CBMC 软件用户使用手册

二零一四年十一月二十三日

目录

1 引言

CBMC 全称是 a Bounded Model Checker for C and C++ programs。是一个 C/C++代码越界分析检查工具。截至二零一四年十一月二十三日，CBMC 的最新版本号为 4.9。

本使用手册，旨在向用户介绍了 CBMC 软件的运行环境、软件功能、软件安装以及使用方法等。

前三章主要是介绍 CBMC 的基本情况，以及 CBMC 的环境。第四章介绍了 CBMC 在 Windows 平台和 Linux 平台上的安装方法。第五章通过四个例子的方式详细介绍了 CBMC 常用的操作。第六章是一些延伸阅读材料。第七章是第五章四个例子的详细输出。用户可以有选择性的阅读各个部分。

手册说明

```
$ cbmc\cbmc.exe --version
```

上例中的美元符($)代表命令行提示符，是用于提示用户需要在命令行中输入除美元符外的其他内容(cbmc/cbmc.exe --version)。

英文缩写

CPU – Central Processing Unit – 中央处理器

参考资料

CBMC 项目主页：http://www.cprover.org/cbmc/

Visual Studio 2010 Express:

http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express

Visual Studio Community:

http://www.visualstudio.com/en-us/products/visual-studio-community-vs

2 简介

CBMC 支持 C89 标准、C99 标准和 C11 的大部分标准，以及 gcc 和 Visual Studio 绝大多数的扩展功能。CBMC 可以检查数组边界越界问题、缓存溢出问题、指针的安全问题、异常处理和用户指定的断言等。

3 运行环境

3.1 硬件要求

CBMC 对内存和 CPU 的要求不高，绝大多数计算机都可以运行。

3.2 支持的操作系统

CBMC 支持 Linux 和 Windows，Linux 的 Debian 和 Fedora 发行版本有编译好的程序提供下载，也提供源代码供其它平台编译安装。为 Windows 提供编译好的 exe 可执行文件，需要在 Visual Studio Command Prompt(Visual Studio 命令提示)下运行。

4 安装

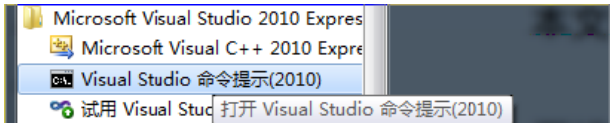
4.1 Windows

注意:Visual Studio 的 cl.exe 依赖许多的环境变量来辨认目标架构和包含头文件的目录。用户必须通过 Visual Studio Command Prompt(Visual Studio 命令提示)来运行 CBMC。

Windows 版本的 CBMC 需要预处理程序 cl.exe 的支持，cl.exe 是 Microsoft Visual Studio 的一部分，可以安装免费的 Visual Studio 2010 Express 或者免费的 Visual Studio Community 来获得。

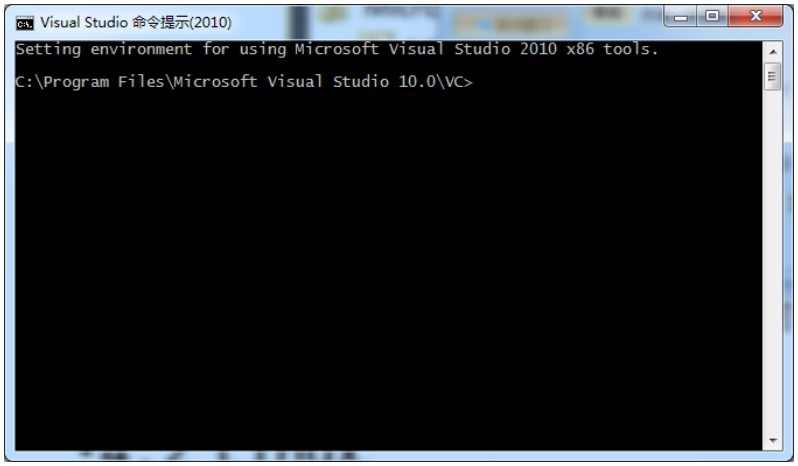
本文以 Visual Studio 2010 Express 为例，介绍如何安装 CBMC 工具。假设用户已经安装好 Visual Studio 2010 Express(具体方法请参考微软的 VS 安装指导手册)。

