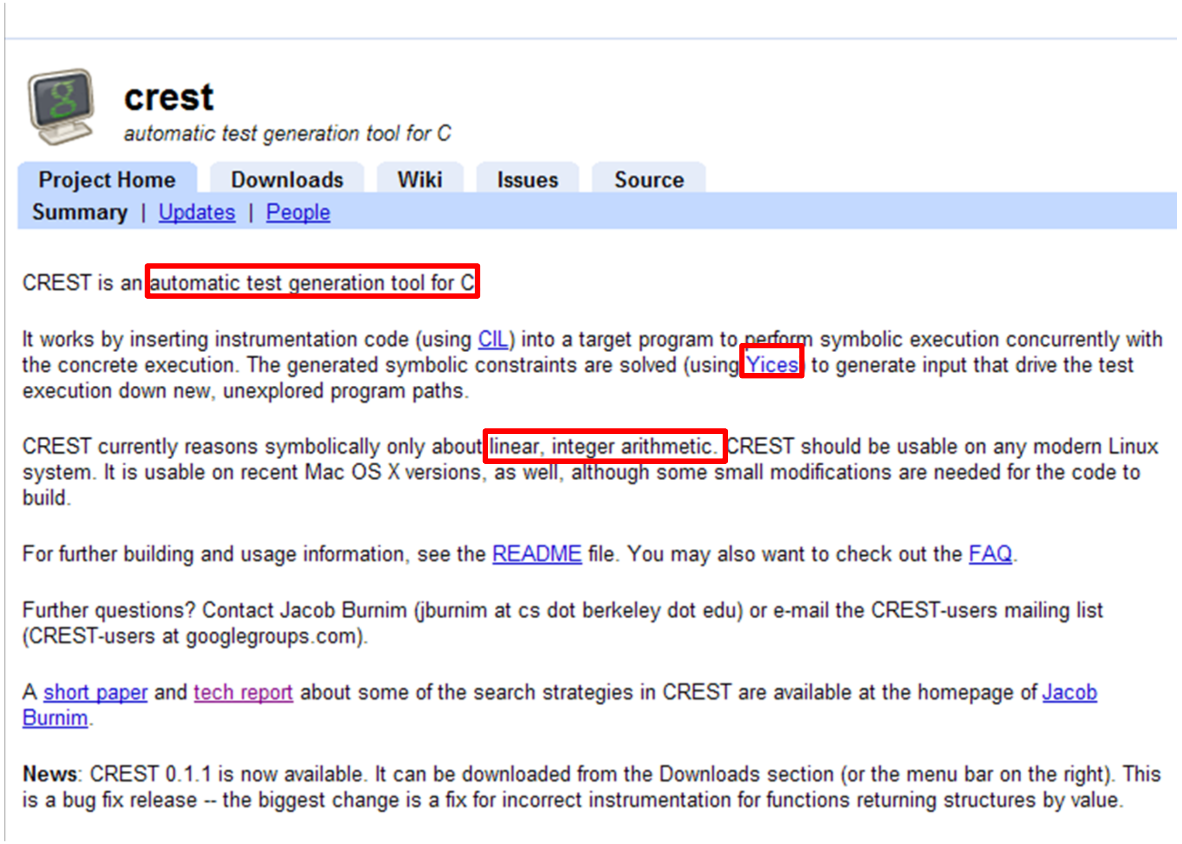
CREST Tutorial

Moonzoo Kim

Provable Software Laboratory

CS Dept. KAIST


crest
automatic test generation tool for C
Project Home
Downloads
Wiki
Issues
Source
Summary | Updates | People

CREST is an automatic test generation tool for C

It works by inserting instrumentation code (using CIL) into a target program to perform symbolic execution concurrently with the concrete execution. The generated symbolic constraints are solved (using Yices) to generate input that drive the test execution down new, unexplored program paths.

CREST currently reasons symbolically only about linear, integer arithmetic. CREST should be usable on any modern Linux system. It is usable on recent Mac OS X versions, as well, although some small modifications are needed for the code to build.

For further building and usage information, see the README file. You may also want to check out the FAQ.

