C Bounded Model Checker

- Targeting arbitrary ANSI-C programs
  - Bit vector operators (>>, <<, |, &)
  - Array
  - Pointer arithmetic
  - Dynamic memory allocation
  - Floating #
- Can check
  - Array bound checks (i.e., buffer overflow)
  - Division by 0
  - Pointer checks (i.e., NULL pointer dereference)
  - Arithmetic overflow/underflow
  - User defined assert(cond)
- Handles function calls using inlining
- Unwinds the loops a fixed number of times

Modeling with CBMC

- Models an environment (i.e., various scenarios) using non-determinism
  1. By using undefined functions
  2. By using uninitialized local variables
  3. By using function parameters
  4. By explicitly using `__CPROVER_assume()`

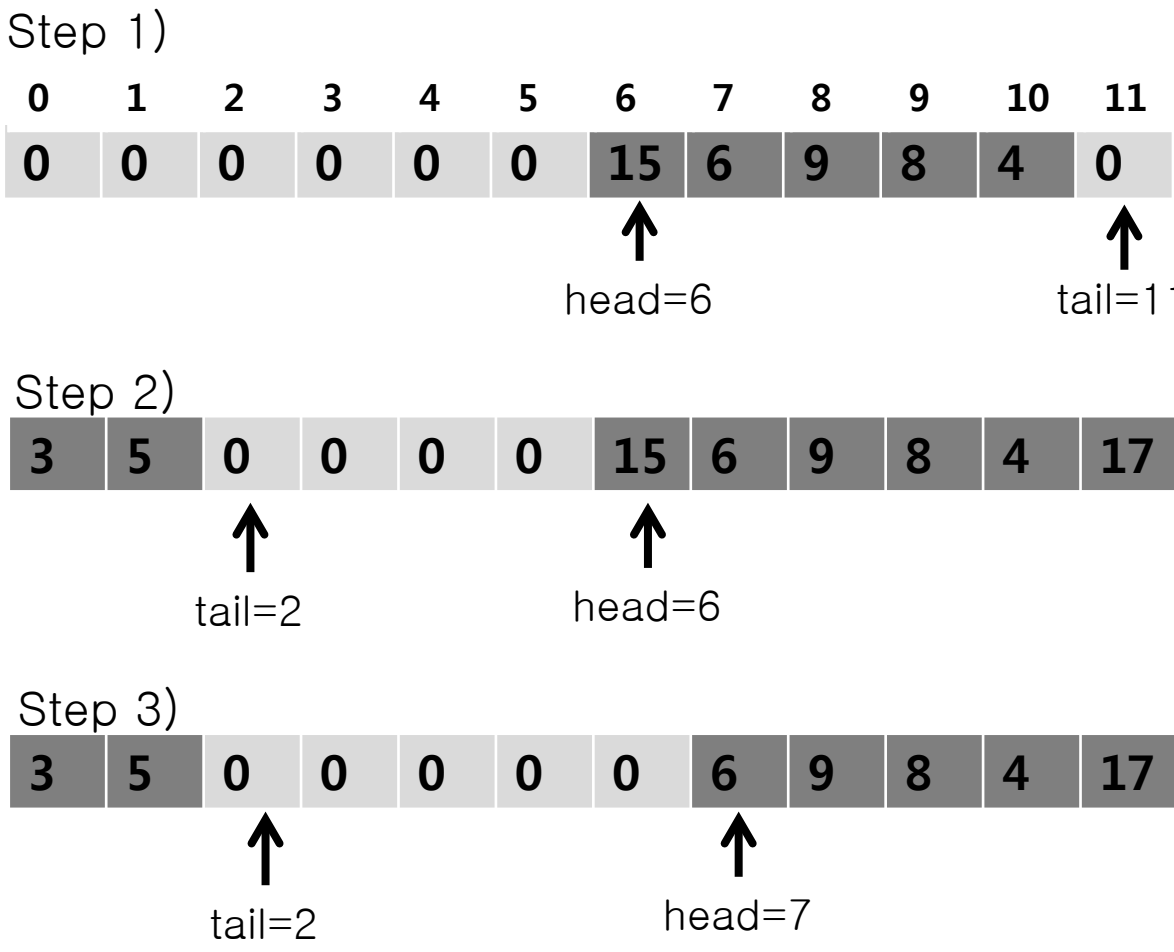
```
foo(int x) {
    __CPROVER_assume
   (0<x && x<10);
    x++;
    assert (x*x <= 100);
}
```

```
bar() {
    int y=0;
    __CPROVER_assume
    ( y > 10);
    assert(0);
}
```

```
int x = nondet();
bar() {
    int y;
    __CPROVER_assume
   (0<x && 0<y);
    if(x < 0 && y < 0)
        assert(0);
}
```