首先，从开始菜单中找到 Visual Studio 2010 Express 的 Visual Studio 命令提示(2010)，点击运行。如图一所示。



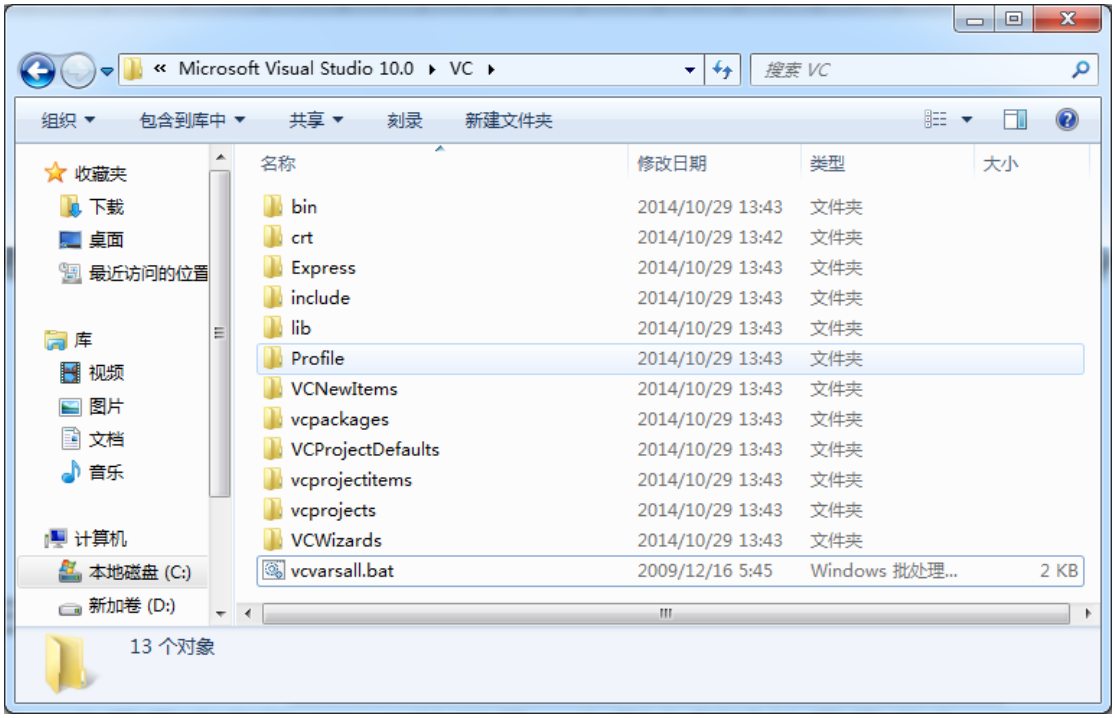
图一 Visual Studio 命令提示(2010)

打开 Visual Studio 命令提示(2010)，可以看到提示行处有当前的路径，图二例中的路径是”C:\Program Files\Microsoft Visual Studio 10.0\VC\”。



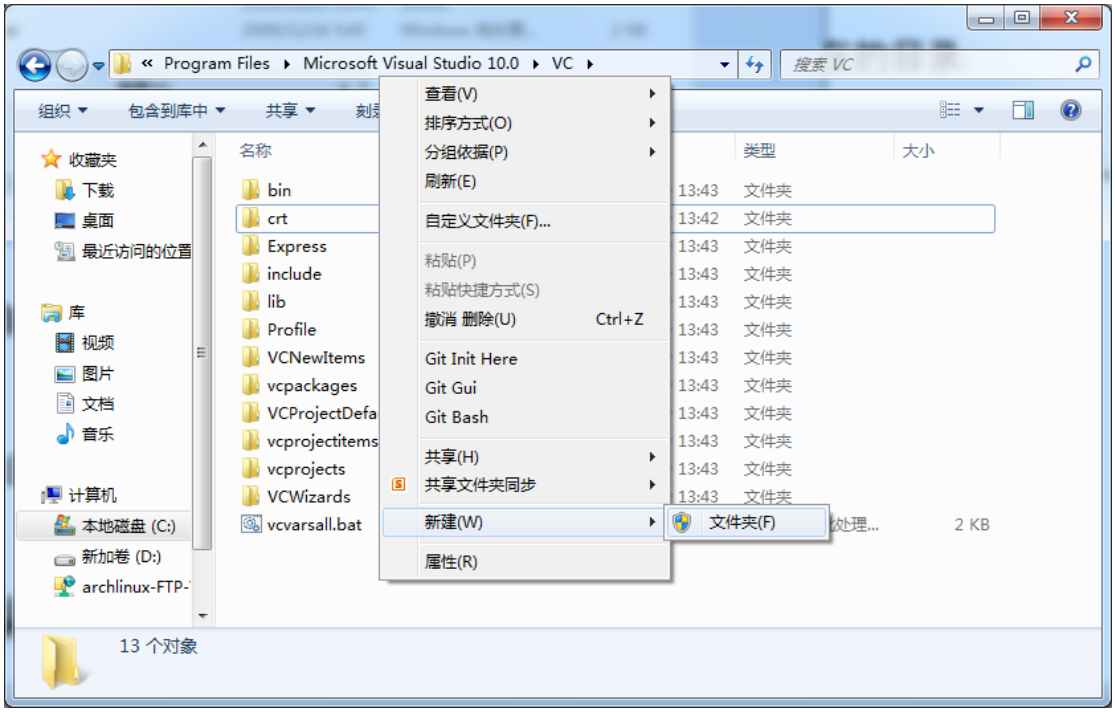
图二 运行 Visual Studio 命令提示(2010)

