;
}
```

CBMC can also be used for programs with unbounded loops.
To disable the “unwinding assertion” test run with the switch

```
--no-unwinding-assertions
```

Unbounded loops [ex11.c]

```
$ cbmc ex11.c --function sumqq --signed-overflow-check
              --unwind 100 --no-unwinding-assertions
```

```
(…)
Generated 400 VCC(s), 300 remaining after simplification
(…)
Runtime decision procedure: 0.036s
VERIFICATION SUCCESSFUL
```

In this case CBMC is used for bug hunting only. CBMC does not attempt to find all bugs. In this case, if you increase the bound you can find a bug.

```
$ cbmc ex11.c --function sumqq --signed-overflow-check
              --unwind 200 --no-unwinding-assertions
```

```
(…)
Violated property:
  file ex11.c line 7 function sumqq
  arithmetic overflow on signed type conversion
  (signed int)i * (signed int)i <= 32767 && (signed int)i * (signed
int)i >= -32768

VERIFICATION FAILED
```

Recursion & Inlining [ex12.c]

```
void f (int a)
{
  if (a == 0)
    assert (1);
  else f (a - 1);
}

void main (void)
{
  f(5);
}
```

```
$ cbmc ex12.c --function f --unwind 100
```

```
(…)
Violated property:
  file ex12.c line 5 function f
  recursion unwinding assertion

VERIFICATION FAILED
```

Recursion & Inlining [ex12.c]

If called from main f will be inlined and unwound.
There is no need to provide `--unwind k` switch:

```
$ cbmc ex12.c
```

```
(…)
Generic Property Instrumentation
Starting Bounded Model Checking
Unwinding recursion f iteration 1
Unwinding recursion f iteration 2
Unwinding recursion f iteration 3
Unwinding recursion f iteration 4
Unwinding recursion f iteration 5
size of program expression: 57 steps
simple slicing removed 0 assignments
Generated 1 VCC(s), 0 remaining after simplification
VERIFICATION SUCCESSFUL
```

Low level properties [ex13.c]

```
int nondet_int();
int *p;
int global;

void f (void)
{
  int local = 10;
  int input = nondet_int();

  p = input ? &local : &global;
}
```

```
int main (void)
{
  int z;

  global = 10;
  f ();
  z = *p;
  assert (z==10);
}
```

```
$ cbmc ex13.c
```

VERIFICATION FAILED **Why?**

```
$ cbmc ex13.c --pointer-check --no-assertions
```

ex14.c

```
char s[] = "abc";

void main(void)
{
  char *p = s;
  p[1] = 'y';
  assert (s[1]=='y');
}
```

```
$ cbmc ex14.c --bounds-check --pointer-check
```

VERIFICATION SUCCESSFUL

ex16.c

```
void f (unsigned int n)
{
  int *p;
  p = malloc(sizeof(int)*n);
  p[n-1] = 0;
  free(p);
}
```

```
$ cbmc ex16.c --function f
```

VERIFICATION SUCCESSFUL

```
$ cbmc ex16.c --function f
              --bounds-check --pointer-check
```

VERIFICATION FAILED **Why?**

ex17.c

```
void f (_Bool i)
{
  int *p, y;

  p = malloc(sizeof(int)*10);
  if (i) p = &y;
  free(p);
}
```

```
$ cbmc ex17.c --function f

VERIFICATION FAILED
```

Why?

Assume-guarantee reasoning

In addition to the assert statement, CBMC provides the `__CPROVER_assume` statement.

The `__CPROVER_assume` statement restricts the program traces that are considered and allows assume-guarantee reasoning.

As an assertion, `__CPROVER_assume` takes a Boolean expression.

Intuitively, one can consider the `__CPROVER_assume` statement **to abort the program *successfully* if the condition is false. If the condition is true, the execution continues.**

ex18.c

```
int nondet_int();
int x, y;

void main (void)
{
  x = nondet_int();
  y = x+1;
  assert (y>x);
}
```

```
$ cbmc ex18.c
VERIFICATION FAILED
```

Why?

```
$ cbmc ex18.c --show-vcc
```

```
VERIFICATION CONDITIONS:
(…)
assertion y > x
(…)
{-8} x#1 == 0
{-9} y#1 == 0
(…)
{-14} x#2 == nondet_symbol(symex::nondet0)
{-15} y#2 == 1 + x#2
|-------------------------
{1} !(x#2 >= y#2)
```

ex19.c

```
int nondet_int();
int x, y;

void main (void)
{
  x = nondet_int();
  __CPROVER_assume (x<10);
  y = x+1;
  assert (y>x);
}
```

```
$ cbmc ex19.c
VERIFICATION SUCCESSFUL
```

```
$ cbmc ex19.c --show-vcc
```

```
(…)
VERIFICATION CONDITIONS:
(…)
assertion y > x
(…)
{-8} x#1 == 0
{-9} y#1 == 0
(…)
{-14} x#2 == nondet_symbol(symex::nondet0)
{-15} x#2 < 10
{-16} y#2 == 1 + x#2
|-------------------------
{1} !(x#2 >= y#2)
```