Example#1

Loop-Unwinding

Source-Code

C/C++ - File1_dividebyzero/src/File1.c - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- File1_dividebyzero
  - Includes
  - src
    - File1.c
      - main(void) : int
      - test_assert(int) : int
  - Debug

File1.c

```c
 Name          : File1_dividebyzero.c


int test_assert ( int x )
{
   assert( x <= 4 );
   return x;
}

int main ( void )
{
  int i;

    for (i=0; i<=9; i++){
        test_assert( i );
        printf("i = %d\n", i);
    }

  return 0;
}
```

Output:

Run in Eclipse:

File1.c

```c
 Name          : File1_dividebyzero.c

#include <stdio.h>
#include <conio.h>
#include <assert.h>


int test_assert ( int x )
{
   assert( x <= 4 );
   return x;
}

int main ( void )
{
  int i;

    for (i=0; i<=9; i++){
        test_assert( i );
        printf("i = %d\n", i);
    }

  return 0;
}
```

File1_dividebyzero.exe

File1_dividebyzero.exe has stopped working

A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.

Debug | Close program

Problems | Tasks | Console | Properties

File1_dividebyzero.exe [C/C++ Application] C:\Users\Pramod\workspace\CBMC_PROGRAMS\File1_dividebyzero\Debug\File1_divic

```
Assertion failed: x <= 4, file ..\src\File1.c, line 18

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

Run in CBMC Model Checking :-

For specific assertion Label :

C:\Users\Pramod\Desktop\CBMC_Demo>cbmc demo.c --show-claims

```
Command Prompt
C:\Users\Pramod\Desktop\CBMC_Demo>cbmc demo.c --show-claims
CBMC version 5.3 32-bit i386 windows
Parsing demo.c
demo.c
Converting
Type-checking demo
file demo.c line 4 function test_assert: function `assert' is not declared
file demo.c line 14 function main: function `printf' is not declared
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Property test_assert.assertion.1:
  file demo.c line 4 function test_assert
  assertion x <= 4
  x <= 4


C:\Users\Pramod\Desktop\CBMC_Demo>
```

For Detailed Output :

C:\Users\Pramod\Desktop\CBMC_Demo>cbmc demo.c --all-claims

```
Command Prompt
Parsing demo.c
demo.c
Converting
Type-checking demo
file demo.c line 4 function test_assert: function `assert' is not declared
file demo.c line 14 function main: function `printf' is not declared
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
Unwinding loop main.0 iteration 1 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 2 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 3 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 4 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 5 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 6 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 7 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 8 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 9 file demo.c line 12 function main thread 0
Unwinding loop main.0 iteration 10 file demo.c line 12 function main thread 0
size of program expression: 129 steps
simple slicing removed 11 assignments
Generated 10 VCC(s), 5 remaining after simplification
Passing problem to propositional reduction
converting SSA
Running propositional reduction
Post-processing
Solving with Glucose Syrup with simplifier
40 variables, 11 clauses
c |  Eliminated clauses:          0.00 Mb
c last restart ## conflicts  :  0 0
SAT checker: negated claim is SATISFIABLE, i.e., does not hold
Runtime decision procedure: 0.01s

** Results:
[test_assert.assertion.1] assertion x <= 4: FAILED

** 1 of 1 failed (1 iteration)
```

Non-determinism Unsignedchar Character

```
//cbmc unsignedchar.c

int main() {

unsigned char a,b;
unsigned int result = 0, i;

a = nondet_uchar();
b = nondet_uchar();

for(i=0; i<8; i++)
  if((b>>i)&1)
     result +=(a<<i);

assert(result==a*b);
}
```

```
C:\Users\Pramod\Desktop\CBMC_Demo\Modelling>cbmc unsignedchar.c
CBMC version 5.3 32-bit i386 windows
Parsing unsignedchar.c
unsignedchar.c
Converting
Type-checking unsignedchar
file unsignedchar.c line 6 function main: function `nondet_uchar' is not declared
file unsignedchar.c line 13 function main: function `assert' is not declared
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
Unwinding loop main.0 iteration 1 file unsignedchar.c line 9 function main thread 0
Unwinding loop main.0 iteration 2 file unsignedchar.c line 9 function main thread 0
Unwinding loop main.0 iteration 3 file unsignedchar.c line 9 function main thread 0
Unwinding loop main.0 iteration 4 file unsignedchar.c line 9 function main thread 0
Unwinding loop main.0 iteration 5 file unsignedchar.c line 9 function main thread 0
Unwinding loop main.0 iteration 6 file unsignedchar.c line 9 function main thread 0
Unwinding loop main.0 iteration 7 file unsignedchar.c line 9 function main thread 0
Unwinding loop main.0 iteration 8 file unsignedchar.c line 9 function main thread 0
size of program expression: 99 steps
simple slicing removed 3 assignments
Generated 1 VCC(s), 1 remaining after simplification
Passing problem to propositional reduction
converting SSA
Running propositional reduction
Post-processing
Solving with Glucose Syrup with simplifier
616 variables, 2056 clauses
c |  Eliminated clauses:           0.01 Mb
SAT checker: negated claim is UNSATISFIABLE, i.e., holds
Runtime decision procedure: 12.009s
VERIFICATION SUCCESSFUL
```

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.

Source Code:

```
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);
  }
```

Command:

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

Outcome:

```
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
```

ex2.c

Source Code:

```
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);
}
```

Command:

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

OutCome:

```
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
$ cbmc ex2.c
```

ex3.c

Source code:

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

```
if (x != 3) x = 2;
else x++;
assert (x <= 3);
}
```

Command:

```
$ cbmc ex3.c --show-vcc
$ cbmc ex3.c
$ 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.
- Then the instrumentation tool goto-instrument automatically add assertions to be checked.
- And finally the assertions are checked.

Command:

```
$ goto-cc ex3.c -o ex3.gb
$ goto-instrument --signed-overflow-check ex3.gb ex3.instr.gb
$ cbmc ex3.instr.gb
$ cbmc ex3.c --signed-overflow-check --show-properties
```

OutCome:

```
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
```

Coomand:

```
$ 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]

Source Code:

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

Command:

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

OutCome:

```
Checking division by zero
(…)
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
$ cbmc ex4.c --function fun --div-by-zero-check
```

ex5.c

Source Code:

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

Command:

```
$ cbmc ex5.c
```

Outcome:

```
(…)
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 (00000000000010000000000000000)
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
```
$ cbmc ex5.c

Array bounds [ex6.c]

Source Code:

```
int puts (const char *s);
int main (int argc, char **argv)
{
int i;
if (argc >= 1)
puts (argv[2]);
}
```

Command:

$ cbmc ex6.c

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

ex6.c outcome

```
(…)
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' (0000001000000000000000000000000000000000000000000000000000000000)
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]

Source Code:

```
int puts (const char *s);
int main (int argc, char **argv)
{
int i;
if (argc >= 2)
puts (argv[2]);
}
```

Command:

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

Outcome:

```
(…)
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

Soirce Code:

```
int array[10];
int sum ()
{
```

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

Command:

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

ex8.c outcome:

```
(…)
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]

Source Code:

```
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;
}
```

Commands:

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

Outcome:

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

```
(…)
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
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
```

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

Explanation: 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.

Bounded loops [ex10.c]

Source Code:

```
int sumq (void)
{
short int i, s;
s = 0;
for (i = 0; i <= 10; i++)
s *= i*i;
return s;
}
```

Command:

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

OutCome:

```
CBMC checks if enough unwinding is done.
Generated 44 VCC(s), 33 remaining after simplification
(…)
Runtime decision procedure: 0.003s
VERIFICATION SUCCESSFUL
```

Unbounded loops [ex11.c]

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

Source Code:

```
int sumqq (int x)
{
short int i, s;
s = 0;
for (i = 0; i <= x; i++)
s *= i*i;
return s;
}
```

Command:

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

OutCome:

```
(…)
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.
(…)
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]

Source Code:

```
void f (int a)
{
if (a == 0)
assert (1);
else f (a - 1);
}
void main (void)
{
f(5);
}
```

Command:

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

OutCome1:

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

Command:

```
$ cbmc ex12.c
```

OutCome2:

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

Low level properties [ex13.c]

Source Code:

```
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);
}
```

Command:

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

OutCome:

```
VERIFICATION FAILED
```

ex14.c

Source Code:

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

Command:

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

OutCome:

```
VERIFICATION SUCCESSFUL
```

ex16.c

Source Code:

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

Command:

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

OutCome:

VERIFICATION FAILED

ex17.c

Source Code:

```
void f (_Bool i)
{
int *p, y;
p = malloc(sizeof(int)*10);
if (i) p = &y;
free(p);
}
```

Command:

$ cbmc ex17.c --function f

OutCome:

VERIFICATION FAILED

Assume-guarantee reasoning

1) In addition to the assert statement, CBMC provides the __CPROVER_assume statement.
2) The __CPROVER_assume statement restricts the program traces that are considered and allows assume-guarantee reasoning.
3) 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

Source Code:

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

Command:

$ cbmc ex18.c

$ cbmc ex18.c --show-vcc

OutCome:

```
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)
VERIFICATION FAILED
```

ex19.c

Source Code:

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

Command:

$ cbmc ex19.c

$ cbmc ex19.c --show-vcc

Outcome:

```
(…)
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)
VERIFICATION SUCCESSFUL
```