C Program Correctness Checker

A Logic Programming project

by David Pouliot

Symbolic Input

Statically analyze a C program with symbolic input to check for errors.

- Integer Overflow
- Array Overflow
- Divide by Zero
- Struct Overflow

Example – Int Overflow

```
/* Int_overflow_test.c */
int main(int argc, char** argv)
{
  int elements;  // elements is symbolic
  int y = elements + 5; // potential int overflow
  return 0;
}
```

Example – Array Overflow

```
/* dynamic_array_test.c */
int main(int argc, char** argv){
   int elements;  // elements is symbolic
   int* some_array = malloc(elements * sizeof(int));
   some_array[10] = 42;          /* overflow if elements < 10 */
   free(some_array);
return 0;
}
```

Game plan

- Design a Haskell Data Structure to represent a C program
- Write a parser for this data structure that parses the program and creates a solver query
- Pass the query to the solver to see if any errors are found

The Parser

C code:	Yices code
Int index;	(define index::int)
index = 8 + 5;	(assert (= (+ 8 5) index))

Prefix notation

All the C expressions had to be converted to prefix notation for the Yices solver.

X = 5 + 4 + 9 / 7 * 33

Becomes

(= (+ (* 33 (/ 9 7)) (+ 5 4)) x)

Shunting-yard algorithm

I used a version of the shunting yard algorithm by Dijkstra

Like the evaluation of RPN, the shunting yard algorithm is stack-based. For the conversion there are two text variables (strings), the input and the output. There is also a stack that holds operators not yet added to the output queue. To convert, the program reads each symbol in order and does something based on that symbol.

http://en.wikipedia.org/wiki/Shunting-yard_algorithm

Test 1

```
t2 = ["/* static_array_test */",
      "int main(int argc, char** argv)",
      "{",
      "int index;",
      "int next;",
      "index = 5 + 8;",
      "next = index * 2;",
      "}"]
```

Test1

```
(define index::int)
(define next::int)
(assert ( = ( + 5 8 ) index ))
(assert ( = ( * 2 index ) next ))
(assert (or (> index 2147483647)
      (< index -2147483648)
      (> next 2147483647)
      (< next -2147483648) ))
(check)
(show-model)
```

Test1 results

yices test.ys

unsat

unsat

The context is unsat. No model.

Test2

```
t3 = ["/* static_array_test */",
"int main(int argc, char** argv)",
"{",
"int index;",
"int next;",
"next = index * 2;",
"}"]
```

Test2

```
(define index::int)
(define next::int)
(assert ( = ( * 2 index ) next ))
(assert (or (> index 2147483647)
            (< index -2147483648)
            (> next 2147483647)
            (< next -2147483648) ))
(check)
(show-model)
```

Test2 results

yices test.ys

sat

(= index -2147483649)

(= next -4294967298)

Branches

```
Int x, y;
If( x < 10)
    y = 4;
else
    y = 6;
```

```
(define x::int)
(define y::int)
(assert (<  x 10 ))
(assert (= y 4))
-----------------------------------
(define x::int)
(define y::int)
(assert (>=  x 10 ))
(assert (= y 6))
```

Alternative to branching

if-Then-Else

Yices provides an if-then-else construct that applies to any type. An if-then-else term can

be written using either one of the two following forms

(ite <c> <t1> <t2>) (if <c> <t1> <t2>)

Arrays

How to represent?

- Function types - Yices does not have a distinct type construct for arrays. In Yices, arrays are the same as functions
- Bitvector
- N separate variables

Loops

```
Int x = 0;
For (i = 0; i < 10; i++){
    X++;
}
(define x::int)
(define i::int)
(assert (<  i 10 ))
(assert (= x ??))
```

Function Calls

- Ignore if void function and the function doesn't modify global state
- If there is a return value, analyze the function being called to get constraints on possible return values

Function Call Example

```
/* Int_overflow_test.c */
int main(int argc, char** argv)
{
   Int elements;  // symbolic
   int y = test(elements );
}
Int test(int e){
    Return e % 100;
}
```

```
(define elements::int)
(define y::int)
(assert (and
                (>= y 0)
                (< y 100)))
```

Future work

Implement

- Branching
- Arrays
- Structs
- Loops
- Function Calls
- Recursion

Questions?