Ex1. Circular Queue APIs

```
#include<stdio.h>
#define SIZE 12
#define EMPTY 0

unsigned int q[SIZE],head,tail;

void enqueue(unsigned int x)
{
    q[tail]=x;
    tail=(++tail)%SIZE;
}

unsigned int dequeue() {
    unsigned int ret;
    ret = q[head];
    q[head]=0;
    head= (++head)%SIZE;
    return ret;}
```



```c
#include<stdio.h>
#define SIZE 12
#define EMPTY 0

unsigned int q[SIZE],head,tail;

void enqueue(unsigned int x)
{
    q[tail]=x;
    tail=(++tail)%SIZE;
}

unsigned int dequeue() {
    unsigned int ret;
    ret = q[head];
    q[head]=0;
    head= (++head)%SIZE;
    return ret;
}
```

```c
// Initial random queue setting following the script
void environment_setup() {
    int i;
    for(i=0;i<SIZE;i++) { q[i]=EMPTY;}

    head=non_det();
    __CPROVER_assume(0<= head && head < SIZE);

    tail=non_det();
    __CPROVER_assume(0<= tail && tail < SIZE);

    if( head < tail)
        for(i=head; i < tail; i++) {
            q[i]=non_det();
            __CPROVER_assume(0< q[i]);
        }
    else if(head > tail) {
        for(i=0; i < tail; i++) {
            q[i]=non_det();
            __CPROVER_assume(0< q[i]);
        }
        for(i=head; i < SIZE; i++) {
            q[i]=non_det();
            __CPROVER_assume(0< q[i]);
        }
    } // We assume that q[] is empty if head==tail
}
```

```
void enqueue_verify() {
    unsigned int x, old_head, old_tail;
    unsigned int old_q[SIZE], i;
    __CPROVER_assume(x>0);

    for(i=0; i < SIZE; i++) old_q[i]=q[i];
    old_head=head;
    old_tail=tail;

    enqueue(x);

    assert(q[old_tail]==x);
    assert(tail== ((old_tail +1) % SIZE));
    assert(head==old_head);
    for(i=0; i < old_tail; i++)
            assert(old_q[i]==q[i]);
    for(i=old_tail+1; i < SIZE; i++)
            assert(old_q[i]==q[i]);
}
```

```
void dequeue_verify() {
    unsigned int ret, old_head, old_tail;
    unsigned int old_q[SIZE], i;

    for(i=0; i < SIZE; i++) old_q[i]=q[i];
    old_head=head;
    old_tail=tail;
    __CPROVER_assume(head!=tail);

    ret=dequeue();

    assert(ret==old_q[old_head]);
    assert(q[old_head]== EMPTY);
    assert(head==(old_head+1)%SIZE);
    assert(tail==old_tail);
    for(i=0; i < old_head; i++)
            assert(old_q[i]==q[i]);
    for(i=old_head+1; i < SIZE; i++)
            assert(old_q[i]==q[i]);}
```

```
int main() {// cbmc q.c –unwind
SIZE+2
    environment_setup();
    enqueue_verify();}
```

```
int main() {// cbmc q.c –unwind
SIZE+2
    environment_setup();
    dequeue_verify();}
```

Ex2. Tower of Hanio

Write down a C program to solve the Tower of Hanoi game (3 poles and 4 disks) by using CBMC

- Hint: you may non-deterministically select the disk to move
- Find the shortest solution by analyzing counter examples. Also explain why your solution is the shortest one.
  - Use non-determinism and `__CPROVER_assume()` properly for the moving choice
  - Use `assert` statement to detect when all the disks are moved to the destination

Top[3]

