Crown Examples

- Basic Examples
- Function Examples
- Limitation Examples

Basic Example 1

```
// Hello Crown example.
// This example shows how to define a symbolic variable.
#include <crown.h> // for Crown
#include <stdio.h>

int main(){
    int x;
    SYM_int(x); // Define x as a symbolic input.

    printf("x = %d\n", x);
    if (x > 100){
        printf("x is greater than 100\n");
    }else{
        printf("x is less than or equal to 100\n");
    }
    return 0;
}
```

```
------------------------
x = 0
x is less than or equal to 100
------------------------
x = 101
x is greater than 100
```

Basic Example 2

```
// Hello Crown example 2 with initial value assigned to
// a symbolic variable using SYM_int_init.
#include <crown.h>
#include <stdio.h>

int main() {
  int x;
  SYM_int_init(x, 7);
  printf("x=%d\n", x);
  if ( x > 10)
    printf("x>10\n");
  else
    printf("x<=10\n");
}
```

```
------------------------
x=7
x<=10
------------------------
x=11
x>10
```

Basic Example 3

```
// Another Hello Crown example.
// Crown can handle linear integer arithmetic expression
// and nested condition statements
#include <crown.h>
#include <stdio.h>
int main() {
    char x, y;
    SYM_char(x);
    SYM_char(y);

    printf("x, y = %d, %d\n", x, y);
    if (2 * x == y){
        if (x != y + 10) printf("Fine here\n");
        else             printf("ERROR\n");
    }
    return 0;}
```

```
------------------------
x, y = 0, 0
Fine here
------------------------
x, y = -10, -20
ERROR
------------------------
x, y = -128, 0
```

Basic Example 4

```
// Symbolic value propagation example.
// In an assign statement, if RHS has a symbolic variable,
// a variable in LHS becomes a symbolic variable
#include <crown.h>
#include <stdio.h>

int main(){
    int x, y;
    SYM_int(x);

    printf("x = %d\n", x);
    y = 2 * x + 3; // y' == 2 * x + 3

    if (y == 7) printf("y(=2x+3) is 7\n"); /* if(y'==7) */
    else        printf("y(=2x+3) is NOT 7\n");
}
```

```
-----------------------
x = 0
y(=2x+3) is NOT 7
-----------------------
x = 2
y(=2x+3) is 7
```

Basic Example 5

```
// SYM_assume() to give constraints on symbolic variables.
#include <crown.h>
#include <stdio.h>
#include <assert.h>

void main() {
    int x, y;
    SYM_int(x);
    SYM_int(y);
    SYM_assume( x + y > 10);
    printf("x=%d, y=%d\n", x, y);
    assert( x + y > 10);
}
```

```
------------------------
### SYM_assume(x + y > 10) is violated at Line 10 (main in basic-example5.c) ###
------------------------
program command is basic-example5
x=11, y=0
```

Basic Example 6 (1/3)

```
// Long symbolic path formula
// generated due to a loop
#include <crown.h>
#include <stdio.h>

int main(){
  int i, x;
  SYM_int(x);
  printf("x=%d\n",x);

  for (i=0; i < x; i++) {
    printf("i=%d\n",i);
    if ( i == 3) {
      printf("i==3 finally\n");
      break;
    }
  }
}
// use print_execution to print a
// symbolic execution path formula
```

TC	x	Sym Path Formula φ	Mdified symbolic path formula φ'	Sol
1	0	!(0<x)	0< x	1
2	1	0<x && !(1<x)	0<x && 1<x	2
3	2	0<x && 1<x && !(2<x)	0<x && 1<x && 2<x	3
4	3	0<x && 1<x && 2<x && !(3<x)	0<x && 1<x && 2<x && 3<x	4
5	4	0<x && 1<x && 2<x && 3<x	COMPLETED	

Basic Example 6 (2/3)

```
// Long symbolic path formula
// generated due to a loop
#include <crown.h>
#include <stdio.h>

int main(){
  int i, x;
  SYM_int(x);
  printf("x=%d\n",x);

  for (i=0; i < x; i++) {
    printf("i=%d\n",i);
    if ( i == 3) {
      printf("i==3 finally\n");
      break;
    }
  }
}
// use print_execution to print a
// symbolic execution path formula
```

