Technische Universität München
Institut für Informatik
Prof. Dr. Helmut Veith
Dipl.-Inf.(FH) Andreas Holzer, M.Sc.

SS 2007
Übungsblatt 7
Mittwoch, 20. Juni 2007

Übung zur Vorlesung Model Checking

CBMC

Use CBMC to solve the following exercises. CBMC can be obtained from `http://www.cs.cmu.edu/~modelcheck/cbmc/`. Use the documentation provided on the CBMC website to solve the following exercises.

(a) *Checking properties with CBMC.*

i) What is wrong with this program? Use CBMC for checking where this program fails.

```
#include <stdlib.h>

void test() {
  int size = nondet_int();

  if (size > 0) {
    int* array = (int*)(malloc(size * sizeof(int)));

    int x = size - 1;
    int y = size - 1;

    int index = (x + y)/2;

    int c = array[index];
  }
}
```

After observing the error in this rather artificial program you may be interested in more common algorithms where it occurs: `http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html`.

ii) *Using* `__CPROVER_assume`. We have already seen that CBMC can check certain properties implicitly. We now want to demonstrate how software components can be verified separately using `__CPROVER_assume`. CBMC reports no errors on the following code. Lets assume `doWork` and `foo` are very big and we want to verify them separately. Place both functions in different files and verify them. What problems occur? How can you solve it?

```
#include <stdlib.h>

void doWork(int size, int index);

void foo() {
  int size = nondet_int();
  int index = nondet_int();

  if (size > 0 && 0 <= index && index < size) {
    doWork(size, index);
  }
}

void doWork(int size, int index) {
  int *array = (int*)(malloc(size * sizeof(int)));

  int value = array[index];

  free(array);
}
```

iii) *Problems that may occur when using* `__CPROVER_assume`. Why does CBMC recognize the following program as correct? What is the underlying problem? How can you check whether this problem occurs when you use `__CPROVER_assume`?

```
#include <stdlib.h>

void foo() {
  __CPROVER_assume(0);
  int *array = (int*)(malloc(10 * sizeof(int)));

  int c = array[-1];

  free(array);
}
```

iv) In this program an allocated list should be traversed. There are errors inside the code. Detect them with the help of CBMC and correct the program. Furthermore, show that the number of calls to the function `traverse` is related to the length of the list by stating some assertion that is then checked by CBMC. *Hint:* Consider issues about loop unwinding and maybe you want to use `assert`.

```
#include <stdlib.h>

struct node {
  struct node* succ;
};

void traverse(struct node* n) {
  traverse(n->succ);
```

```
}

int main(int argc, char **argv) {
  struct node *list = (void*)0;
  struct node *new_node;
  int i;
  int size = 10;

  for (i = 0; i < size; i++) {
    new_node = malloc(sizeof(struct node));
    new_node->succ = list;

    list = new_node;
  }

  traverse(list);

  return 0;
}
```

(b) *Test generation using CBMC.* A software model checker can not only be used for verifying software but also for generating test cases for this software. This exercise shows you how CBMC can be used for this purpose.

i) Use CBMC to generate test cases that reach the `fprintf` in the following program.

```
#include <stdio.h>

void foo(int x, int y);

int main(int argc, char **argv) {
  if (argc >= 3) {
    int x = atoi(argv[1]);
    int y = atoi(argv[2]);

    foo(x, y);
  }

  return 0;
}

void foo(int x, int y) {
  if (x == 55) {
    if (y >= 10) {
      fprintf(stdout, "(%d, %d)\n", x, y);
    }
  }
}
```

ii) With CBMC we have a tool for software verification, so why is testing still useful? What other things can you do with our generated test cases? How can we automize the test case generation? What problems occur with our method to generate test cases?

Caution: On Windows systems there may be a bug in stdio.h that is delivered with CBMC. In the case that you get an error then you have to change the line `#define printf( format, args... ) __CPROVER_printf( format, ## args)` to `#define printf( format, args ) __CPROVER_printf( format, ## args)`.