[The Analyzers – Asha Joshi, Dibyojyoti Sanyal, Thomas Glaser]

14: Hoisting Nested Functions

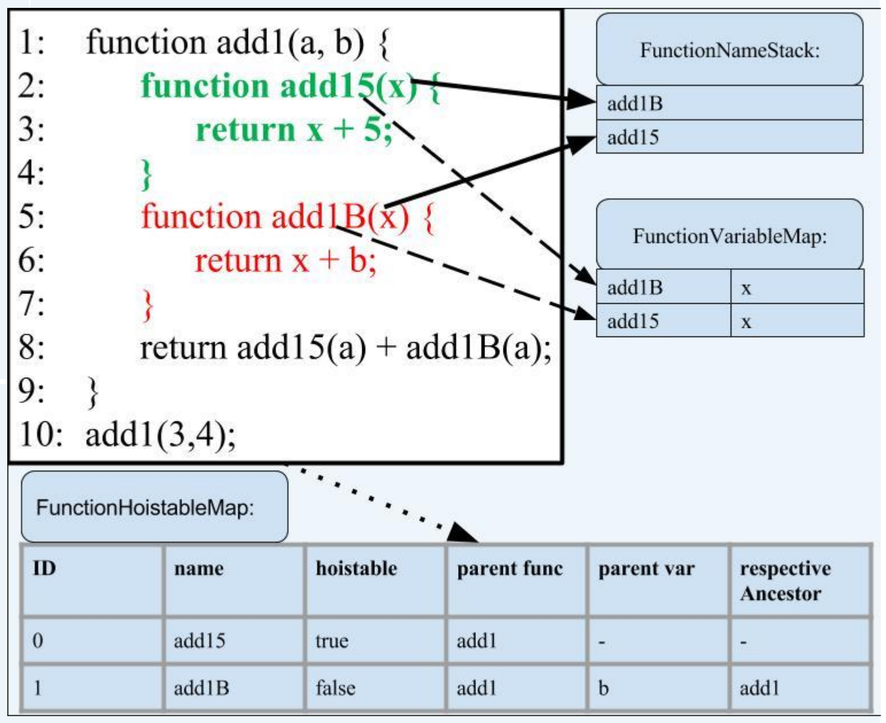
Course Project for Program Testing and Analysis

Background and aims:

The Google's V8 JavaScript engine nested version(functions inside other functions) is 42% slower than the equivalent version without this nesting. Thus we present results of a dynamic analysis aiming to detect these functions which can be hoisted.

Implementation background [jalangi2]

- Our analysis uses hooks, callback function provided by the framework Jalangi2 [5].
- Callbacks used in the analysis : *functionEnter, functionExit declare, read, write and endExecution.*
- Our analysis logic is implemented inside the callbacks.
- Jalangi generates an intermediate representation of the code on which our analysis is performed.



Implementation details :

functionEnter callback : The function name is stored in funcNameStack.

declare callback:
function declaration encountered : entry is made in funcHoistbleMap array.
variable declaration encountered: entry is made in funcVarMap array.

write callback : Same logic as *declare* declared as var v = function foo(){}.

read callback: Check the scope of function variable being read whether it is in current function or it is an outer function. funcVarMap stack is used for this check.
If an entry is found parent information, variable name, ancestor function is updated and the hoistable property is set to false in funcHoistbleMap array.

functionExit callback : {func,var} pairs from funcVarMap and function name from funcNameStack is removed. If the hoistable property of the function is not set as false in Read callback, it is set to true.

endExecution callback: nested hoistability is checked. A nested hoistability check is done to make sure if any inner function is non hoistable make the outer function containing it also non hoistabel, provided the outer function is declared inside respective Ancestor. Respective Ancestor is the function whose variable has been used in the inner function. At the end the funcHoistbleMap is used to write a report in text file in easily understandable format.

Results

The Analysis was tested on our ten scenarios an behaved like expected in all cases. It was able to find hoistable nested functions while checking dynamically whether variables are actually used. Among other collected facts we are able to suggest nested functions that can be hoisted without breaking the semantics of the original program and therefore saving execution time and memory.
Additionally we tested our analysis with three of the most depended upon npm packages [1]: *underscore* [2], *lodash* [3] and *commander* [4].
We executed the testing suites of the packages to analyze them dynamically.

Package	Total # functions	# not executed	# not hoistable	# outer most	# hoistable
underscore	24	23	0	0	1
lodash	700	296	7	0	397
commander	14	1	0	12	1

Conclusion:

We have developed a dynamic analysis which was tested on three node.js libraries. The hoistable functions detected in all there cases where checked manually in the source code and no false positives were found. Our analysis provides additional information on the functions which were not executed, cannot be hoisted and which are outer most functions.

References:

[1]: https://www.npmjs.com/browse/depended
[2]: https://www.npmjs.com/package/underscore
[3]: https://www.npmjs.com/package/lodash
[4]: https://www.npmjs.com/package/commander
[5]: https://github.com/Samsung/jalangi2