Another sequence of solutions generated

TC	x	Sym Path Formula φ	Mdified symbolic path formula φ'	Sol
1	0	!(0<x)	0< x	1
2	1	0<x && !(1<x)	0<x && 1<x	4
3	4	0<x && 1<x && 2<x && 3<x	0<x && 1<x && 2<x && !(3<x)	3
4	3	0<x && 1<x && 2<x && !(3<x)	0<x && 1<x && !(2<x)	2
5	2	0<x && 1<x && !(2<x)	COMPLETED	

Basic Example 6 (3/3)

```
// Long symbolic path formula
// generated due to a loop
#include <crown.h>
#include <stdio.h>

int main(){
  int i, x;
  SYM_int(x);
  printf("x=%d\n",x);

  for (i=0; i < x; i++) {
    printf("i=%d\n",i);
    if ( i == 3) {
      printf("i==3 finally\n");
      break;
    }
  }
}
// use print_execution to print a
// symbolic execution path formula
```

```
----------------------
x=0
----------------------
x=1
i=0
----------------------
x=4
i=0
i=1
i=2
i=3
i becomes 3 finally
----------------------
x=3
i=0
i=1
i=2
----------------------
x=2
i=0
i=1
```

Function Example 1

```
// Simple function example
// Symbolic var. can be passed into a func.
#include <crown.h>
#include <stdio.h>

void test_me(char x, char y){
    // body of test_me is same to basic2 ex.
    if (2 * x == y){
        if (x != y + 10){
            printf("Fine here₩n");
        }else{
            printf("ERROR₩n");
        }
    }
}
```

```
int main(){
    char a, b;

    SYM_char(a);
    SYM_char(b);

    printf("a, b = %d, %d₩n", a, b);
    test_me(a, b);
    return 0;
}
```

```
------------------------
a, b = 0, 0
Fine here
------------------------
a, b = -10, -20
ERROR
------------------------
a, b = -128, 0
```

Function Example 2

```
// Another simple function example.
// A function can return a symbolic value
#include <crown.h>
#include <stdio.h>

int sign(int x){     return (x >= 0);}

int main(){
     int a;
     SYM_int(a);
     printf("a = %d₩n", a);

     if (sign(a) == 0)       printf("%d is negative₩n", a);
     else          printf("%d is non-negative₩n",a);
     return 0;
}
```

```
------------------------
a = 0
0 is non-negative
------------------------
a = -1
-1 is negative
```

Function Example 3

```
// Recursive function example.
// Crown can handle a recursive func.
// A recursive function may generate
// infinite # of iterations.
#include <crown.h>
#include <stdio.h>

unsigned int fac(unsigned int n){
    if (n == 0) return 1;
    else return n * fac(n-1);
}

int main(){
    unsigned int a;
    SYM_unsigned_int(a);
    if(a> 10 ) exit(-1);
    printf("a = %u₩n", a);

    if (fac(a) == 24) printf("Reach!₩n");
    return 0;
}
```

```
-----------------------
a = 0
-----------------------
a = 1
-----------------------
a = 4
Reach!
-----------------------
a = 8
-----------------------
a = 9
-----------------------
a = 10
-----------------------
a = 7
-----------------------
a = 6
-----------------------
a = 5
-----------------------
a = 3
-----------------------
a = 2
-----------------------
Iteration 12 (1s, 0s, 0.135s): covered 6
branches [2 reach funs, 6 reach
branches].(6, 5)
```

Limitation 1: No Binary Library Support

```
// External library example.
// The return value of an external binary function is
// a concrete value, not a symbolic value
#include <crown.h>
#include <stdio.h>
#include <stdlib.h>
int main(){
    int x;
    SYM_int(x);
    printf("x = %d₩n", x);
    if (abs(x) == 4) { // a concrete condition, not a symbolic one
        printf("|x| is 4₩n");
    }else{
        printf("|x| is not 4₩n");
    }
    return 0;}
```

```
------------------------
x = 0
|x| is not 4
```

