Project 4 Intra-procedual Data-Flow Analysis

This document describes the content of project 4, all theory needed can be found in Compiler textbook **Chapter 10 & 17**.

Goals

In this project, you are provided with a simple Android apk file (`test.apk`), and a framework based on `dexlib2` to do some simple static analysis. More specifically, there are two goals in this project:

- Firstly you should generate a control flow graph (CFG@wiki) whose nodes are basic blocks(basic block@wiki, also Textbook Page 394) for the given `test.apk` sample.
- Secondly, do some dataflow analysis (data-flow analysis@wiki) based on the above CFG you constructed. You only need to consider intra-procedual problems (i.e. for a single method) in this step and generate a data dependency graph using the tools we provided.

Details

Files released:

```
project_4
    - README.pdf: this file
    - lib: all the libs needed
    - src
        - java
            - Main: main method
            - compiler: support code and your code here
        - resources
            - java_source_code: source code of test.apk
            - test.apk: target apk that we are going to analysis
```

Steps:

1. **Building**
   Download released files and extract to your workspace, then use gradle to build the project:
   For Intellij Idea users: use `gradle idea` to build the project;
   For Eclipse users: use `gradle eclipse` to build the project.
   Then you can import this to your prefered IDE.

If you do not want to use gradle, just import this project using our source, and all the libraries can be found in `lib` directory. But in this case, if you use any other libraries, please notice TA in your document.

2. Generally, to do a data-flow analysis on an APK file you should do the follow steps:

    1. Get all dex bytecode from the apk file. This has been done in our support code, so you can use `InstructionFetcher.fetch()` method to get all the dex bytecode. (see our example in `Main` class)

    2. Combine structions to basic blocks according to the definitions. (You are not requried to consider exception blocks here)
    Tips: In this step you may reference Dex Format from Google official documents to know the meaning of each instruction. And you may use `DexBackedInstruction` in dexlib2 to do operations on these instructions.

    3. Using basic blocks as nodes to construct the Control Flow Graph (CFG), and generate a **`cfg.dot`** using `GraphWriter.print()` in our support code. (see our example in `Main` class)
    Tips: You can refer `dot` part below to see more usage of dot.

    4. Do intra-procedural data-flow analysis on the CFG above and construct a data dependency graph, generate a **`dpg.dot`** . You may refer to textbook for knowledge of `def-use` and `gen-kill` .
    We provide a `part.png` in our release files to show a sample of data dependency graph of the target method.

Tools

dexlib2

Android apps are written in Java language (we do not consider NDK here) and they are compiled into dalvik bytecode (called dex). Our support code use **dexlib2** to fetch dex from apk file.

dot

dot is a plain text graph dexcription language, see dot@wiki for more details. You can use visualization tools, such as Graphviz to get PNG format of .dot files. For example, after installing Graphviz, you can use the following command: `dot -Tpng sample.dot -o sample.png` .

Deadline

This project is due on **23:59, 22th Jan, 2016**.
Please submit your answers to our FTP server before due.

Submission

Please put all your source code and document into a directory named after your student number, for example:

```
12302010001
    - src
    - cfg.dot
    - dpg.dot
    - document.txt
    - build.gradle
```

Then compress using the follow command

```
tar zcvf 12302010001.tar.gz 12302010001
```

and upload to our FTP server before due.

Help

- Textbook, wiki and **Google** are good places to answer your questions.
- Also, you can contact TA: Zhang Xiaohan `<xh_zhang@fudan.edu.cn>`, Room 310 Software Building.

Good luck and start early!