CS 639A Project Proposal

Sumit Lahiri	Group No	Amit Kumar Sharma
19111274	1	20111012

Figure 1: Is my code bloated ?

Part 1 : Problem

Software bloating is quite a common problem in any real world software project where the code base is plagued with LOCs that are not useful while the program runs in a given execution context, a common example being exposing too many **redundant APIs**, **default configurations** for each context or many **command line options** that are not used or never invoked in general but are still in the codebase. One primary reason for this is that it isn't possible to structure the code base before hand or to choose a strict design pattern for all components that we write. Many times code needs to be written on demand or ad-hoc basis because not all requirements are captured in the early stages of the project and that becomes the potential root cause for **software bloating**. **Software bloating** can lead to bugs, slower code execution and even expose vulnerabilities in the code base. The current techniques used are **manually thought** clever **metrics & heuristics** for identifying such **bloat sites** and refactoring the code to remove them.

Part 2 : Objective & Proposal

The project we are undertaking is more aligned towards **program analysis** . Below we briefly explain our objective. We propose to debloat a piece of **C or C++ Code** via **Policy Based Reinforcement Learning** using techniques and approaches as laid out in the two research papers we have selected for accomplishing the task. Both the papers have demonstrated novel techniques for debloating the codebase for a given **C & C++ Software** using **Policy Based Reinforcement Learning**, the first paper mentioned does this by **Learning-Guided Delta Debugging** while the second paper does this by **Guided Function Specialization** technique. In our project (based out of Paper 2), we model the debloating problem as **reinforcement learning** where action would be to specialize a program or not. The state of the model will be a **vector** containing all the required details about the **specialized code**. The **reward** will be a **reduction** of the number of **instructions or code size** in the program after the program is debloated. As the model keeps learning, we expect the rewards to be improved with more inputs.

Part 3 : Procedure

We will initially focus on the implementation, analysis and comparison of both the techniques as mentioned above and then move on to finding new sites for debloating or in figuring a better policy for the **RL model** we will build. The tentative procedure for our project submission is as outlined below (**RL** means **Reinforcement Learning** henceforth):

Stage 1 : Pre RL Stage :: Week 1 & 2

- Start with **LLVM IR** of the code and use **Inst2Vec** as outlined in Paper 2 to extract feature information as outlined in the paper before **RL stage**
- Finding more sites other than the ones listed for **Program Specialization** in Paper 2.
- Understanding **Chisel** & **Trimmer** tool as implemented in Paper 1 in terms of how **markov decision process** enhances **delta debugging**. This steps helps in **Stage 2**.
- Collect and prepare bloated code samples for analysis, this is the input to our tool infrastructure. We will explore the use of fuzzing here.

Stage 2 : RL Implement Stage :: Week 3 & 4

- Specification for debloating using **Chisel** Tool, which essentially becomes a policy for the **RL Model** to learn.
- Implementing **DeepOCCAM** tool from **OCCAM** Tool and run or analysis.
- Exploring the use of other off-the-shelf decision tree based model other than **FastDT** in **Chisel Tool**
- Compare performance with **Chisel** tool of Paper 1 based on the metrics we choose before hand in **Stage 1**.
- Seeing if the metrics we choose for **Program Specialization** in **Stage 1** for **DeepOCCAM** are any good versus the specification for debloating in **Chisel Tool**.

Stage 3 : Analysis & Wrap Up :: Week 5

- Debloating can introduce new errors in the code, we write tests and run samples to check if no un-intended behavior is introduced.
- We produce the results of our analysis in a systematic way and show it using a web-based interface that connects to our tool communicating over **gRPC** or as a **REST API**
- Finish writing the **report** and **presentation** for the **Final Review**.
- We demonstrate a **run** of our tool to report **new sites** for better debloating.
- Report the new **heuristics** and **approaches** we adopted to accomplish the same.

Deliverables

- A complete **comparison report** & **project report**.
- **Final Slides** for **review** & **presentation** purpose.
- **Code Repository** of our tool. (Would add a docker image for ad-hoc running).
- **Sample code** files on which we ran out tools for testing and implementation purpose.
- Report on new **heuristics**, **approaches** and **sites** for software debloating.
- **Extra Task :** Explore the use of other **machine learning** techniques or approaches for better software debloat.

- **Extra Task :** Run an LLVM pass which would do **debloating** using our tool. We wont commit it or share with LLVM repository but just integrate and run our implementation as an LLVM Pass

Paper 1

- Kihong Heo, Woosuk Lee, Pardis Pashakhanloo, and Mayur Naik. 2018. Effective Program Debloating via Reinforcement Learning. In 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada. ACM, New York, NY, USA, 15 pages. Link : https://dl.acm.org/doi/10.1145/3243734.3243838

Paper 2

- Le Van, Nham, Ashish Gehani, Arie Gurfinkel, Susmit Jha, and Jorge A. Navas. "Reinforcement Learning Guided Software Debloating.", NIPS 2019. Link : http://www.csl.sri.com/users/gehani/papers/MLSys-2019.DeepOCCAM.pdf

Tools

We may add a few tools later on.

- Chisel Tool : https://github.com/aspire-project/chisel
- OCCAM Tool : https://github.com/SRI-CSL/OCCAM
- PyTorch : https://pytorch.org/
- Inst2Vec : NCC Tool https://github.com/lahiri-phdworks/ncc
- LLVM : https://github.com/lahiri-phdworks/llvm-project
- OpenAI Gym : https://github.com/lahiri-phdworks/gym
- gRPC-go Tool : https://github.com/codersguild/grpc-go
- Kibana Dashboard : https://www.elastic.co/downloads/kibana
- American Fuzzy Loop : https://github.com/google/AFL

Terms

We have already done a preliminary analysis of the task at hand and also of both papers we shared here in the proposal. Our best efforts would be to deliver all the things listed but we may get stuck at stages where we can't proceed further. In such instances we shall seek help from your side on implementation or further guidance, which will be appropriately mentioned in the report as well. Thanking you for giving us the opportunity to do this project.

References & Study

- What is Software Bloat? : https://en.wikipedia.org/wiki/Software_bloat
- Chisel ACM Presentation : https://www.youtube.com/watch?v=8eRZKoLFakw
- IR 2 VEC : https://arxiv.org/pdf/1909.06228.pdf
- Neural Code Comprehension : http://www.cs.columbia.edu/~suman/gabe_slides.pdf
- Neural Code Comprehension: A Learnable Representation of Code Semantics https://www.youtube.com/watch?v=8aXl53pGflU
- How LLVM Optimizes a Function : https://blog.regehr.org/archives/1603
- On Software Disenchantment : https://tonsky.me/blog/disenchantment/