웹 소프트웨어 신뢰성

- ✓ **Instructor: Gregg Rothermel**
- ✓ **Institution: 한국과학기술원**
- ✓ **Dictated: OES 봉사단**

Okay, so my name is Pablo, this is the first presentation of the paper for this class.

And I am going to present this paper, the title is efficient symbolic execution of strings for validating web applications.

Okay, the authors are mainly from the University of Texas at Austin.

This was presented in a workshop, that is, that was held with the EASTA conference in 2009.

Okay, so it's a workshop paper, so it's a shorter paper and small appliance.

So it was, when I was trying to find something that relates symbolic execution in a web environment, this was, there aren't many papers written with that.

I don't know why.

But, I found this and although the idea seems good, this paper has some drawbacks.

So maybe we can analyze them.

Okay, that's the short overview of what is symbolic execution.

So, well, symbolic execution is a kind of an analysis programs and we consider us see for each variable and the symbolic, this symbolic values instead of the concrete values.

So given that, we can generate a set of constraints.

Okay, for instance, for each predicates, or other structures like "if" statement, okay?

We can get constraints that we can work on them as inequalities.

Alright? Uh, both the real constraints and its negations is its condition, okay?

And it is store in something called "bad condition." Okay?

One of the main applications for symbolic execution is the test case generation.

The test case generation, how can we achieve that? Well, we can do that solving the constraints.

Okay, so the result of this process is for each variable is, that we contain in these set of inequalities, we can have a numeric called interbug, or which we can pick our concrete value that will be used in all these cases.

Okay, thus, for each path, there are the so-called intervals, contains and values that we can guarantee that covers the path. So this is an example.

This is our small program.

And as you can see, we start traversing this flow of this representation.

At the end, you can see some, some inequalities.

Alright, the one that is this, this is just one.

Okay, but if you go by the other, we have this, and at the end, we can have this, which corresponds to the second “if” statement, which leads us to these two sets of inequalities.

Given this, we can solve this. Okay, if we solve this, we will have values, numerical values for X and Y.

That will guarantee that this path, will be covered. Okay?

In this case, the set of inequalities is invisible.

So we cannot do anything.

That’s the main idea how symbolic execution works.

This is another, something that I did yesterday.

So I mean if we want to generate that this case of this spot, we start to collect all the inequalities from each of these notes and in the end we will have this.

So we will solve this, and we will get a numeric interval from this one.

And from that we can get this.

In terms of the main issues, the symbolic execution is that path explosion.

So it deals with this possibility of this system.

Also, there are some issues of how to handle none-numerical values, for instance, this constraint, and the other one is that usually has to be instructed in a way we deals the performance of the system, and the amount of time that we will spend will be higher. (Professor talking)

Oh, that is, that is one other issues.

That, there is something called "generalized symbolic execution" that handles this kind of execution of some way.

There are some approaches that generate just account value for the loop.

So you cover the loop that traversing the loop once, exactly.

But that deals with the symbolic execution.

If you just have that explosion with simple programs, if you want to handle loops in, in the finer way. (Professor talking)

Constraint solver.

Constraint solver is this.

I want to say it's a program.

Let's say this is infrastructure of something like that.

More like, uh, that deals with and can provide the solution.

And numerical solution, or multi-numerical based of representation validation of either buzz that are containing are these feasible solution of this set-up of inequalities. (Professor talking)

Yes. What we are having here is, okay, the constraint solver is given the current inequalities, we can't find any numerical solutions.

Ah, it depends on what you're looking for.

For instance, you can reverse in an empty set or for instance, the choco, there is the one that uses the symbolic choco (08:44).

It allows us to represent the output of us that something that you can handle easily, for instance, no value, or something. (Professor talking)

No, not always.

This, the result is not sure.

For instance, there are some implementations of the symbolic execution, for instance, in this case, this is not a feasible solution, it generates a backtracking function that goes back to the previous sets, okay?

Because maybe in the previous set, there is a feasible solution.

So you can get in this case for, not for the entire path, until the lowest one but for the ones in between.

There are sort of people who built this things that are usually pretty sophisticated mathmatically trying to find, trying to build ones that are successful are higher probability of the time basically.

I guess in this paper they are basically comparing two ways of instrumenting, some code again, so the first one that they presented is something called a blind instrumentation.

What is the mean of the blind in this case is like do not consider any specific activities of the code, just transform every element into our presentation that can be useful for this symbolic analysis.

So the key elements are first week we changed the elements in the program of the variables the operations under predicates.