在资源管理器的地址栏中输入该路径，打开 Visual Studio 命令提示(2010)所在的目录。如图三所示。



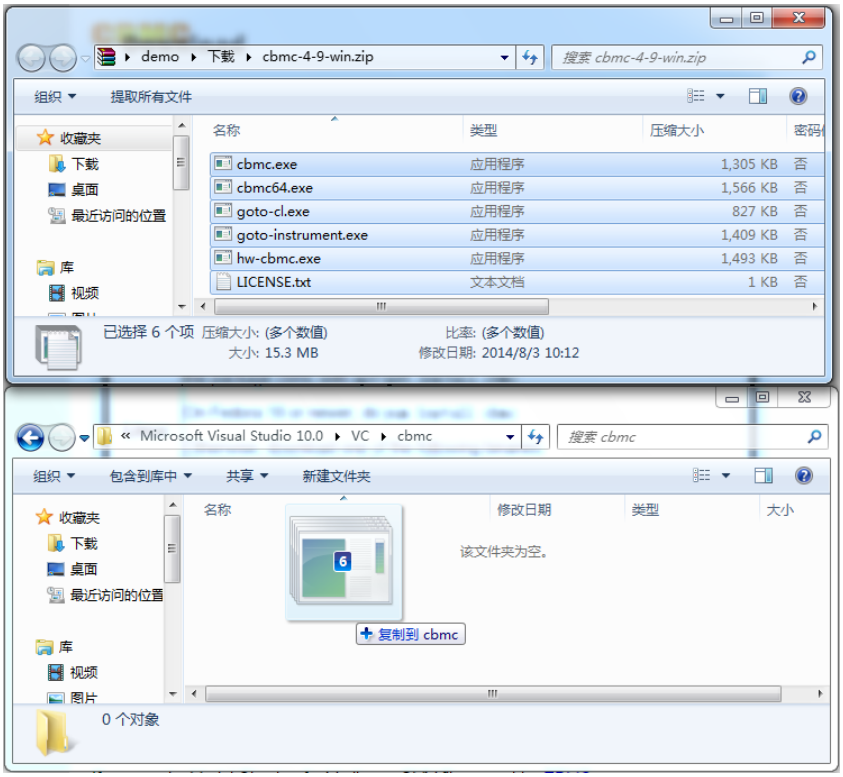
图三 打开 Visual Studio 命令提示(2010)所在的目录

新建一个目录来存放 CBMC 的工具。本例中新建的目录名为 cbmc。如图四所示。



图四 新建文件夹 cbmc

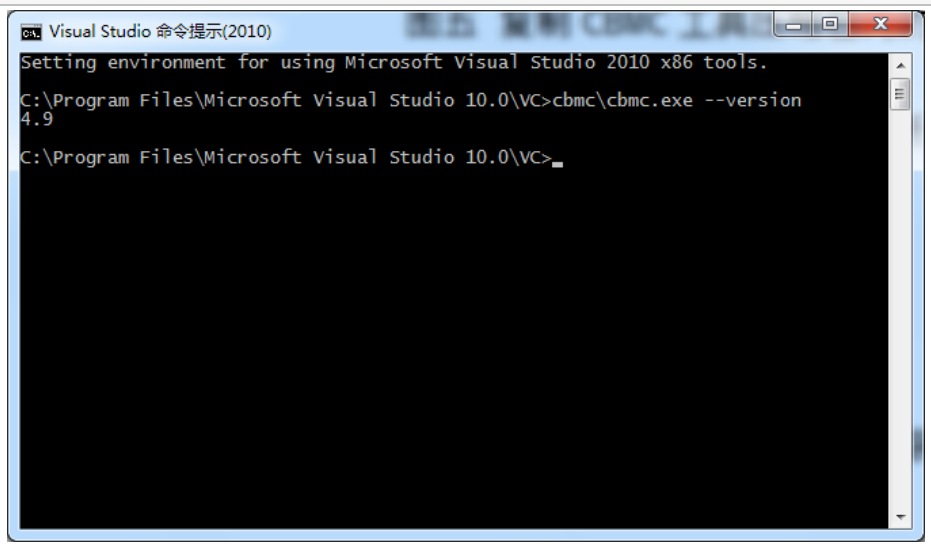
下载 CBMC 工具压缩包(cbmc-4.9-win.zip)，把工具内的程序解压到刚刚新建好的文件夹内。如图五所示。



图五 复制 CBMC 工具压缩包内的工具至文件夹 cbmc 内

运行 CBMC 程序（输入以下除美元符$外的内容），确认工具可运行。如图六所示。

```
$ cbmc\cbmc.exe --version
```



图六 确认 CBMC 安装成功

4.2 Linux

Linux 的部分发行版有提供官方软件库的安装方法。而发行版官方软件库提供的软件一般都落后于最新版，比如 Ubuntu 的 CBMC 的版本是 4.5。

4.2.1 从发行版的官方软件库中安装

Debian Wheezy 或更新的版本：

```
$ sudo apt-get install cbmc
```

Fedora 18 或更新的版本：

```
$ sudo yum install cbmc
```

安装完成后，请运行如下命令确认 CBMC 已完成安装：

```
$ type cbmc             // 确认系统中能够找到 cbmc 命令所处位置
cbmc is /usr/bin/cbmc
$ cbmc --version        // 查看 cbmc 程序版本
4.5
```

4.2.2 下载编译好的程序

如果发行版官方软件库没有提供编译好的 CBMC，可以从 CBMC 官网下载编译好的可执行文件。

32 位的软件包：

```
$ wget http://www.cprover.org/cbmc/download/cbmc-4-9-linux-32.tgz
$ tar xfz cbmc-4-9-linux-32.tgz
```

64 位的软件包：

```
$ wget http://www.cprover.org/cbmc/download/cbmc-4-9-linux-64.tgz
$ tar xfz cbmc-4-9-linux-64.tgz
```

解压后，会在当前目录找到五个文件，分别是 cbmc hw-cbmc goto-cc

goto-instrument 和 LICENSE 文件。其中的 cbmc 便是最主要的程序。运行如下的命令，如果输出是一个版本号，则说明 cbmc 程序可以运行。

```
$ ./cbmc --version
4.9
```

4.2.3 编译安装

本文中以 Ubuntu 为操作系统，gcc 为编译器（假设用户知道如何在自己的平台上安装 C/C++编译器），介绍编译安装 CBMC 的步骤。其他环境下的编译方法，请参考源代码目录下的 COMPILING 文件。

4.2.3.1 准备工作

首先，需要准备好 C/C++编译器，Flex 和 Bison，GNU make（版本号大于 3.81）。

在类 Debian 的发行版下，可以通过以下命令安装：

```
$ apt-get install g++ gcc flex bison make subversion libz-dev libwww-perl patch
```

