CS 639A Progress Report 1

Sumit Lahiri	Group No	Amit Kumar Sharma
19111274	1	20111012

First & Second Sprint : Brief Progress Report

We started working on the **debloating project** in **three splits** for our **Pre-RL Stage Work**. The most **time-consuming** part was to migrate the outdated packages used in some the tools chosen by us for delivering the project. We gained a fair idea on how **embedding spaces** are used and the novelty in **inst2vec** tool as modelling an LLVM IR as **XFG** & mapping to an **NLP** like **skip-gram** model. We started running the **inst2vec** tool to generate an embedding for our debloating `C++` sample space, like **call-site** information, **function arguments**, **type parameters** etc as described in **Paper 2**. We are reporting the highlights from sprint of **Commit-1 01/11/2020** to **Commit-22 09/11/2020** which is **Week-1 : November 2020**.

Proposed Stage 1 Milestones : 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.

We list below the progress on the above points that we have been able to achieve so far.

Stage 1 : Split 1

- Completed the setup for LLVM-IR generation, clang tool, **inst2vec** tool.
- Completed the setup for **OCCAM** tool, **Trimmer** tool & **Chisel** tool. All setup & build related issues were **resolved** and **ran** once one sample example.
- We are in the middle of running the **inst2vec** tool and using the **public data-sets** available to train it.
- We are using the pre-trained embeddings from **inst2vec** repository for the starting point for the **DeepOCCAM** tool we are developing.

Stage 1 : Split 2

- We found **2** more features where **program debloating** may be possible.

- We understood how to write specification for the script used by **Chisel** tool as in Paper 1 in terms of the three parts of the scripts namely `compile()`, `desired()` & `undesired()`.
- **Chisel** tool works by learning a **policy** for **delta debugging** by **reinforcement learning** which guarantees **1-minimal P*** & $O(|P|^2)$ runtime. The abstraction is that a **markov decision process** is being used to model the **reinforcement learning** problem for meaningful **guidance** to learn the **policy** in a better way.
- We spend time understanding how **OCCAM** tool, **Trimmer** tool & **Chisel** tool are working.

Stage 1 : Split 3

- Bloated Sample collection and preparation started, We are targetting bitcoin1-src & eth-solidity. Past experience tells that the code is most probably **bloated**. We may have to change it later to publicly available samples. **[This needs some more work]**.
- We found this **video** useful to understand how **program debloating** can be improved using **Stochastic Optimization**, which we intend to explore as **extra work** if time permits.

We are almost **ready** to move to the next **Stage 2** of modelling and developing the **reinforcement learning model** using **PyTorch**. We intend to wrap up an left over tasks from **Stage 1** and move on to working on **Stage 2** in the next couple of days.

Our first task now will be to start pre-processing the **embedding output** for **features vector** extraction & develop the **reinforcement learning** model hand-in-hand so that we can start analysis and see some debloated samples as a part of the **Stage 2** task. We have resorted to a **pipe-and-filter** like approach in-order to complete the project. In-sense, we take in **bloated C++** programs and produce **debloated C++** code, processing **stage-by-stage** where intermediate from the previous step moves to the next step for more processing.

In the **Stage 2** report, we shall describe the full architecture and modelling formulation we used for **DeepOCCAM** and show some results on how it works currently. In **Stage 3** we do the full comparison and share a complete project report with **targetted features** we used and how it fairs against **Chisel** tool. For **Stage 1**, the setup and running the tools took more than expected time. We are maintaining our progress on **GitHub** as a **private repository**.