Also here is some assumption. We will add new variables on the original variable can hold a symbolic value.

So at the beginning I supposed that they define the set of types of variable step can handle symbolic values.

For instance the most common are inter-gears float numerically.

Also the operations are replaced by method calls so they can handle that in a easy way.

The predicates are also replaced by the method calls.

In this terms of the predicates, there is a relevant question.

Found this like How to handle predicates that have symbolic inputs. Is that true or is that false., The symbolic value is not the real value right?

It’s a symbolic presentation of the variable use.

So in this case the others just add the two branches to the bad condition.

So if it is true, and this is false also it’s added to the analysis.

And then they perform back tracking when the bad condition which means the set of qualities are unsatisfiable.

Ok, in this case, How to choose which elements should be changed in the code? As well you’ve seen the blind instrumentation this instrumenter doesn’t know which variable code can contain symbolic expressions so again another assumption that we can discuss is it assumes that all variables can contain symbolic expressions. (Student asking a question)

For instance, I mean that this that we are here.

Ok we want to generate this case this but we are here this set is unfeasible, We cannot solve this.

Thus we can back track to the previous subset.

In this case, this example is very vulnerable because this set is from by two in qualities, So the previous one will be just one.

But in a bigger program you can have multiple in qualities, if you perform back tracking, you will go to the previous one that maybe has more than one. (Student asking a question)

Yes, in some extent (Professor talking)

That's the main idea it but it seems that you can get the foot coverage.

That's quite impossible.

I think the purpose of symbolic execution in this paper is checking the program, verifying the program, clarifying the program, so like symbolic executions are possible passes in the program.

(Professor answering the student's question) Maybe that's related to they presented after related to their requirements.

How to verify given a requirement you have to transform certain variables, certain inputs into symbolic inputs. (Professor summarizing for the student)

(Student asking a question).

Yeah well, as you are considering both though, both branches the positive and the negative, when you backtrack you can go to the previous one and then to the predicates in this case, if this is not feasible ok, backtracking and then go to this.

So in that way, yes you are right.

That's because once explore all the paths if you can stuck somewhere you are not in the backtrack.

But another thing is symbolic executes, it may not be able to solve on the red one there, we know there is no inputs that's going to, we know that' we are looking at.

But in general, constraints are may not be able to find a solution and that maybe due to weakness in the solver not due to thing that beyond unsolvable at all

So if you are tester generally inputs and go back to my scenario where I am just trying to reach that one statement, So not what they are doing in this paper, I'm just trying to meet one statement, All right, If the solver cannot find inputs to reach that statement it might trying to tell me inputs that reach predicate above it.

And now, as the tester I can say ok it couldn't reach that statement maybe it's still reachable, it may be due to the weakness in this solver.

But here at least it starts here are the inputs required to reach predicate before let me think about whether I can go farther.

So we were in this slide right? So ok in this case, the instrumenter as you know which variables could contain symbolic expressions.

So it assumes all of them.

So this is the first example of blind instrumentation and you can see here whole each expression has been a change.

The second approach is called intelligent instrumentation.

Tries to talk one of the main concerns about instrumentation is When we instrument, we are additional over hear.

Because we're adding maybe useless code.

So I found two ways of generating instrumentation.

The first one is the one that is in this paper which is transformation, using ad-hoc objects and the other one is inserting additional code but maintaining the original one.

So, a solution.

This overhead can be reduced if we only instrument certain parts of the code.

But we have to have a notion of which variables could contain symbolic expressions in this case.

So we can focus only on them.

Some classes which never interact with symbolic expressions will not be instrumented.

Others can be only partially instrumented only where it's necessary.

The main goal is to reduce the work on the instrumentation part.

But in order to select which classes to instrument, we need to know which variables can contain symbolic values.

So there’s kind of a trade-off in this case.

Because we can reduce the overall overhead but at the same time, we need to perform an extra step, right, which is to perform a discovery process.

We need to analyze the variables.

Well in this paper they performed the discovery process.

Well, how did they do that? Well they did a static analysis to trace the flow of symbolic process through the program.

So they defined the starting points for each class, and identified all the variables in which the flow can get into.

Like a flow.

They didn’t explicitly explain that.

Maybe we can discuss this later.

In this paper maybe the idea is good but the approach I think has some flaws.

In the presentation we will cover, how did they transform their genius application into the upper ground.

And I think they are oversimplifying the process.

Well let’s analyze this after. No, no it’s not mentioned.

So this is a example of the intelligent instrumentation.