Solution for Limitation 1: Add Libray Code

```
#include <crown.h>
#include <stdio.h>
#include <stdlib.h>

int abs2(int v) {
  int r = 0;
  if (v < 0) r = - v;
  else r = v;
  return r;
}

int main(){
    int x;
    SYM_int(x);
    printf("x = %d₩n", x);
    if (abs2(x) == 4) { // a symbolic condition
        printf("|x| is 4₩n");
    }else{
        printf("|x| is not 4₩n");
    }
    return 0;
}
```

```
-----------------------
x = 0
|x| is not 4
-----------------------
x = 4
|x| is 4
-----------------------
x = -1
|x| is not 4
-----------------------
x = -4
|x| is 4
```

Limitation 1': (Partial) Binary Library Support

```
// When a target program calls an external library function,
// Crown may cause 'prediction failure' error since Crown
// does not know a body of the external function
#include <crown.h>
#include <stdio.h>
#include <stdlib.h>
int main(){
    int x;
    SYM_int(x);
    printf("x == %d₩n", x);
    if (x == abs(x)){// Generate symbolic path formula using
                     // a concrete return value (e.g., x == 0)
        printf("x is non-negative₩n");
    }else{
        printf("x is negative₩n");
    }
    return 0;}
```

```
------------------------
x = 0
x is non-negative
------------------------
x = -1
x is negative
```

Limitation 2: No Symbolic Pointer

```
include <stdio.h>
int main(){
    int x, y;
    int *ptr;

    SYM_int(x); SYM_int(y);
    // SYM_int_ptr(ptr); // NOT supported
     printf("x=%d, y=%d, *ptr=%d₩n", x, y, *ptr);


    // The following code does not generate a symbolic
    // path formula because no expression in the
    // condition is symbolic
    if (ptr == &x) printf("ptr points to x₩n");
    else if (ptr == &y) printf("ptr points to y₩n");

    if (*ptr == x) printf("*ptr equals to x₩n");
    else if (*ptr == y) printf("*ptr equals to y₩n");
}
```

```
----------------------
x=0, y=0, *ptr=1
----------------------
x=-2, y=1, *ptr=1
*ptr equals to y
----------------------
x=1, y=0, *ptr=1
*ptr equals to x
```

Limitation 3: No Symbolic Array

```
// Array cannot be declared symbolically.
// Instead, each element can be declared symbolically
#include <crown.h>
#include <stdio.h>
int main(){
    int i;
    int array[4];

    // SYM_int(array); // NOT supported
   for(i=0; i < 4; i++)
       SYM_int(array[i]);

    if (array[1] == 3)
          printf("array[1] is 3₩n");
    else printf("array[1] is not 3 but ₩%d₩n",array[1]);
}
```

```
----------------------
array[1] is not 3 but 0
----------------------
array[1] is 3
```

Limitation 4: No Symbolic Index (1/2)

```
// Symbolic dereference is not supported.
// If an array index is a symbolic variable,
// Crown does not generated
// a corresponding symbolic path formula
#include <crown.h>
#include <stdio.h>
int main(){
    int x;
    int array[4];

    SYM_int(x);
    SYM_assume( 0< x && x <=4);

    printf("x = %d₩n", x);
    array[0] = 0;
    array[1] = 1;
    array[2] = x;
    array[3] = 4;

    if (array[x-1] == 3)   printf("ERROR₩n");
    else          printf("Fine₩n");}
```

```
------------------------
### SYM_assume(0< x && x
<=4) is violated at Line 11 (main
in limit4-sym-index.c) ###
------------------------
x = 1
Fine
------------------------
### SYM_assume(0< x && x
<=4) is violated at Line 11 (main
in limit4-sym-index.c) ###
```

Limitation 4: No Symbolic Index (2/2)

```
// Symbolic array index is not supported.
// If an array index is a symbolic variable, Crown does not generated
// a corresponding symbolic path formula
#include <crown.h>
#include <stdio.h>
int main(){
    int x;
    int array[4];

    SYM_int(x);
    SYM_assume( 0< x && x <=4);

    printf("x = %d₩n", x);
    array[0] = 0;
    array[1] = 1;
    array[2] = x;
    array[3] = 4;

    if (array[x-1] == 3)   printf("ERROR₩n");
    else          printf("Fine₩n");
}
```