在 Red Hat 系发行版下，可以通过以下命令安装：

```
$ yum install gcc gcc-c++ flex bison perl-libwww-perl patch
```

警告：g++ 4.5.x 有已知问题，应当使用更新的版本。

4.2.3.2 获得 CBMC 代码

使用 svn 工具下载代码：

```
$ svn co http://www.cprover.org/svn/cbmc/releases/cbmc-4.9
```

svn 签出所有代码后，所有的代码都在目录 cbmc-4.9 内。

```
$ cd cbmc-4.9
```

4.2.3.3 获得 MiniSat2

代码主要存放在 src 目录下，在编译 CMBC 之前，我们需要先获得 MiniSat2。

```
$ cd src
$ make minisat2-download
Saving to 'minisat2_2.2.0.orig.tar.gz'...
42.8 KB received in 1 seconds (42.8 KB/sec)
patching file core/Solver.cc
patching file mtl/IntTypes.h
patching file mtl/Vec.h
patching file simp/SimpSolver.cc
patching file utils/Options.h
patching file utils/ParseUtils.h
```

4.2.3.4 编译 cbmc

MiniSat2 下载完成后，在 src 目录下执行 make 命令，可以开始 CBMC 的编译。如下所示：

```
$ make
```

编译过程的输出内容过多，这里就不贴出来了。如果编译过程中没有出错，会在当前目录(cbmc-4.9/src/)下的 cbmc 目录下生成 cbmc 程序。

```
$ cbmc/cbmc --version
4.9
```

至此，CBMC 的编译工作就完成了。

5 常用操作

5.1 第一个例子

例如，我们有以下的一份文件(file1.c):

```
$ cat file1.c
int puts(const char *s) { }
int main(int argc, char **argv) {
    int i;
    if (argc>=1)
        puts(argv[2]);
}
```

显而易见，这份程序中有明显的错误。因为 argv 数组可能只有一个元素，那么对argv[2]的取值就会越界。

我们使用 cbmc 检查 file1.c 文件，方法如下:

```
$ cbmc file1.c --show-properties --bounds-check --pointer-check
```

（输出内容过长，可参考 7.1.2 。）

这里的后两个参数--bounds-check 和--pointer-check 告诉 CBMC 查找关于指针和数组越界的错误。CBMC 会列出它检查的属性。

我们可以注意到，CBMC 输出了一个标有”array argv upper bound”的区块，一起输出的还有产生错误的数组访问的位置。正如例子中看到的，我们需要为 CBMC 指定检查的内容。

需要注意的是，每个自动生成的区块并不都对应着 bug，这些仅仅是可能有错误。这些区块是否对应 bug 需要通过进一步的分析来确定。

一种分析的方法是符号模拟(symbolic simulation)。这个过程会把程序转化成公式，而公式和每个区块有联系。如下所示:

```
$ cbmc file1.c --show-vcc --bounds-check --pointer-check
```

（输出内容过长，可参考 7.1.3 。）

通过这种方法，CBMC 执行符号模拟，并输出验证条件。一个验证条件需要被证明是可靠的，这需要通过决策过程(decision procedure，更详细内容参看维基百科)。这样也就能保证每个区块所阐述的属性的正确性。执行决策过程的方法如下：

```
$ cbmc file1.c --bounds-check --pointer-check
```

（输出内容过长，可参考 7.1.4 。）

更多的原理请参考《Decision Procedures》一书（参看 6 更多资料）。

5.2 验证模块

在第一个例子里，我们检查的是一个包含 main 函数的源代码。然而，CBMC 的一个目标是针对嵌入式程序，这部分程序通常会有不同的程序进入点。更进一步讲，CBMC 在验证程序的模块时，也十分出色。例如我们有以下的一段代码(file2.c)：

```
$ cat file2.c
int array[10];
int sum() {
    unsigned i, sum;
    sum = 0;
    for(i=0; i<10; i++)
        sum += array[i];
}
```

设置 sum 函数为程序进入点，使用如下的命令：

```
$ cbmc file2.c --function sum
```

（输出内容过长，可参考 7.2.2 。）

5.3 展开循环

你可能注意到了，CBMC 会展开程序中的 for 循环。CBMC 需要检查模型越界问题，为了保证所有的 bug 都能被找到，所有的循环在运行时间上都应该有一个上界，这样 bug 才能够被找到。我们来看下面这个例子(binsearch.c):

```
$ cat binsearch.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 // a[middle] = x !
            return middle;
    }

    return -1;
}
```

如果我们用 CBMC 来验证这个程序，会注意到循环的展开过程并不会停止。内置的简化器没能决定该把这个循环展开多少次,我们需要在命令行的参数中设置好循环展开次数的上限。如下所示:

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

（输出内容过长，可参考 7.3.2 。）

5.4 无限循环

CBMC 也可以被用在那些有无限循环的程序上。在这种情况下，CBMC 只能被用来找错误，但 CBMC 并不会试图找到所有的 bug。请看下面的一个例子(lock-example.c):

```
$ cat lock-example.c
_Bool nondet_bool();
_Bool LOCK = 0;

_Bool lock() {
    if(nondet_bool()) {
        assert(!LOCK);
        LOCK=1;
        return 1;
    }

    return 0;
}

void unlock() {
    assert(LOCK);
    LOCK=0;
}

int main() {
    unsigned got_lock = 0;
    int times;

    while(times > 0) {
        if(lock()) {
            got_lock++;
            /* critical section */
        }

        if(got_lock!=0)
            unlock();

        got_lock--;
        times--;
    }
}
```

