PhD thesis proposal
(Ka.Shrinivaasan)

Algorithms and Datastructures for Program Comprehension, Analysis and Optimization (with lowerbounds)

I would like to research on Algorithms/Datastructures and their lower bounds for some Program Comprehension, Analysis and Optimization problems. To start with I want to research on the following. There are many papers and theses already written on these which I have been reading. Hereby I have listed few points on proposed direction of research though they are in preliminary stages:

(*). Let us consider a program written in some high-level language like C/C++/Java. Many program slicers are available which would output the slices of the program. Thus a slice dependency graph can be built from the slices with the dependence relation as the edge. Each slice is a node in the slice dependence graph. Let us assign each a boolean variable s(i) to each vertex i of this Slice dependence graph. The variable s(i) takes value 1 if slice is executed , otherwise 0. Any path in Slice dependence graph corresponds to an execution path(trace). From the possible execution paths, a DNF formula can be obtained which is a disjunction of conjunction of boolean variables. This DNF corresponds to all possible slice paths a program can take. For example, DNF(f) = (s1 . s3 .s5) + (s2 . s3. s7) + ... (s3 . s6 .s2) implies the disjunctions of all possible execution paths, s1-s3-s5, s2-s3-s7, s3-s6-s2 .... Dnf(f) is satisfied if program takes one of the execution paths.Similarly a CNF(f) can also be written, which is satisified if program does not take an execution path. Counting the number of execution paths in Slice dependency graph is #P-Complete (Valiant's result).

(*) Various properties of the Slice dependency graph can be tested. For example Clique of slice dependency graph corresponds to closely related portions of the code. Vertex cover of the slice dependency graph (set of vertices with atleast one edge incident) are the slices with high impact (a summary) on the program execution. Also Feedback vertex set of the graph yields set of slices whose removal leaves the slice dependency graph without cycles. Thus Feedback vertex set of Slice dependency graph is a good measure of subset of code which has high impact.

(*) Two programs can be compared for their similarity an oft-required functionality for anti-plagiarism tools. Graph isomorphism(a renumbering of vertices) can also be applied to two Slice dependency graphs of two different programs. Comparing two codebases for similarity reduces to Graph Isomorphism problem. This involves comparing each vertex(slice) on both the graphs and to see if renumbering of vertices yields the other graph.

(*) Two programs can be compared for similarity with the use of BK-Trees which use metric spaces to return closest matching set of strings to a given query string. The metric for the metric space can be the edit distance of two Slice dependency graphs.

(*) SATURN - Does Boolean encoding of program constructs and uses SAT solver. [This project made me to think on the following - What does it mean to say that a boolean encoding of a program is satisfied. SAT is NP-Complete and Program termination is Undecidable. A program terminates only if it has a satisfying assignment and finding a satisfying assignment to Boolean encoding of a program is NP-complete, which seems to be conflicting.]

(*) Program Comprehension - using boolean encoding as in SATURN to get the summary of a program which is human readable description of what a program does. This might also add a code search engine which takes a natural language query "Does the program do X?" comprehends the program and outputs the answer for this query in natural language.

(*) Program analysis with Binary Decision Diagram - We can construct a BDD from program slices with each slice being a boolean variable in the BDD. BDD compactly represents a boolean function on slice variables s(i). s(i) is 1 if slice i is executed and 0 if it is not.This BDD compactly represents a program.

(*) Program timeout algorithm 1 - In a multiprocessing system, there often arises a need to timeout processes which consume too much of time and possibly hanging due to programming bug. So there is a need for an application level scheduler or a Timeout Manager. I have attached a document on a Timeout manager for transactions which I wrote in 2002 and did an invention disclosure in Sun Microsystems which can be modified for process timeout(Patent application has not been filed). This algorithm maintains an index which is equal to number of timeout thread executions a transaction survives and a datastructure which has both hashmap and linkedlist capabilities(it maps survival index to list of transactions with that survival index). The timeout thread decrements the survival index on each invocation through hashcode() and equals() adjustments

(*) Program timeout algorithm 2 - I also have an alternative algorithm for Timeout as follows which has complexity of O(timeout/timeout_thread_interval + max_length_of_process_list) per invocation of timeout thread. This doesnot use hashmaps but a simple vector of timeout entries as defined hereunder:

Timeout thread

```
struct timeoutEntry
{
        int timeout;
        list<process*> processes;
}
```

```
vector<timeoutEntry> timeoutList;

while true
{
        wait for timeout interval
        cached_max_timeout_entry = NULL
        for i in timeoutList
        {
                i.timeout = i.timeout - timeoutInterval
                if i.timeout <= 0
                        timeout all processes in i.processes
        }
}

When new process is created
---------------------------
if(cached_max_timeout_entry == NULL)
{
        create a new timeout entry t
        t.timeout = timeout value from config
        push the process p to t.processes
        append t to timeoutList
        cached_max_timeout_entry = t
}
else
        push process p to cached_max_timeout_entry.processes
```

References

1. SATURN project - http://saturn.stanford.edu
2. Static detection of software errors – PhD thesis by Yichen Xie (SATURN)
3. Program Analysis with BDDBDDB – PhD thesis by John Whaley
4. Various sites on Program Comprehension
5. New Slicing Algorithms for Parallelizing Single-Threaded Programs – Intel Labs
6. Computation Slicing: Techniques and Theory – Neeraj Mittal and Vijay Garg
7. Various sites on Program Slicing and Compilers