Blog on Assembly Theory

Sam Epstein

January 20, 2025

The Information Content of Assembly Traces

There's a remarkable amount of information content in assembly traces. By encoding them into directed acyclic graphs, you can tease apart what sort of information is streaming through the traces. I'm planning to write two papers: the first one will be about turning constants to variables. The second one will be about ensembles of traces. For example, take a trace that recieves a linked list of size 5 and doubles the number of every node. For the first paper, an improvement can be made such that when the corresponding SECode is compilied and a new trace comes in with a linked list of size 5, then the values in the nodes will be automatically doubled. This is an example of how you can partially recover code from traces.

7: Writeup

My original excitement is well founded. Theres a complication but the overall setup is the same. Each trace results in a constraint and if the new trace matches a constraint then the side effects can be used. I'm going to try and write it up.

6: From Constants to Functions

Currently, SEConditions and SEChanges deal with pointers or constants. However there could be a way to have the write values be a function of the read values. Thus instead of creating a const node, one has the out line as a function of the in line. Then, when it goes into a SW node, the functions are composed together. Branch operations will create constraints between the initial values. This area is very exciting! You can get every write operation to be the minimized function of every read operation. Furthermore, every branch is transformed into a precise constraint for the reads!

5: Machine Learning Component

The machine learning component wasn't addressed in my paper. The goal is to find hot paths. Technically, this means finding a memory equivalence class that is run often and has small SEConditions and SEChanges. One way to

do this is to disregard the arguments and only deal with the commands. The reason for this is one doesn't know what argument is a pointer or a constant unless on creates a reduced SEDiagram. Furthermore, its my belief that the control flow will be adequately captured if only the commands are taken account of. So one can construct a suffix tree in $O(n)$ time of a trace containing only commands. Long branches with many matches make for good candidates for hot paths.

Another method is to look at loops and for each run through the loop again take only the commands. Then put each list into a hash table. Then send paths with the highest hash count to the SEDatabase for processing.

One wants hot paths with a very high number of transient operations. One can count the number of NEW and FREE operations, stack pointer manipulations, and math operations. Hot paths with high scores make for good candidates to the SEDatabase.

4: Bounds on Combination

Good news. Let N be the number of SEConditions to be combined. Let m be the number of unique memory equivalence classes. Let s be the number of nontransient heap operations. There's a way to combine the SEConditions such that the combined graph has space $O(Ns)$ and can make matches in time $O(((\log N)m + \log s)s)$. Thus, if you have the space, you can quickly determine if there is a match. This is because the constant values can be sorted. We reiterate $O(Ns)$ is the worst case. On average, the space used is going to be much better because one can take advantage of the redundancy of the constants.

What this means is that given the best 1000 traces, their SEConditions can be combined together in a really efficient way. For example, if the input is a linked list then the memory equivalence classes group the lists by their size and then number values of the linked lists are actually sorted in the combined SEConditions construct.

3: Floats

An interesting question is how to handle floats. The first thing to note is that a float always produces a *Const* line. However the corresponding SEConditions will need work because the num values will be floats. It's unrealistic to think that two traces will have exactly the same float value. Thus it is an open question on how to modify the num values in SEConditions to handle floats. One method is to specify an average error value threshold between the num values. Another method is to have special programmer code that compiles into the SEConditions. Another idea is as follows. For example, say you produce the SECode for the top 1000 float traces. When a new trace comes in, the SEChanges of the trace with the closest float values to that of the new trace is used to compute the side-effects. Due to efficiency, you might want to test only a small fraction of the float nums.

2: Combining Conditions and Changes

I'm going to go ahead and write a followup paper entitled "Greedy Combination of Conditions and Changes in Assembly Theory". It will detail the greedy algorithm to merge the SECONDITIONS and SECHANGES constructs. This will codify what I'm talking about in post 1. I thought the idea to be relatively simple but actually there are some cases that make the endeavor quite tricky.

1: Welcome to Assembly Theory

It apppears that there's no reason why SECONDITIONS and SECHANGES can't be combined together. This means given the best thousand traces, they'll turned into a two part code of conditions and changes and then the constructs will be compressed together by their likeness, all in $O(n \log n)$ time. Each time a trace that comes in with memory-isomorphic match to one of the thousand, then its side-effects will be computed with a single table.