Should check the following Symbolic path formula

1. (x==1 && array[0] ==3)
2. (x==2 && array[1] ==3)
3. (x==3 && array[2] ==3)
4. (x==4 && array[3] ==3)

Partial Solution for Limitation 4 (1/2)

```
#include <crown.h>
#include <stdio.h>
#define ENUM_4(array, index, ret) ₩
do{ ₩
    switch(index){ ₩
        case 0: ₩
            ret = array[0]; ₩
            break;₩
        case 1: ₩
            ret = array[1]; ₩
            break; ₩
        case 2: ₩
            ret = array[2]; ₩
            break; ₩
        case 3: ₩
            ret = array[3]; ₩
            break; ₩
    } ₩
}while(0);
```

```
int main(){
    int x, tmp;
    int array[4];

    SYM_int(x);
    SYM_assume( 0< x && x <=4);

    printf("x = %d₩n", x);
    array[0] = 0;
    array[1] = 1;
    array[2] = x;
    array[3] = 4;
    // tmp = array[x-1]
    ENUM_4(array, x-1, tmp);

    if (tmp/*array[x-1]*/ == 3){
        printf("ERROR₩n");
    }else{
        printf("Fine₩n");
    }}
```

Partial Solution for Limitation 4 (2/2)

```
int main(){
    int x, tmp;
    int array[4];

    SYM_int(x);
    SYM_assume( 0< x && x <=4);

    printf("x = %d₩n", x);
    array[0] = 0;
    array[1] = 1;
    array[2] = x;
    array[3] = 4;
    // tmp = array[x-1]
    ENUM_4(array, x-1, tmp);

    if (tmp/*array[x-1]*/ == 3){
        printf("ERROR₩n");
    }else{
        printf("Fine₩n");
    }}
```

```
-----------------------
### SYM_assume(0< x && x
<=4) is violated at Line 26 (main
in limit4-sym-index-sol.c) ###
-----------------------
x = 1
Fine
-----------------------
x = 4
Fine
-----------------------
x = 3
ERROR
-----------------------
x = 2
Fine
-----------------------
### SYM_assume(0< x && x
<=4) is violated at Line 26 (main
in limit4-sym-index-sol.c) ###
```

Heuristic Guideline to Overcome the Limitations

```
#include <crown.h>
#include <stdio.h>
int main(){
    int x;
    int array[4];

    SYM_int(x);
    SYM_assume( 0< x && x <=4);

    // Guide Crown to generate TC (x=3)
    // w/o changing program behavior
    // Be careful not to be removed by
    // compiler optimization
    if (x==3) printf("x becomes 3₩n");

    printf("x = %d₩n", x);
    array[0] = 0;
    array[1] = 1;
    array[2] = x;
    array[3] = 4;

if (array[x-1] == 3)  printf("ERROR₩n");
    else          printf("Fine₩n");
```

```
-----------------------
### SYM_assume(0< x && x
<=4) is violated at Line 11 (main
in limit4-sym-index-sol2.c) ###
-----------------------
x = 1
Fine
-----------------------
x becomes 3
x = 3
ERROR
-----------------------
### SYM_assume(0< x && x
<=4) is violated at Line 11 (main
in limit4-sym-index-sol2.c) ###
```

Model Checking vs Concolic Testing

	Model checking	Concolic testing
Analysis approach	Monolithic (i.e., whole analysis should be completed)	Incremental (i.e., analysis results are accumulated step-by-step) - Anytime algorithm
Compositional analysis	No	Yes (analysis of each symbolic execution path is **independent** from each other)
Accuracy	Very high	Very high (per given assert statements) - Known as path model checker
Explicit test inputs	Not generated	Generated
Requires abstraction	Yes	No
Memory consumption	Very high	Low
CPU time consumption	Very high	Very high
External binary library handling	None	Partial
Debugging support	**Limited (except a counter example generated)**	**Fully supported (you can freely use gdb or add your code (e.g.,printf) to analyze each concrete execution)**
Scalability	**Very limited**	**Large**

Various Automated SW Analysis Techniques Have Its Own Pros/Cons and Its Best Uses !!!

https://www.escoffier.edu/blog/culinary-arts/different-knives-and-the-best-uses-for-each/