2	-1	-1

	0	1	2
2	1	0	0
1	2	0	0
0	3	0	0

```
//cbmc hanoi3.c –unwind 7 –no-unwinding-assertions
1:signed char disk[3][3] = {{3,2,1},{0,0,0},{0,0,0}};
2:char top[3]={2,-1,-1};// The position where the top disk is located at.
3:                          // If the pole does not have any disk, top is -1
4:int main() {
5:  unsigned char dest, src;
14:     while(1) {
15:     src = non_det();
16:     __CPROVER_assume(src==0 || src==1 || src==2);
17:     __CPROVER_assume(top[src] != -1);
18:
19:     dest= non_det();
20:     __CPROVER_assume((dest==0 || dest==1 || dest==2) && (dest != src));
21:     __CPROVER_assume(top[dest]==-1 || (disk[src][top[src]] < disk[dest][top[dest]]) );
22:
25:     top[dest]++;
26:     disk[dest][top[dest]]=disk[src][top[src]];
27:
28:     disk[src][top[src]]=0;
29:     top[src]--;
30:
31:     assert( !(disk[2][0]==3 &&  disk[2][1]==2 &&  disk[2][2]==1 ));
} }
```

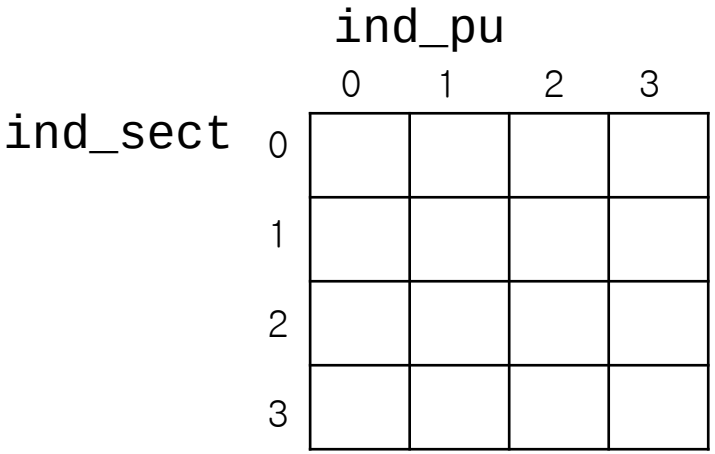
	0	1	2
2	0	0	1
1	0	0	2
0	0	0	3

Ex3. Flash read verification

1. Formal verification of a flash memory reading unit (70 pts)
   - Show the correctness of the `flash_read()`
     - By using randomized testing
       - Randomly select the physical sectors to write four characters and set the corresponding SAMs
     - By using exhaustive testing
       - Create 43680 (16*15*14*13) distinct test cases
         - » Do not print test cases in your hardcopy to save trees
     - By using CBMC
       - Create environment model satisfying the invariant formula by using `__CPROVER_assume()` and nested loops

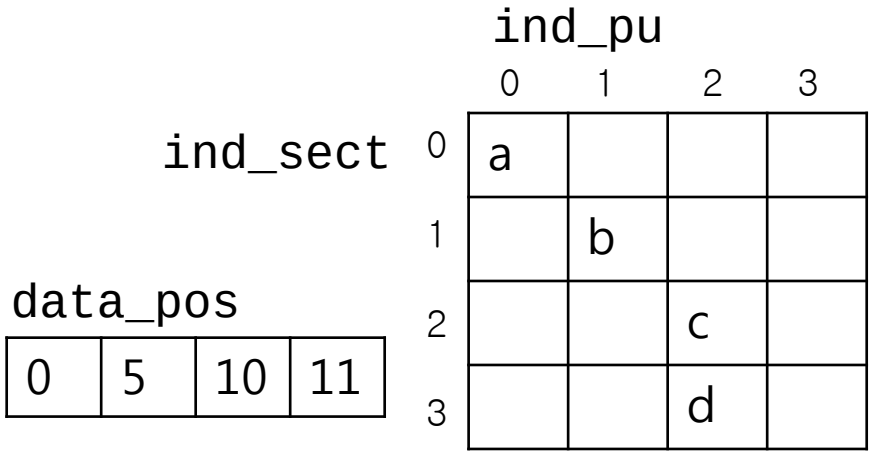
- Problem #1.
  Random solution

```
void main(){
char res[50];
int tc = 0;
int count = 0;
int nTC = 43680;// # of possible distribution
                // 16*15*14*13
while(tc++ < nTC){
     randomized_test();
     flash_read(&res[count],SAM,pu);
     assert(res[0] == 'a' && res[1] == 'b' &&
            res[2] == 'c' && res[3] =='d');
}}
```




```
 #include <stdio.h>
#include <time.h>
#include <assert.h>
#define SECT_PER_U 4
#define NUM_PHI_U 4

typedef struct _SAM_type{
    unsigned char offset[SECT_PER_U];
SAM_type;

typedef struct _PU_type{
   unsigned char sect[SECT_PER_U];
}PU_type;

char data[SECT_PER_U] = "abcd";

PU_type pu[NUM_PHI_U];
SAM_type SAM[NUM_PHI_U];

void randomized_test(){
    unsigned int i = 0, j = 0;
    unsigned char ind_pu, ind_Sect;
    // Initialization
    for(i = 0;i < NUM_PHI_U;i++){
        for(j = 0;j < SECT_PER_U;j++){
            SAM[i].offset[j] = 255;
            pu[i].sect[j] = 0;
    }}

    while (i< SECT_PER_U) {
       ind_pu  = rand()%4;
       ind_Sect= rand()%4;
       if(pu[ind_pu].sect[ind_Sect] == 0){
          pu[ind_pu].sect[ind_Sect] = data[i];
          SAM[ind_pu].offset[i] = ind_Sect;
          i++;
       }
    }
 }
```