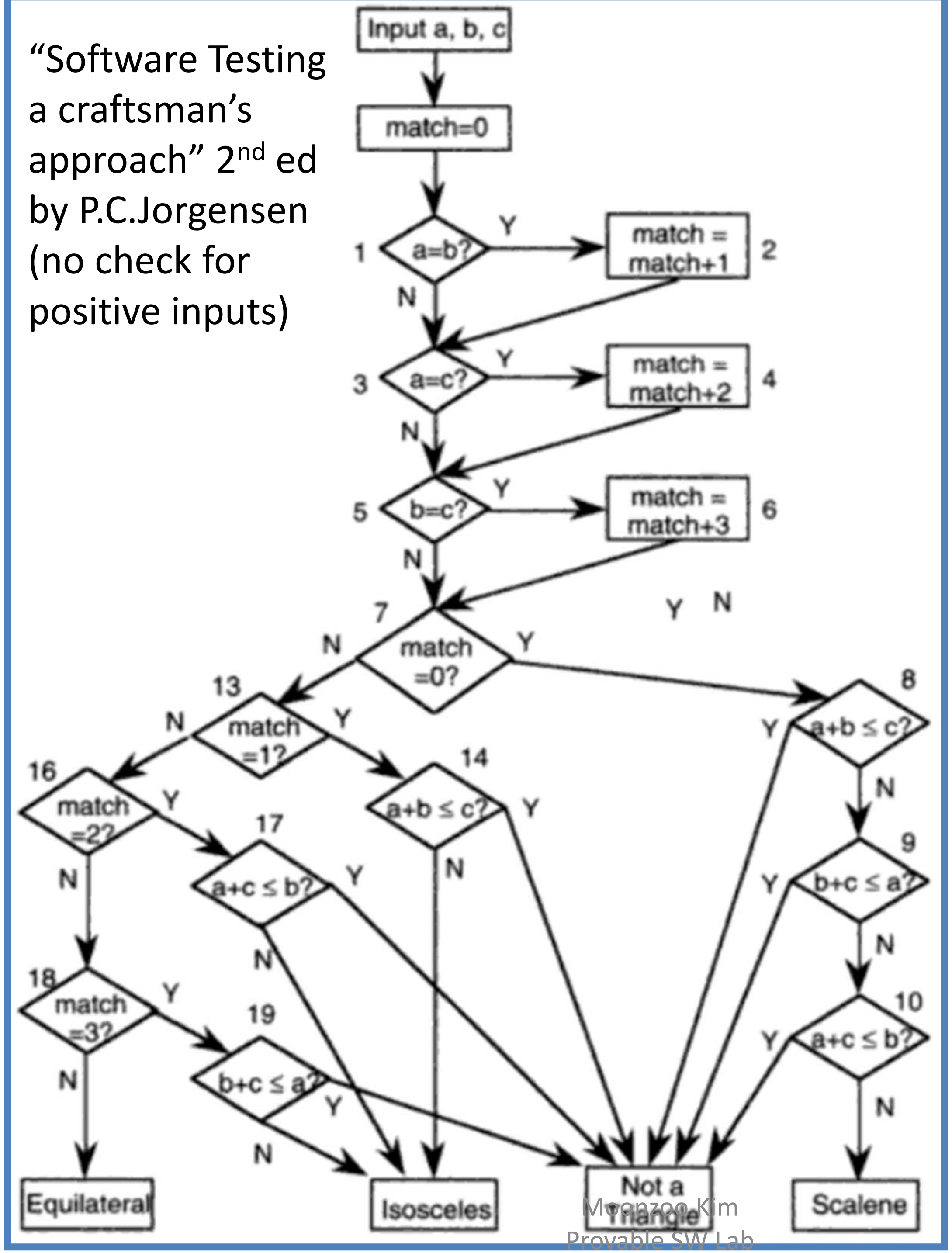
Further questions? Contact Jacob Burnim (jburnim at cs dot berkeley dot edu) or e-mail the CREST-users mailing list (CREST-users at googlegroups.com).

A short paper and tech report about some of the search strategies in CREST are available at the homepage of Jacob Burnim.

News: CREST 0.1.1 is now available. It can be downloaded from the Downloads section (or the menu bar on the right). This is a bug fix release -- the biggest change is a fix for incorrect instrumentation for functions returning structures by value.