On this you can see the add method was instrumented, the subtract method was not instrumented. Why?

Because it was not used in the main class.

So there’s no need to instrument that.

And also in the driver class, we can see on lead why some variables aren’t instrumented.

So here is a comparison between the renewable program, the plain instrumentation and the instrumentation.

So we can see that we have to, for the intervene instrumentation, we have to instrument an S code.

But we have to perform this additional regal process.

And the third approach is no instrumentation.

So instead of changing the code, we the others, what they do is they modify the checker.

And it symbolically executes the code instead of using the instrumentation.

So here is one of the main assumptions.

So the model checker must support the ability to track which variables contain symbolic expressions.

In this case, it’s like, the model checker has to do half of the work for me.

In that case, the model checker has to support or provide a special object that we can associate to the selected variables.

In the case of the Doha, what they used after for the stacks we can associate a specific attribute.

So we can have tagging for intervariable.

This can be used easy enough.

Also semantics of the model checker must be modified.

The model checker must be fully aware of which kind of expressions are being executed.

And I analyzed.

If they are symbolic, it must ensure the resulting expression, must be handled as symbolic.

What if I’m handling concrete? I have to handle it in other way.

So this part is this.

One of the most difficult. Then the paper moved to the how to apply the exhibition on web applications.

Their main goal is to verify business logic portions of web applications.

Java based web applications.

So to achieve that they use symbolic Java PathFinder.

Some additional modifications were necessary because Java PathFinder doesn't provide all the functionalities that the authors of this paper are required.

So this is something that you maybe are related.

The first place, symbolic execution requires this closed system to run on.

We have to be able to represent this as graph form.

That web applications usually are very heterogeneous.

They contain minimals.

They are built in many languages and they contain several layers.

Each layer is specific to some functions.

So they have to perform a processing which this normal standard web applications can be transformed into java program.

But they didn't go deep, how did they do that?

They just said okay they use an approach that is based on a favor.

But they didn't provide any results and they didn't provide any comments on that.

So it seems that it was successful at first try.

So it's difficult to believe.

Okay the second issue after we transform the java application into java program is symbolic execution usually deals with numerical values.

It's the natural way as we are working with in equalities it's easy.

But in web environment when we receive an input it is handled as a string.

In web environments, when we are receiving input, it is commonly handled as a string.

For instance, when you give this: post or get, variable from forms of each values.

Also, although there are some approaches in terms of the whole traditional constraint solvers can handle the strings.

The efficiency of those is not going to do.

So, one day, it was his favor, new approach for 2009, which is, okay, we are going to represent strings as a finix, through machines.

And the universe of states that each string variable, is a universe of strings.

Also, they define a set of string constraints.

They didn't mention of all whole, did they built a set up string constraints.

This set of constraints is used to manipulate, and you use this a finite statement machine.

In terms of, the number of states that I can reach from which state I can move to the other.

Also, they combine this with the use of traditional symbolic execution equations.

In terms of some of the rules, the most important thing is, if the Finite State Machine becomes empty, which means the states are give on the refinement, the set of states becomes empty.

Or the group of the inequalities have no solution, the search on that path is over.

They finish the process one day.

And then go to the false, which is the alternative path.

So, here is the example, and you can see them in control-pro point here, from around the each statements, and you can see how this is..

This is not a good picture of you can get the idea of whole the set of the constraints. So one other main issue they have in this step is they have to modify symbolic JPF.

Symbolic JPF is a common path condition.

This was stored in numerical constraints.

But in this case, they have to augment it to give the string support.

They generate this string path condition.

I was a little bit disappointed, because I wanted to know how they did that and I just quote from paper they answered.

And they said, “Okay. New classes were implemented to store and manipulate symbolic string expressions.”

New classes were implemented.

This is all the detail that they provided.

And then, “the constraints were solved using our in-house string solver.”

And they provide some references.

So, they didn’t provide the details on this part.

Another issue that they had to cover is they have to generate an interface definite that machine solver on the JPF tool.

So, they had to re-write sort of symbolic versions of Java string, the String Buffer, and String builder.

In terms of, they could provide some support for strings, for strings, symbolic strings, but as we know, that may combine a numerical string values.

So, for instance using ValueOf, Length, or IndexOf. I can receive certain type, object, and then the output is another.

In this case, how can the numeric constraints interact with the string constraints.

The submission in this paper is, they set of rudimentary rules for commonly occurring cases.

How do they choose that? Based on the examples that they use, their obligations, they extract the frequency and generates them.

(A student askes a question)Sorry?

