A static analyzer for finding dynamic programming errors

2002. 11. 19

Traditional Check's Problems

- for code generation, not for error-check
- Inter-procedural (interactions between function are not detected)
- reporting false errors. (non-achievable path)
- Require substantial additional work by the user
- requiring test cases

Goal

- Develop a source code analyzer that could find Purify-like errors with Purify's ease of use, but without needing test cases.

Requirement

- C and C++ program should be checked effectively.
- Information should be derived from the program text rather than acquired through user annotations.
- Analysis should be limited to achievable paths
- The information produced from the analysis should be enough to allow a user to characterize the underlying defects easily.

PREfix

- Simulation: Use Virtual Machine => Memory Test & restrict *achievable* path.
- The behavior of a function : Model
- Model is automatically generated by information extracted from programs.
- Bottom-Up: can apply an entire program, or subset of a program.

The Advantage of Simulation

- Easily get achievable path
- **Memory: exact values and predicates**
  - Memory Layout - Bit Operation - Dereference
- **End-of-path analysis**
  Any unreachable memory or resource that has been allocated, but not freed, is reported.

(*disadvantage*)
achievable #paths is often quite large -> samples of achievable paths to simulate.
required a user config

Pseudo-code for function simulation

while (there are more paths to simulate){
  initialize memory state

  simulate the path, identifying inconsistencies and updating the memory state

  perform end-of-path analysis using the final memory state,

  identifying inconsistencies and creating per-path summary
}

combine per-path summaries into a model for the function

Conditions, assumptions and choice points

```
4 char *f(int size)
5 {
6       char *result;
7
8       if (size = 0)
9           result = (char *)malloc(size);
10      if (size == 1)
11          return NULL;
12      result[0] = 0;
13      return result;
14 }
```

We don't know the value of "size".

- Assumption: size = 0
- Condition: 8: if (size = 0) == false
- Choice points: 10: if (size == 1) *(need to make assumption : size == 1 , size != 1)*

- Assumption: size == 0
- Condition: 8: if (size = 0) == false
  10: if (size == 1) == false
- Choice points:

Models

- MODEL: The behavior of a function: model consists of exclusive outcomes.

OUTCOME

- *Guards* -> an element of test set (input dependent)
- *Constraint* -> precondition
- *Results* -> postcondition

Ex. Model

```
1  int deref(int *p)      //deref
2  {                              (param p)
3      if (p==NULL)           (alternate return_0
4          return NULL;           (guard pos p NULL)
5      return *p;                 (constraint memory_initialized p)
6  }                              (result pos return NULL)
                                  )
                              (alternate return_*
                                  (guard pos p !NULL)
                                  (constraint memory_initialized p)
                                  (constraint memory_valid_pointer p)
                                  (constraint memory_initialized *p)
                                  (result pos return *p)
                                  )
                              )
```

Figure 8. The model of the dereferencing function.