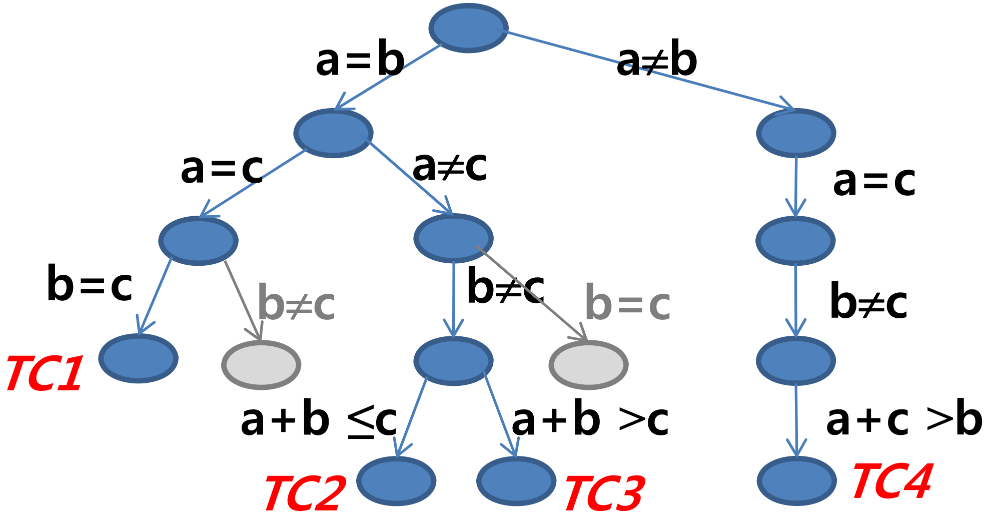
```
#include <crest.h>
main() {
   int a,b,c, match=0;
   CREST_int(a); CREST_int(b); CREST_int(c);
   // filtering out invalid inputs
   if(a <= 0 || b<= 0 || c<= 0) exit();
   printf("a,b,c = %d,%d,%d:",a,b,c);

   //0: Equilateral, 1:Isosceles,
   // 2: Not a traiangle, 3:Scalene
   int result=-1;
   if(a==b) match=match+1;
   if(a==c) match=match+2;
   if(b==c) match=match+3;
   if(match==0) {
      if( a+b <= c) result=2;
      else if( b+c <= a) result=2;
      else if(a+c <= b) result =2;
      else result=3;
   } else {
      if(match == 1) {
         if(a+b <= c) result =2;
         else result=1;
      } else {
         if(match ==2) {
            if(a+c <=b) result = 2;
            else result=1;
         } else {
            if(match==3) {
               if(b+c <= a) result=2;
               else result=1;
            } else result = 0;
         } }}
   printf("result=%d\n",result);
}
```

"Software Testing a craftsman's approach" 2nd ed by P.C.Jorgensen (no check for positive inputs)



Concolic Testing the Triangle Program

Test case	Input (a,b,c)	Executed path conditions (PC)	Next PC	Solution for the next PC
1	1,1,1	a=b ∧ a=c ∧ b=c	a=b ∧ a=c ∧ b≠c	Unsat
			a=b ∧ a≠c	1,1,2
2	1,1,2	a=b ∧ a≠c ∧ b≠c ∧ a+b ≤c	a=b ∧ a≠c ∧ b≠c ∧ a+b > c	2,2,3
3	2,2,3	a=b ∧ a≠c ∧ b≠c ∧ a+b >c	a=b ∧ a≠c ∧ b=c	Unsat
			a≠b	2,1,2
4	2,1,2	a≠b ∧ a=c ∧ b≠c ∧ a+c>b	a≠b∧ a=c ∧ b≠c ∧ a+c ≤b	2,5,2



Moonzoo Kim
Provable SW Lab

KAIST

CREST Commands

- `crestc <filename>.c`
  - Output
    - `<filename>.cil.c` // instrumented C file
    - `branches` // lists of paired branches
    - `<filename>` // executable file
- `run_crest ./filename <n> -[dfs|cfg|random|random_input|hybrid]`
  - `<n>`: # of iterations/testings
  - `dfs`: depth first search (but in reverse order)
  - `cfg`: uncovered branch first
  - `random`: negated branch is randomly selected
  - `random_input`: pure random input
  - `hybrid`: combination of dfs and random

