Learn to Build Automated Software Analysis Tools with Graph Paradigm and Interactive Visual Framework

GIAN, September 12-16, 2016

Suresh C. Kothari
Richardson Professor
Department of Electrical and Computer Engineering

Ben Holland, Iowa State University

How complex problems are solved?

Acknowledgement: **Team members at Iowa State University and EnSoft, DARPA contracts FA8750-12-2-0126 & FA8750-15-2-0080**

Day 1- Overview

- Lecture Topics:
  - Experiments and Abstractions
  - Control and Data Flow
  - Program Graphs as Micro and Models
  - An introduction to inter-procedural analysis
- Lab: Dissecting a complex problem – illustrated with an instance of a memory leak problem from the XINU operating system

Memory Leak Example 1

```
dsread(devptr, buff, block)
     struct  devsw   *devptr;
     char *buff;
     DBADDR  block;
{
     struct  dreq *drptr;
     int stat;
     char ps;

     disable(ps);
     drptr = (struct dreq *) getbuf(dskrbp);
     drptr->drdba = block;
     drptr->drpid = currpid;
     drptr->drbuff = buff;
     drptr->drop = DREAD;
     if ( (stat=dskenq(drptr, devptr->dvioblk)) == DONQ) {
          stat = drptr->drstat;
     }
     freebuf(drptr);
     restore(ps);
     return(stat);
}
```

Verification Easy!

Memory Leak Example 2

```
dswrite(devptr, buff, block)
    struct  devsw   *devptr;
    char *buff;
    DBADDR  block;
{
    struct  dreq *drptr;
    char ps;

    disable(ps);
    drptr = (struct dreq *) getbuf(dskrbp);
    drptr->drbuff = buff;
    drptr->drdba = block;
    drptr->drpid = currpid;
    drptr->drop = DWRITE;
    dskenq(drptr, devptr->dvioblk);
    restore(ps);
    return(OK);
}
```

Verification Hard!

Let us get to the fundamentals of what makes it hard.

Let's try to Characterize *Hardness*

```
dswrite(devptr, buff, block)
     struct   devsw    *devptr;
     char *buff;
     DBADDR   block;
{
     struct   dreq *drptr;
     char ps;

     disable(ps);
     drptr = (struct dreq *) getbuf(dskrbp);
     drptr->drbuff = buff;
     drptr->drdba = block;
     drptr->drpid = currpid;
     drptr->drop = DWRITE;
     dskenq(drptr, devptr->dvioblk);
     restore(ps);
     return(OK);
}
```

Interprocedural – not a good enough characterization of hardness, Why?

Let’s Look a Little Deeper - 1

4 paths

```
dskenq(drptr, dsptr)
      struct   dreq *drptr;
      struct   dsblk    *dsptr;
      { struct dreq *p, *q;
      if ( (q=dsptr->dreqlst) == DRNULL ) {
            dsptr->dreqlst = drptr;
   1        dskstrt(dsptr);
            return();}
      for () {
    2       if ((st=dskqopt(p, q, drptr)!=SYSERR))
                        return();
            if (   ) {
        3         q->drnext = drptr;
                  return();} }
  4   q->drnext = drptr;
      return();   }
```

Path 1: drptr passed to dskstrt(), which does not deallocate –a memory leak?

How will you solve this *ranger* problem?

Ralph is a forest ranger. From each of the four windows of his lookout station, Ralph can see three campsites. The north window shows campsites A, B, C; the east window shows campsites C, E, H; the south window shows campsites F, G, H; the west window shows campsites A, D, F. One day Ralph counted five campers through each of the four windows. There were 14 campers altogether and at least one camper at each campsite. What are the all possible arrangements of the campers in the campsites, i.e. how many campers at each campsite?

The work of a genius

Problem 1: John is 3 years older than Jill. Together their ages add up to their mother's age. Their mother is 45 years old. How old are John and Jill?

Problem 2: A water tank holds 3 gallons of water more than another water tank. Together the two water tanks can hold 45 gallons of water. What is thecapacity of each water tank?

Are these different problems?

