To embark on, I figured out that applying MemoIR can positively affect the program in 2 different aspects:

The first aspect is the memory optimization which was highly emphasized in the paper. The way MemoIR organizes the data objects and their fields to store them more efficiently.

This characteristic makes me wonder how MEMOIR can be applied in Real-Time systems in which efficient memory management and time predictability extremely matter.

Similar to the current MemoIR implementation, some strategies can be applied to optimize Real-Time systems' memory operations and usage following the idea of decoupling the organization and storing of data, therefore, those systems could be greatly affected in terms of performance and response time.

Besides, MemoIR can be useful to satisfy Resource Constraints due to the fact that it optimizes the memory usage and the possibility of such implementation can be assessed.

For example, in case of Embedded Systems, if static memory allocation for sensor data processing is utilized with the help of MemoIR, some unpredictable latencies associated with dynamic memory allocation will be removed and the predictability level will improve.

The other aspect is the "Bad Smell Detection" aspect. All operations done by MEMOIR not only optimizes the memory usage of the code, but they detect some bad smells such as "Unused Variables" and "Unused Fields" by applying Def-Use testing.

I believe this view can be widely extended. Moreover, Currently the operations are applied at element-level and only bad smells of that specific level are detected, however, what if the operations are applied at function-level, class-level, or inheritance-level? Below are some ways to extend MEMOIR in the aforementioned direction:

Firstly, one of the bad smells defined on the level of functions is "Unused Functions". Similar work for finding Def-Use pairs can be done to detect such functions. For example, creating Caller and Callee lists to check whether there is a caller that was never referred in the callee list. This means some function exists that was never used.

Secondly, "Field Elision" is applied for fields of objects, however, it can be applied at the level of classes as well. The same pattern would work for finding and eliminating unused class fields like looking for fields for which, there is no reference by their owner class.

Additionally, another technique can be applied to detect the bad smell "Data Clumps" in which a couple of data fields are observed together frequently in the code and they may be better to be wrapped in a class or struct. In order to discover this one, after generating the collections and creating the index-value pairs, in case the spatial locality of a couple of variables' USE's is higher than a threshold, a Data Clump has possibly occured.

However, some challenges regarding the introduction of Quality Metrics could emerge to check whether the techniques applied by MemoIR actually enhanced the code quality.

All in all, MemoIR can be developed in both directions, there are absolutely more views on extending it, however, due to my interest and experience, I delved more into possible improvements in fields of “Code Quality and Bad Smell Detection” and “Real-Time Embedded Systems’ Verification”.

Thanks for reading my review till the end.

Razieh Dorehgard