- Problem #1.
  Exhaustive solution




```
void exhaustive_test(int *data_pos){
    unsigned int i = 0, j = 0;
    unsigned char ind_pu, ind_Sect;

    for(i = 0;i < NUM_PHI_U;i++){
        for(j = 0;j < SECT_PER_U;j++){
            SAM[i].offset[j] = 255;
            pu[i].sect[j] = 0;
        }
    }

    for(i = 0;i < NUM_PHI_U;i++){
        ind_pu = data_pos[i]/4;
        ind_Sect = data_pos[i]%4;
        pu[ind_pu].sect[ind_Sect] = data[i];
        SAM[ind_pu].offset[i] = ind_Sect;
    }
}
```

```
void main(){
char res[4];
int i, j,k,l, data_pos[4];
 //# of all distributions = 16*15*14*13
for(i = 0;i < NUM_PHI_U * SECT_PER_U;i++){
    for(j = 0;j < NUM_PHI_U * SECT_PER_U;j++){
        if (j == i) continue;
        for(k = 0;k < NUM_PHI_U * SECT_PER_U;k++){
            if (k == i || k == j) continue;
            for(l = 0;l < NUM_PHI_U * SECT_PER_U;l++){
                if(l == i || l == j || l == k) continue;

                data_pos[0] = i;
                data_pos[1] = j;
                data_pos[2] = k;
                data_pos[3] = l;
                exhaustive_test(data_pos);
                flash_read(&res[count],SAM,pu);
                assert(res[0] == 'a' && res[1] == 'b'
                && res[2] == 'c' && res[3] == 'd');
} } } } }
```

- Problem #1. CBMC solution

```
void CBMC_environ_setting(){
    unsigned int i = 0, j = 0;
    unsigned char ind_pu, ind_Sect;
    for(i = 0;i < NUM_PHI_U;i++){
        for(j = 0;j < SECT_PER_U;j++){
            SAM[i].offset[j] = 255;
            pu[i].sect[j] = 0;
    }}

    for(i = 0;i < SECT_PER_U;i++){
        ind_pu = nondet_char();
        ind_Sect = nondet_char();
        __CPROVER_assume(ind_pu>=0 && ind_pu <NUM_PHI_U);
        __CPROVER_assume(ind_Sect>=0 && ind_Sect <SECT_PER_U);
        __CPROVER_assume(pu[ind_pu].sect[ind_Sect] == 0);

        pu[ind_pu].sect[ind_Sect] = data[i];
        SAM[ind_pu].offset[i] = ind_Sect;
    }
}
void main(){
  char res[50];
  int count = 0;
  CBMC_environ_setting();
  flash_read(&res[count],SAM,pu);
  assert(res[0] == 'a' && res[1] == 'b'  && res[2] == 'c' && res[3] == 'd');}
```