在 main 函数中的 while 循环在运行时间上没有上限，因此，我们需要设置 CBMC 展开循环次数的上限。有两种方法：

1 命令行参数--unwind 可以设置循环展开次数的上限。例如：

```
$ cbmc lock-example.c --unwind 3 --bounds-check //输出内容请参看 7.4.2
```

2 命令行参数--depth 可以设置程序执行的深度。例如：

```
$ cbmc lock-example.c --depth 3 --bounds-check //输出内容请参看 7.4.3
```

CBMC 可以判断--unwind 的参数是否足够大（足以覆盖程序的所有部分）。如果--unwind 设置的参数过小，CBMC 会检测到没有足够展开循环，并且会报出一个展开断言错误(unwinding assertion violation)，并终止程序。如果想要禁用这个功能，在运行 CBMC 的同时，添加如下的命令行参数：

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

例如：

```
$ cbmc lock-example.c --unwind 3 --bounds-check --no-unwinding-assertions
```

（输出内容请参看 7.4.4）

我们回过头来看本节的例子。如果我们只循环一次，就不会找到 bug。但如果我们展开循环两次，那么 CBMC 会检测到展开断言错误(unwinding assertion violation)。如果我们使用不检查展开断言错误(--no-unwinding-assertions)，或者使用--depth 参数指定程序执行的深度，CBMC 就不能证明程序的正确性，但这对于寻找程序中的 bug 还是很有帮助。

CBMC 程序有提供更多关于循环展开的命令行参数，可以参看 CBMC 官方用户手册（英文）的理解循环展开一节（链接附在 6 更多资料内）。

6 更多资料

CPROVER 的 Visual Studio 插件：

http://www.cprover.org/visual-studio/

CBMC Eclipse 插件的安装指导：

http://www.cprover.org/eclipse-plugin/

CBMC 的 Slide：

http://www.cprover.org/cbmc/doc/cbmc-slides.pdf

《Decision Procedures》：

http://www.decision-procedures.org/

CBMC 官方用户手册（英文）的理解循环展开：

http://www.cprover.org/cprover-manual/cbmc-loops.shtml

7 附录：例子

7.1 例子一(file1.c)

7.1.1 检验的程序

```
$ cat file1.c
int puts(const char *s) { }
int main(int argc, char **argv) {
    int i;
    if (argc>=1)
        puts(argv[2]);
}
```

7.1.2 第一次运行

```
$ cbmc file1.c --show-properties --bounds-check --pointer-check
CBMC version 4.9 32-bit linux
file file1.c: Parsing
Converting
Type-checking file1
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Property main.1:
  file file1.c line 5 function main
  dereference failure: pointer NULL
  !(2 + argv == ((char **)NULL))
  in "argv[2]"
Property main.2:
  file file1.c line 5 function main
  dereference failure: pointer invalid
  !INVALID-POINTER(argv)
  in "argv[2]"
Property main.3:
  file file1.c line 5 function main
  dereference failure: deallocated dynamic object
```

```
  !(POINTER_OBJECT(argv) == POINTER_OBJECT(__CPROVER_deallocated))
  in "argv[2]"
Property main.4:
  file file1.c line 5 function main
  dereference failure: dead object
  !(POINTER_OBJECT(argv) == POINTER_OBJECT(__CPROVER_dead_object))
  in "argv[2]"
Property main.5:
  file file1.c line 5 function main
  dereference failure: dynamic object bounds
  !(8 + POINTER_OFFSET(argv) < 0) && __CPROVER_malloc_size >= 12u + (unsigned
int)POINTER_OFFSET(argv) || !(POINTER_OBJECT(argv) ==
POINTER_OBJECT(__CPROVER_malloc_object))
  in "argv[2]"
Property main.6:
  file file1.c line 5 function main
  dereference failure: object bounds
  !(8 + POINTER_OFFSET(argv) < 0) && OBJECT_SIZE(argv) >= 12 + POINTER_OFFSET(argv)
|| DYNAMIC_OBJECT(argv)
  in "argv[2]"
```

7.1.3 第二次运行

```
$ cbmc file1.c --show-vcc --bounds-check --pointer-check
file file1.c: Parsing
Converting
Type-checking file1
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
size of program expression: 39 steps
simple slicing removed 8 assignments
Generated 5 VCC(s), 2 remaining after simplification

VERIFICATION CONDITIONS:

file file1.c line 5 function main
dereference failure: pointer NULL
{-1} __CPROVER_pipe_count#1 == 0u
```

```
{-2} __CPROVER_rounding_mode!0#1 == 0
{-3} __CPROVER_threads_exited#1 == ARRAY_OF(FALSE)
{-4} __CPROVER_next_thread_id#1 == 0ul
{-5} __CPROVER_deallocated#1 == NULL
{-6} __CPROVER_malloc_object#1 == NULL
{-7} __CPROVER_malloc_size#1 == 0u
{-8} __CPROVER_malloc_is_new_array#1 == FALSE
{-9} argc'#0 >= 1
{-10} argc'#0 <= 268435456
{-11} argv'#1 == argv'#0 WITH [argc'#0:=((char *)NULL)]
{-12} argc!0@1#1 == argc'#0
{-13} argv!0@1#1 == argv'
{-14} \guard#1 == argc!0@1#1 >= 1
|--------------------------
{1} \guard#1 ==> !(argv' + 2 == ((char **)NULL))

file file1.c line 5 function main
dereference failure: object bounds
{-1} __CPROVER_pipe_count#1 == 0u
{-2} __CPROVER_rounding_mode!0#1 == 0
{-3} __CPROVER_threads_exited#1 == ARRAY_OF(FALSE)
{-4} __CPROVER_next_thread_id#1 == 0ul
{-5} __CPROVER_deallocated#1 == NULL
{-6} __CPROVER_malloc_object#1 == NULL
{-7} __CPROVER_malloc_size#1 == 0u
{-8} __CPROVER_malloc_is_new_array#1 == FALSE
{-9} argc'#0 >= 1
{-10} argc'#0 <= 268435456
{-11} argv'#1 == argv'#0 WITH [argc'#0:=((char *)NULL)]
{-12} argc!0@1#1 == argc'#0
{-13} argv!0@1#1 == argv'
{-14} \guard#1 == argc!0@1#1 >= 1
|--------------------------
{1} \guard#1 ==> (1 + argc'#0) * 4 >= 12
```

