Review 19

4/3/12

CS6V81.502

Seungtack Baek

**Bugs as Deviant Behavior:**

**A General Approach to Inferring Errors in Systems Code**

In this Paper, Engler et al. demonstrates how to find errors in System code (such as Linux and OpenBSD). To find errors, they had to infer rules (what they called, *beliefs –* not *truth*) from the source code without a priori knowledge of the code itself. In the process of inferring rules, they took the source code and inferred beliefs from each statement.

For an example, they inferred a null check, such as *p ==* NULL, as p could either be null or not null, and dereference of *p*, such as *p->func()*, as p cannot be a null pointer. So, if you dereference and apply null check, it will be an unnecessary check or vice versa.

On the other hand, they also implemented a concept of MUST belief and MAY belief. MUST belief is one that should be obeyed for entire source code and the counter example will be marked as bug. Unlike MUST belief, MAY belief is inferred from statistical analysis. If “majority” follows a certain rule, it is more likely to be a rule that programmers should follow and the counter example will also be marked as a bug or something to be checked upon.

They made an interesting point in this paper. Towards the end of section 3.3, the author states, “Programmers are usually right. If they are not, then we have much bigger concerns than a few concurrency bugs.” I “usually” agree with this statement, but the question comes to my mind, “Doesn’t bugs *usually* occur when programmers are not right?” In my opinion, assuming that programmers are right, while trying to find programmers’ mistake can lead to contradictions.

Yet I do want to see how many logics they included in the inferring a belief, such as checking for nulls, pointer dereferencing, lock, etc. I am pretty sure that there are a lot more rules than just this handful of them.

**Question:**

1. I do not believe that they mentioned anything about what happens on branching, (or jumping, rather). What happens if their algorithm encounter

while(p->flag())

{

// do something..

if(p==NULL)

{

continue;

}

else

{

break;

}

}