Program Analysis Project 2021W

Beltrán Aceves Gil
beltran.aceves@udc.es

I. INTRODUCTION

The goal of this project is to implement dynamic backward slicing for ECMAScript 5 using Jalangi2, a framework for dynamic analysis.

II. APPROACH

To implement such program a combination of three techniques has been used:

A. Jalangi Analisis

Using *ChainedAnalyses.js*, both *SMemory.js* and the custom *Analysis.js* are combined such that Shadow Objects and Frames are available. Then, callbacks are included for conditionals, writes, reads, object property access and object property put. Each callback instantiates a node with the operation attributes, namely its type, name, value, position, line and the shadow object associated with it. When it finishes it calls the AST Handler.

B. Custom PDG Class

This is the object that contains the nodes created by the Analysis. Along with some helper functions, it provides methods to complete what Jalangi couldn't:

1. addUntrackableNodes

Loops through the AST checking if a given node type is included in the untrackable node list, in this case [BreakStatement, ContinueStatement]. If any is found, it creates a node like any other and adds it to the PDG object.

2. addSwitchCases

Much like the previous method it loops through the AST trying to find a specific node type, SwitchCase. If any is found, it creates a node and also adds the control dependencies with its contents.

3. computeDataDependencies

Here an algorithm very similar to the one provided in the slides is used, Dynamic Slice (Revised Approach). There is a node for every occurrence of a node of static PDG and the edges constitute the dynamic data flow dependencies. To assign these, it loops through node list and depending on the node type it uses one strategy or another:

- Conditional:

```
nodes =getNodesByLine(conditional.line)
for node in nodes do
    add dependency between node and conditional
end for
```

- Read:

```
nodes =getNodesByLineAndOperation(read.line, Write)
for writeNode in nodes do
    add dependency between read and writeNode
    if read.variable is ObjectReference then
        add read.name == writeNode.name to synonim table
    end if
end for
nodes =getNodesByLineAndOperation(read.line, ObjectPropertyPut)
for objectPropertyPutNode in nodes do
    add dependency between read and objectPropertyPutNode
    if read.variable is ObjectReference then
        add read.name == objectPropertyPutNode.name to synonim table
    end if
end for
nodes =getNodesByLineAndOperation(read.line, ObjectPropertyAccess)
for objectPropertyAccessNode in nodes do
    add dependency between read and objectPropertyAccessNode
    if read.variable is ObjectReference then
        add read.name == objectPropertyAccessNode.name to synonim table
    end if
end for
```

- Write:

```
nodes =getAllNodes
for node in nodes do
    if node.operation == Read and node.position > write.position then
        if node.name == write.name OR its synonims then
            add dependency between node and write
        end if
    end if
end for
```

- Object property access:

```
nodes =getNodesByLineAndOperation(ObjectPropertyAccess.line, Read)
for readNode in nodes do
    if readNode.parent == objectPropertAccess.parent then
        add dependency between read and objectPropertyAccess
    end if
end for
nodes =getNodesByLineAndOperation(ObjectPropertyAccess.line, Write)
for writeNode in nodes do
    add dependency between objectPropertyAccess and writeNode
    if writeNode.variable is ObjectReference then
        add objectPropertyAccesss.name == writeNode.name to synonim table
    end if
end for
```

- Object property put:

```
nodes =getNodesByLineAndOperation(ObjectPropertyPut.line, Read)
for readNode in nodes do
    if readNode.position > objectPropertyPut.position then
        if readNode.name == objectPropertyPut.name OR its synonims then
            add dependency between objectPropertyPut and readNode
        end if
    end if
end for
nodes =getNodesByLineAndOperation(ObjectPropertyPut.line, ObjectPropertyAccess)
for objectPropertyAccessNode in nodes do
    if objectPropertyAccessNode.position > objectPropertyPut.position then
        if objectPropertyAccessNode.parent == objectPropertyPut.parent then
            if objectPropertyAccessNode.name == objectPropertyPut.name OR its synonims then
                add dependency between objectPropertyPut and objectPropertyAccessNode
            end if
        end if
    end if
end for
```

4. computeControlDependencies

Accomplishes the same as in the previous section but for control flow dependencies. Within the node list there is not enough information to find the closures a conditional generates, so AST manipulation is used to find them.
It iterates over the AST trying to match a given AST node to a PDG conditional node. If any is found, it uses the start and end line to stablish dependencies between the conditional node and all nodes whose position is within those limits. The exception is a SwitchStatement, in which case the dependencies are between the discriminant and the SwitchCase nodes.

C. AST Handler

Once all the nodes are added, the program needs to be pruned. Everything will be deleted except:

- **Function calls and definitions**: Through a Bottom-Up approach, starting from the sclicing criterion, every function encountered gets added to the exceptions along with its defintion, if it's on the same file.
- **The dynamic slice**: Given a line it finds all slicing criterions, and for each of them explores the PDG with Breath First search, adding all found nodes to the exceptions

Finally, it walks through the AST one last time, deleting every node not contained in the exceptions.

III. RESULTS

This analysis manages to succesfully slice small programs without Class definitions, that only contain ECMAScript 5 structures and whose function calls don't modify any parameter by reference. Regarding the given scope limitations, multivariable declaration in the same line is possible and switch statements are supported.

IV. LIMITATIONS

Since this is a dynamic analysis, Break and Continue statements are a problem. They unexpectedly change the control flow o a program, generating bugs in some conditional structures. The worst case is the doWhile loop, in which if a break statement is executed during its first iteration the conditional statement will never be generated and the AST Handler will prune the node.

V. FUTURE WORK

In its current state the program is just a proof of concept. A good first step would be to do away with the intra-procedural limitation and properly track modifications by reference in function parameters.