CS 639A Project Proposal

Sumit Lahiri	Group No	Amit Sharma
19111274	1	20111012

Figure 1: Is my code bloated ?

Part 1 : The 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 in 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 codebase 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 per se 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 : The Proposal

We propose to debloat a piece of **C & C++ Code** using **Policy Based Reinforcement Learning** using techniques and approaches as shared in the two research papers we have selected for accomplishing the task. Both the papers have demonstrated novel techniques for finding potential sites in 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.

Part 3 : Tools and 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

- 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**
- Start with **LLVM IR** of the code and come up with a specification for **debloating** for **Paper 1**
- Finding more sites other than the ones listed for **Program Specialization** in **Paper 2**.
- Understanding **Chisel** & **Trimmer** Tool as implemented in **Paper 1**.

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

- Finding good code samples and repositories with bloated code We have already identified a few other than what is in the paper .
- Implementing **DeepOCCAM** tool from **OCCAM** Tool and run or analysis.
- Compare performance with **Chisel** tool of **Paper 1** based on the metrics we choose before hand in **Stage 1**. We need to implement **Trimmer** and a few other tools by ourselves, although we don't rule out using off-the-shelve tools
- Seeing if the metrics we choose for **Program Specialization** in **Stage 1** are good w.r.t to **Chisel Tool**.

Stage 3 : Analysis & Wrap Up :: Week 4 & 5

- We produce the results in a good and systematic way, probably using a web-based interface to use our tool communicating over **gRPC**
- Finish writing the report and presentation for the **Final Review**.
- We demonstrate a **run** of the tool, to what ever level we finish it and show new sites for better program debloating.
- Report the new **heuristics** and **approaches** we adopted to accomplish the same.

Deliverable

- A complete **comparision report** and **project report**.
- Final Presentation for **Review** and delivery purpose.
- **Code Repository** & **Sample codebases** on which we ran out tools.
- **Extra Task :** Run an LLVM pass which would do **debloating** using our tool. We wont commit it or share with LLVM Repo but just to run an LLVM Pass which in our opinion is a cool pass to do !

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.

Paper 2

- Le Van, Nham, Ashish Gehani, Arie Gurfinkel, Susmit Jha, and Jorge A. Navas. "Reinforcement Learning Guided Software Debloating.", NIPS 2019

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

Preliminaries

We have already done a preliminary analysis of the task at hand and also of both papers we shared here in the proposal.

Terms

Our best efforts would be to deliver all the things listed but we may get stuck at stages where we can't proceed further or trace back to change the proposal here. Any appropriate action on your part can be exercised in such scenarios. Instances where we can seek help from your side on implementation or further guidance will be appropriately mentioned in the report as well. Thanking You for giving us the opportunity for doing 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
- Bloated Software : https://hackaday.com/2018/09/22/one-mans-disenchantment-with-the-world-of-software/