Instrumented C Code

```
#line 10
 { /* Creates symbolic expression a==b */
  __CrestLoad(36, (unsigned long )(& a), (long long )a);
  __CrestLoad(35, (unsigned long )(& b), (long long )b);
  __CrestApply2(34, 12, (long long )(a == b));
  if (a == b) {

    __CrestBranch(37, 11, 1); //extern void __CrestBranch(int id , int bid , unsigned char b )
    __CrestLoad(41, (unsigned long )(& match), (long long )match);
    __CrestLoad(40, (unsigned long )0, (long long )1);
    __CrestApply2(39, 0, (long long )(match + 1));
    __CrestStore(42, (unsigned long )(& match));
     match ++;

  } else {
    __CrestBranch(38, 12, 0);
  }  }
```

Execution Snapshot

```
[moonzoo@verifier crest]$ run_crest ./triangle 10000 -dfs
Iteration 0 (0s): covered 0 branches [0 reach funs, 0 reach branches].
Iteration 1 (0s): covered 1 branches [1 reach funs, 32 reach branches].
Iteration 2 (0s): covered 3 branches [1 reach funs, 32 reach branches].
Iteration 3 (0s): covered 5 branches [1 reach funs, 32 reach branches].
a,b,c = 1,1,1:result=0
Iteration 4 (0s): covered 13 branches [1 reach funs, 32 reach branches].
a,b,c = 2,1,1:result=2
Iteration 5 (0s): covered 17 branches [1 reach funs, 32 reach branches].
a,b,c = 2,1,2:result=1
Iteration 6 (0s): covered 20 branches [1 reach funs, 32 reach branches].
a,b,c = 1,2,1:result=2
Iteration 7 (0s): covered 21 branches [1 reach funs, 32 reach branches].
a,b,c = 3,2,1:result=2
Iteration 8 (0s): covered 24 branches [1 reach funs, 32 reach branches].
a,b,c = 2,1,3:result=2
Iteration 9 (0s): covered 25 branches [1 reach funs, 32 reach branches].
a,b,c = 4,3,2:result=3
Iteration 10 (0s): covered 27 branches [1 reach funs, 32 reach branches].
a,b,c = 2,3,1:result=2
Iteration 11 (0s): covered 28 branches [1 reach funs, 32 reach branches].
a,b,c = 3,2,2:result=1
Iteration 12 (0s): covered 29 branches [1 reach funs, 32 reach branches].
a,b,c = 2,2,1:result=1
Iteration 13 (0s): covered 31 branches [1 reach funs, 32 reach branches].
a,b,c = 1,1,2:result=2
Iteration 14 (0s): covered 32 branches [1 reach funs, 32 reach branches].
elapsed time = 0.0015093
```

```
[moonzoo@verifier crest]$ cat coverage
3 /* covered branch ids*/
4
5
6
7
8
11
12
14
15
17
18
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
```

Supported Symbolic Datatypes

- #define CREST_unsigned_char(x) __CrestUChar(&x)
- #define CREST_unsigned_short(x) __CrestUShort(&x)
- #define CREST_unsigned_int(x) __CrestUInt(&x)
- #define CREST_char(x) __CrestChar(&x)
- #define CREST_short(x) __CrestShort(&x)
- #define CREST_int(x) __CrestInt(&x)

Decision/Condition Coverage Analysis by CREST

```
int main(){
    int A, B, C, D;
    if (A && B || C && D){
        printf("Yes\n");
    }else{
        printf("No\n");
    }
}
```

```
if (A != 0) {
    __CrestBranch(5, 2, 1);    A == T
    if (B != 0) {
      __CrestBranch(10, 3, 1); A == T && B == T
      printf("Yes\n");
    } else {
      __CrestBranch(11, 4, 0); A == T && B != T
      goto _L;
    }
  } else {
    __CrestBranch(6, 5, 0);    A != TRUE
    _L: /* CIL Label */
    if (C != 0) {                  (A != T || A == T && B != T)
      __CrestBranch(16, 6, 1);     && C == T
      if (D != 0) {
        __CrestBranch(21, 7, 1);     (A != T || A == T && B != T)
        printf("Yes\n");             && C == T && D == T
      } else {
        __CrestBranch(22, 8, 0);     (A != T || A == T && B != T)
        printf("No\n");              && C == T && D != T
      }
    } else {
      __CrestBranch(17, 9, 0);     (A != T || A == T && B != T)
      printf("No\n");              && C != T
    }
  }
```

- CREST consider all possible cases with short-circuit
- Thus, coverage reported by CREST might be lower than actual branch coverage