7.1.4 第三次运行

```
$ ./cbmc file1.c --bounds-check --pointer-check
CBMC version 4.9 32-bit linux
file file1.c: Parsing
Converting
Type-checking file1
```

```
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
size of program expression: 47 steps
simple slicing removed 8 assignments
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
570 variables, 1551 clauses
SAT checker: negated claim is SATISFIABLE, i.e., does not hold
Runtime decision procedure: 0.013s
Building error trace

Counterexample:

State 3 file <built-in-additions> line 58 thread 0
------------------------------------------------------
  __CPROVER_rounding_mode=0 (00000000000000000000000000000000)

State 4 file <built-in-additions> line 31 thread 0
------------------------------------------------------
  __CPROVER_deallocated=NULL (00000000000000000000000000000000)

State 5 file <built-in-additions> line 32 thread 0
------------------------------------------------------
  __CPROVER_dead_object=NULL (00000000000000000000000000000000)

State 6 file <built-in-additions> line 33 thread 0
------------------------------------------------------
  __CPROVER_malloc_object=NULL (00000000000000000000000000000000)

State 7 file <built-in-additions> line 34 thread 0
------------------------------------------------------
  __CPROVER_malloc_size=0 (00000000000000000000000000000000)

State 8 file <built-in-additions> line 35 thread 0
------------------------------------------------------
  __CPROVER_malloc_is_new_array=FALSE (0)
```

```
State 9 file <built-in-additions> line 36 thread 0
------------------------------------------------------
  __CPROVER_memory_leak=NULL (0000000000000000000000000000000)

State 10 file <built-in-additions> line 79 thread 0
------------------------------------------------------
  __CPROVER_pipe_count=0 (0000000000000000000000000000000)

State 11 file <built-in-additions> line 22 thread 0
------------------------------------------------------
  __CPROVER_thread_id=0 (0000000000000000000000000000000)

State 12 file <built-in-additions> line 23 thread 0
------------------------------------------------------
  __CPROVER_threads_exited=__CPROVER_threads_exited#1 (?)

State 13 file <built-in-additions> line 24 thread 0
------------------------------------------------------
  __CPROVER_next_thread_id=0 (0000000000000000000000000000000)

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

State 20 file file1.c line 2 thread 0
------------------------------------------------------
  argc=1 (0000000000000000000000000000001)

State 21 file file1.c line 2 thread 0
------------------------------------------------------
  argv=argv' (0000001000000000000000000000000)

State 22 file file1.c line 3 function main thread 0
------------------------------------------------------
  i=0 (0000000000000000000000000000000)

Violated property:
  file file1.c line 5 function main
  dereference failure: object bounds
  !(8 + POINTER_OFFSET(argv) < 0) && OBJECT_SIZE(argv) >= 12 + POINTER_OFFSET(argv)
|| DYNAMIC_OBJECT(argv)
VERIFICATION FAILED
```

7.2 例子二(file2.c)

7.2.1 检验的程序

```
$ cat file2.c
int array[10];
int sum() {
    unsigned i, sum;
    sum = 0;
    for(i=0; i<10; i++)
        sum += array[i];
}
```

7.2.2 运行的结果

```
$ ./cbmc file2.c --function sum
CBMC version 4.9 32-bit linux
file file2.c: Parsing
Converting
Type-checking file2
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
Unwinding loop c::sum.0 iteration 1 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 2 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 3 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 4 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 5 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 6 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 7 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 8 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 9 file file2.c line 5 function sum thread 0
Unwinding loop c::sum.0 iteration 10 file file2.c line 5 function sum thread 0
size of program expression: 65 steps
simple slicing removed 0 assignments
Generated 0 VCC(s), 0 remaining after simplification
VERIFICATION SUCCESSFUL
```

7.3 例子三(binsearch.c)

7.3.1 检验的程序

```
$ cat binsearch.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 // a[middle]=x !
            return middle;
    }

    return -1;
}
```