Without knowledge of algebra, the solution will be by trial and error

$$\boxed{\begin{array}{l} X - Y = 3 \\ X + Y = 45 \end{array}}$$

Algebra = Abstraction + Computation

A solution to the *ranger* problem

Ralph is a forest ranger. From each of the four windows of his lookout station, Ralph can see three campsites. The north window shows campsites A, B, C; the east window shows campsites C, E, H; the south window shows campsites F, G, H; the west window shows campsites A, D, F. One day Ralph counted five campers through each of the four windows. There were 14 campers altogether and at least one camper at each campsite. What are the all possible arrangements of the campers in the campsites, i.e. how many campers at each campsite?

$$\downarrow$$

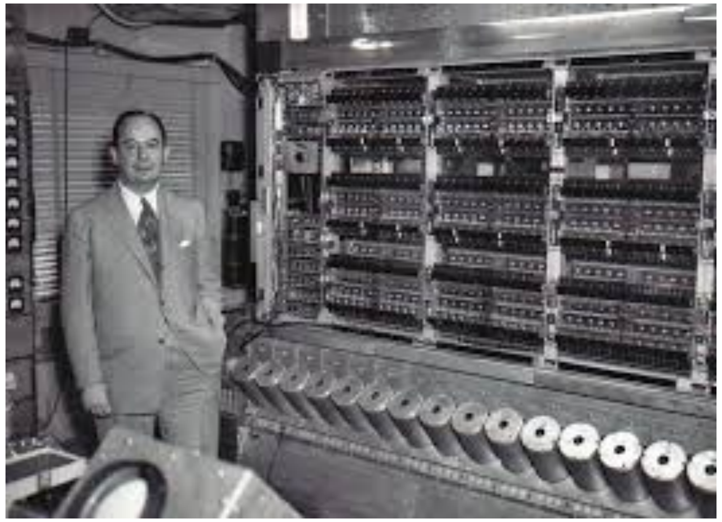
$$\boxed{\begin{aligned} &A+B+C = 5\\ &C+E+H = 5\\ &F+G+H = 5\\ &A+D+F = 5\\ &A+B+C+D+E+F+G+H = 14 \end{aligned}}$$

Elegant abstraction

Have you encountered a problem where you could think of multiple abstractions, however, among the many abstractions there was just one *elegant abstraction* to solve the problem?

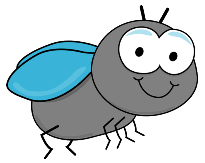
elegant abstraction = simple method to solve the problem + makes us wiser

A story with two abstractions



A car travels from **B** to **A** and fly goes from **A** to **B**. The car as soon as it meets the fly turns back, comes back to **B**, and start again towards **A**. The car does this repeatedly until the fly reaches **B**. What is the total distance travelled by the car? The car and the fly speeds are respectively 20 and 5 miles per hour. The distance AB is 10 miles.

5 miles/hour



20 miles/hour

distance = 10 miles

A

B

Abstraction: D_i: the distance travelled by the car for the i-th meeting

Total distance = sum of the infinite series $D_1+D_2+..$

Abstractions for solving software problems

- We cannot do software abstractions by hand – we need a tool.
- Important questions:
  - What abstractions do we create for a given a problem?
  - What computations do we perform on the abstraction to solve the problem?
  - Given two problems do they need different abstractions?
- Have you abstracted software to solve a problem?

A loop iteration count problem

```
for $I$ = 1 to $N$
        for $j$ = 1 to $i$
                count = count +1;
```

How many times will the innermost loop iterate?

i	1	2		3						
j	1	1	2	1	2	3				

The answer is $1 + 2 + 3 + \ldots + (N-2) + (N-1) + N$

The formula $\frac{1}{2} N(N+1)$ works for counting the number of iterations of a 2-level loop.

An abstraction that would make us wiser?

Consider a **k**-level nested loop. For example, a loop with **k** = 3

```
for $I$ = 1 to $N$
        for $j$ = 1 to $i$
        for $l$ = 1 to $j$
        $count$ = $count$ +1;
```

How many times will the innermost loop iterate?