Yes. In the paper they show there’s an example with ValueOf.

So, for instance, if a symbolic string X start with a character this:”_”. And then you perform this:”+y” This is integral.

This is a Symbolic string X.

Then this value should have the constant value in the distance zero.

It is a negative.

This was part of the string, and constrain this.

And then when you want to represent it, and then the numerical value, you will have “-1”. Okay? (A student asks a question)

Yeah, this is also confusing for me, but I try to think of all this.

If you have this finits, some kind of finits state machine, what they want to do is, at the end we need to generate the state of constrains.

So, they define some kinds of universe of strings. Okay?

Also, they didn't provide this physically.

So, let's assume that they have 'and' strings, from which you can represent your variables.

Your symbolic variables.

For instance, if you are dealing with a numerical variables, if you get a symbolic representation, it's easy, because even if this is an integral of both, you have the idea.

Okay? I don't know the concrete value, but I know that this, the real number.

Or an integral number.

But in terms of strings, you know how this, or you know how this, right?

So, they provide this, like it's the same notion.

So, they have this, and the thing we have in this case our strings, and we have this notion of the finite state machine, what allows to in first place, how explicitly decurrent the state of this.

This is for a symbolic value.

4 strings, so this is the basic representation ok? And then they also, along this, they provide a set of constraints.

String constraint, so as you can see the finite statement in the machine it's work is to be an interface right?

From which you can track of the behavior of the relationship between the universe of strings in this case and the strings constraints.

So this what I think that they wanted to represent (student asking a question) They didn't provide any something. (Student asking a question)

Well what I understand that this is, this universe, universe of physical state that this machine can handle. Okay?

So this set along with the, with all the path tracing and the constraints.

All your feasible thoughts will start to some of them will disappear.

Some of them will be out.

So at the end you will get a final presentation of this relationship.

This is what I understand from the paper (professor talking) so they choose a five applications.

They claim that these are heterogeneous components well there were Java based and they performed this environment generation process.

So from a heterogeneous set up of applications we can identify us components and the result was standalone system, Java system they did that in a way that at the end each of this application has a driver that provides the input to more general mean application.

So in that sense they achieved the idea of a close system.

So the inputs can be a concrete or symbolic variables.

So this is like a summary of the applications that they used in this paper there are some bench marking metrics.

There are also the handling of the inputs.

The inputs were made symbolic which one were made symbolic first place they decided to do that to all the primitive types and then they match this with a set of requirements ok? In order to verify the applications.

This requirement was coded as an assertion; we usually know it’s the assertion.

And also placed in to the program.

So the requirements that they generate are part of this five categories ok?

And they provide some examples.

So in terms of the experimental procedure first the blind instrumentation there is only one path that is we have a concretable graph and we have to translate into symbolic program.

In terms of the intelligent instrumentation they use something called relevancy analysis.

So they perform some kind of point to analysis.

And they performed the model checking using the JPF.

The decision procedure was using the CVC3.

Finally and this using the FSM based symbolic string solver and third place the no instrumentation approach the modified version of JPF, ok with all the semantics modification on the liability to track the which variables can be handled as a symbolic, ok, handle symbolic values.

So this is the main result.

Here we can see the blind instrumentation or the intelligent instrumentation or the no instrumentations and the analysis of the result is based on the CPU time and the each approach achieved in terms of how they calculated the times in the first case in the blind instrumentation.

They are considering the instrumentation time, how long does it take and then the model checking time in JPF in the single case which is the intelligence instrumentation they are considering the pre analysis, relevancy analysis the instrumentation and finally the JPF.

So in terms of the results we can see here that no instrumentation is the clear winner.

The main reasons authors provide is ok there is no time needed in the pre processing so I think that's not a good answer because in some way they spend a lot of time modifying the module together.

The reduction on the number of instructions executed which is obviously because is not instrumented and as they provide their own infrastructure for handling the variables because they generate all these effects.

Constraints over all the symbolic structure are not part of the JPF JBM. So it's less expensive for the heap.

So they provide this as an argument to the results of the paper but at the same in time they know these are some drawbacks although no instrumentation was a clear winner in the first place the symbolic semantics manipulation is complex in second place they realized that there was a poor handling of concrete values by the symbolic functionality.

So in some ways they assume us null value.

They also talk about there is the probability to get a full support in this criteria because they cannot recognize very well what are they working on and the third one is related with the lack of observability that makes difficult to uncover bugs in their own infrastructure so that's what's basically all about the paper.

So now let's discuss.