7.3.2 运行的结果

```
$ cbmc binsearch.c --function binsearch --unwind 6 --bounds-check
CBMC version 4.9 32-bit linux
file binsearch.c: Parsing
Converting
Type-checking binsearch
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
Unwinding loop c::binsearch.0 iteration 1 (6 max) file binsearch.c line 5 function
binsearch thread 0
Unwinding loop c::binsearch.0 iteration 2 (6 max) file binsearch.c line 5 function
binsearch thread 0
```

```
Unwinding loop c::binsearch.0 iteration 3 (6 max) file binsearch.c line 5 function
binsearch thread 0
Unwinding loop c::binsearch.0 iteration 4 (6 max) file binsearch.c line 5 function
binsearch thread 0
Unwinding loop c::binsearch.0 iteration 5 (6 max) file binsearch.c line 5 function
binsearch thread 0
Not unwinding loop c::binsearch.0 iteration 6 (6 max) file binsearch.c line 5
function binsearch thread 0
size of program expression: 209 steps
simple slicing removed 45 assignments
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
8033 variables, 32805 clauses
SAT checker: negated claim is UNSATISFIABLE, i.e., holds
Runtime decision procedure: 0.124s
VERIFICATION SUCCESSFUL
```

7.4 例子四(lock-example.c)

7.4.1 检验的程序

```
$ cat lock-example.c
_Bool nondet_bool();
_Bool LOCK = 0;

_Bool lock() {
    if(nondet_bool()) {
        assert(!LOCK);
            LOCK=1;
            return 1;
    }

    return 0;
}

void unlock() {
    assert(LOCK);
    LOCK=0;
}

int main() {
    unsigned got_lock = 0;
    int times;

    while(times > 0) {
        if(lock()) {
            got_lock++;
            /* critical section */
        }

        if(got_lock!=0)
            unlock();

        got_lock--;
        times--;
    }
}
```

7.4.2 第一次运行

```
$ cbmc lock-example.c --unwind 3 --bounds-check
CBMC version 4.9 32-bit linux
file lock-example.c: Parsing
Converting
Type-checking lock-example
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
Unwinding loop c::main.0 iteration 1 (3 max) file lock-example.c line 23 function
main thread 0
Unwinding loop c::main.0 iteration 2 (3 max) file lock-example.c line 23 function
main thread 0
Not unwinding loop c::main.0 iteration 3 (3 max) file lock-example.c line 23 function
main thread 0
size of program expression: 162 steps
simple slicing removed 15 assignments
Generated 7 VCC(s), 6 remaining after simplification
Passing problem to propositional reduction
Running propositional reduction
Post-processing
Solving with MiniSAT 2.2.0 with simplifier
773 variables, 2145 clauses
SAT checker: negated claim is SATISFIABLE, i.e., does not hold
Runtime decision procedure: 0.016s
Building error trace

Counterexample:

State 3 file <built-in-additions> line 58 thread 0
------------------------------------------------------
  __CPROVER_rounding_mode=0 (00000000000000000000000000000000)

State 4 file lock-example.c line 2 thread 0
------------------------------------------------------
  LOCK=0 (00000000)

State 5 file <built-in-additions> line 31 thread 0
------------------------------------------------------
```

```
  __CPROVER_deallocated=NULL (00000000000000000000000000000000)

State 6 file <built-in-additions> line 32 thread 0
----------------------------------------------------
  __CPROVER_dead_object=NULL (00000000000000000000000000000000)

State 7 file <built-in-additions> line 33 thread 0
----------------------------------------------------
  __CPROVER_malloc_object=NULL (00000000000000000000000000000000)

State 8 file <built-in-additions> line 34 thread 0
----------------------------------------------------
  __CPROVER_malloc_size=0 (00000000000000000000000000000000)

State 9 file <built-in-additions> line 35 thread 0
----------------------------------------------------
  __CPROVER_malloc_is_new_array=FALSE (0)

State 10 file <built-in-additions> line 36 thread 0
----------------------------------------------------
  __CPROVER_memory_leak=NULL (00000000000000000000000000000000)

State 11 file <built-in-additions> line 79 thread 0
----------------------------------------------------
  __CPROVER_pipe_count=0 (00000000000000000000000000000000)

State 12 file <built-in-additions> line 22 thread 0
----------------------------------------------------
  __CPROVER_thread_id=0 (00000000000000000000000000000000)

State 13 file <built-in-additions> line 23 thread 0
----------------------------------------------------
  __CPROVER_threads_exited=__CPROVER_threads_exited#1 (?)

State 14 file <built-in-additions> line 24 thread 0
----------------------------------------------------
  __CPROVER_next_thread_id=0 (00000000000000000000000000000000)

State 18 file lock-example.c line 20 function main thread 0
----------------------------------------------------
  got_lock=0 (00000000000000000000000000000000)

State 19 file lock-example.c line 20 function main thread 0
----------------------------------------------------
```

```
  got_lock=0 (00000000000000000000000000000000)

State 20 file lock-example.c line 21 function main thread 0
----------------------------------------------------
  times=1073742081 (01000000000000000000000100000001)

State 22 file lock-example.c line 24 function main thread 0
----------------------------------------------------
  return_value_lock$1=0 (00000000)

State 25 file lock-example.c line 5 function lock thread 0
----------------------------------------------------
  return_value_nondet_bool$1=0 (00000000)

State 26 file lock-example.c line 5 function lock thread 0
----------------------------------------------------
  return_value_nondet_bool$1=0 (00000000)

State 29 file lock-example.c line 11 function lock thread 0
----------------------------------------------------
  return_value_lock$1=0 (00000000)

State 33 file lock-example.c line 32 function main thread 0
----------------------------------------------------
  got_lock=4294967295 (11111111111111111111111111111111)

State 34 file lock-example.c line 33 function main thread 0
----------------------------------------------------
  times=1073742080 (01000000000000000000000100000000)

State 37 file lock-example.c line 24 function main thread 0
----------------------------------------------------
  return_value_lock$1=0 (00000000)

State 40 file lock-example.c line 5 function lock thread 0
----------------------------------------------------
  return_value_nondet_bool$1=0 (00000000)

State 41 file lock-example.c line 5 function lock thread 0
----------------------------------------------------
  return_value_nondet_bool$1=0 (00000000)

State 44 file lock-example.c line 11 function lock thread 0
----------------------------------------------------
```

```
  return_value_lock$1=0 (00000000)

Violated property:
  file lock-example.c line 15 function unlock
  assertion LOCK != FALSE
  LOCK != FALSE

VERIFICATION FAILED
```

7.4.3 第二次运行

```
$ cbmc lock-example.c --depth 3 --bounds-check
CBMC version 4.9 32-bit linux
file lock-example.c: Parsing
Converting
Type-checking lock-example
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
size of program expression: 9 steps
simple slicing removed 0 assignments
Generated 0 VCC(s), 0 remaining after simplification
VERIFICATION SUCCESSFUL
```

7.4.4 第三次运行

```
$ cbmc lock-example.c --unwind 3 --bounds-check --no-unwinding-assertions
CBMC version 4.9 32-bit linux
file lock-example.c: Parsing
Converting
Type-checking lock-example
Generating GOTO Program
Adding CPROVER library
Function Pointer Removal
Partial Inlining
Generic Property Instrumentation
Starting Bounded Model Checking
```

```
Unwinding loop c::main.0 iteration 1 (3 max) file lock-example.c line 23 function
main thread 0
Unwinding loop c::main.0 iteration 2 (3 max) file lock-example.c line 23 function
main thread 0
Not unwinding loop c::main.0 iteration 3 (3 max) file lock-example.c line 23 function
main thread 0
size of program expression: 162 steps
simple slicing removed 23 assignments
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
648 variables, 1708 clauses
SAT checker: negated claim is SATISFIABLE, i.e., does not hold
Runtime decision procedure: 0.013s
Building error trace

Counterexample:

State 3 file <built-in-additions> line 58 thread 0
------------------------------------------------
  __CPROVER_rounding_mode=0 (00000000000000000000000000000000)

State 4 file lock-example.c line 2 thread 0
------------------------------------------------
  LOCK=0 (00000000)

State 5 file <built-in-additions> line 31 thread 0
------------------------------------------------
  __CPROVER_deallocated=NULL (00000000000000000000000000000000)

State 6 file <built-in-additions> line 32 thread 0
------------------------------------------------
  __CPROVER_dead_object=NULL (00000000000000000000000000000000)

State 7 file <built-in-additions> line 33 thread 0
------------------------------------------------
  __CPROVER_malloc_object=NULL (00000000000000000000000000000000)

State 8 file <built-in-additions> line 34 thread 0
------------------------------------------------
  __CPROVER_malloc_size=0 (00000000000000000000000000000000)
```

```
State 9 file <built-in-additions> line 35 thread 0
----------------------------------------------------
  __CPROVER_malloc_is_new_array=FALSE (0)

State 10 file <built-in-additions> line 36 thread 0
----------------------------------------------------
  __CPROVER_memory_leak=NULL (0000000000000000000000000000000)

State 11 file <built-in-additions> line 79 thread 0
----------------------------------------------------
  __CPROVER_pipe_count=0 (00000000000000000000000000000000)

State 12 file <built-in-additions> line 22 thread 0
----------------------------------------------------
  __CPROVER_thread_id=0 (00000000000000000000000000000000)

State 13 file <built-in-additions> line 23 thread 0
----------------------------------------------------
  __CPROVER_threads_exited=__CPROVER_threads_exited#1 (?)

State 14 file <built-in-additions> line 24 thread 0
----------------------------------------------------
  __CPROVER_next_thread_id=0 (00000000000000000000000000000000)

State 18 file lock-example.c line 20 function main thread 0
----------------------------------------------------
  got_lock=0 (00000000000000000000000000000000)

State 19 file lock-example.c line 20 function main thread 0
----------------------------------------------------
  got_lock=0 (00000000000000000000000000000000)

State 20 file lock-example.c line 21 function main thread 0
----------------------------------------------------
  times=1342177281 (01010000000000000000000000000001)

State 22 file lock-example.c line 24 function main thread 0
----------------------------------------------------
  return_value_lock$1=0 (00000000)

State 25 file lock-example.c line 5 function lock thread 0
----------------------------------------------------
  return_value_nondet_bool$1=0 (00000000)
```

```
State 26 file lock-example.c line 5 function lock thread 0
------------------------------------------------------
  return_value_nondet_bool$1=0 (00000000)

State 29 file lock-example.c line 11 function lock thread 0
------------------------------------------------------
  return_value_lock$1=0 (00000000)

State 33 file lock-example.c line 32 function main thread 0
------------------------------------------------------
  got_lock=4294967295 (11111111111111111111111111111111)

State 34 file lock-example.c line 33 function main thread 0
------------------------------------------------------
  times=1342177280 (01010000000000000000000000000000)

State 37 file lock-example.c line 24 function main thread 0
------------------------------------------------------
  return_value_lock$1=0 (00000000)

State 40 file lock-example.c line 5 function lock thread 0
------------------------------------------------------
  return_value_nondet_bool$1=0 (00000000)

State 41 file lock-example.c line 5 function lock thread 0
------------------------------------------------------
  return_value_nondet_bool$1=0 (00000000)

State 44 file lock-example.c line 11 function lock thread 0
------------------------------------------------------
  return_value_lock$1=0 (00000000)

Violated property:
  file lock-example.c line 15 function unlock
  assertion LOCK != FALSE
  LOCK != FALSE

VERIFICATION FAILED
```