实验一：基于 JoeQ 的数据流分析

JoeQ 是一个三地址码版本的 Java Bytecode 变种；原始的 Java Bytecode 是基于栈操作的。JoeQ 支持库能够将 Java Class 反汇编并翻译为 JoeQ 格式的中间码，用于分析。

介绍

本次实验的主要内容是实现一个对 JoeQ 中间码进行数据流分析的框架。基础的接口和设施已经提供。你需要实现的部分包括：

1. 基于迭代的数据流分析求解器。它需要能够接受各种满足我们提供的接口的特定类型的分析算法。
2. Reaching Definitions 和 Faint Variables 两种数据流分析算法。它们应基于你实现的迭代求解器运行。

本次实验语言为 Java 5，环境为 JDK SE 5。更高版本的 JDK 将无法正确执行 joeq.jar 中的代码。

本课程的实验会为同学们提供实验机并为每个同学提供单独的用户，其部署在助教本人的 PC 的 Hyper-V 虚拟机上（方便我搞事），轻点折腾。

代码结构

提供的程序包中包含以下内容：

```
lib/joeq.jar packaged JoeQ
src/examples Examples of using JoeQ
src/flow     The dataflow framework and interfaces
src/submit   The code you will submit (Only make modifications here!)
src/test     Tests for your solutions
.idea/       Intellij IDEA project file for ease of development, with
             various run configurations for testing
Makefile     Makefile for compiling your code
run.sh       Executor script for running your code
```

如何运行

实验机上 `~/exp1` 中包含了本次实验中需要的所有内容。

本次实验中提供了两种测试运行方式。

使用 Intellij IDEA

推荐同学们在进行实验时使用 Intellij IDEA。使用它你可以方便地查看 JoeQ 的接口实现、快捷地进行调试，享受 IDE 带来的便利。

本地开发时，你将需要自己安装 JDK 1.5。由于这一过老版本较难获得（在 Oracle 的 archive 中，需要注册 Oracle ID 才能下载），我在 `/opt/` 下预先准备好了 Windows 和 Linux 的 64 位版本安装程序，同学们可以通过 SFTP 下载并安装之。

将 `~/exp1` 通过 SFTP 下载到本地，作为 Intellij IDEA 项目导入后，在项目配置中指定 JDK 1.5 的位置（如果没有自动配置好）后，你就可以通过右上角的 configurations 选项栏选中需要运行的配置，运行或调试。

各配置的功能如下：

```
PrintQuads {Test}      输出类 {Test} 的 JoeQ 中间码
Flow {Analysis} {Test} 对类 {Test} 基于 MySolver 进行数据流分析 {Analysis}
```

在你完成 `submit.MySolver` 的实现前，后者不会输出有实际意义的内容。

使用 `Makefile` 和 `run.sh`

推荐同学们在本地使用 IDE 完成后，将 `~/exp1/src/submit` 目录上传更新（当然，是覆盖原文件）后在实验机上再次测试运行（这将是评分时的测试环境）。你也可以直接在测试机上进行开发，有 Vim 可用，不过你们的助教并没有进行任何的配置；有经验的同学可以自行操作 `~/.vimrc`，没有经验的同学个人不建议这么玩。

为编译项目，使用：

```
make
```

对应于 Intellij IDEA 中的配置 `PrintQuads *`，运行

```
./run.sh examples.PrintQuads examples.ExprTest
```

其中 `examples.ExprTest` 可以替换为任意测试代码类。

对应于 Intellij IDEA 中的配置 `Flow *`，运行

```
./run.sh flow.Flow submit.MySolver flow.ConstantProp test.Test
```

其中 `flow.ConstantProp` 可以替换为任意继承自 `flow.Flow.Analysis` 的类；`examples.ExprTest` 可以替换为任意测试代码类。

其它本地工作方式

如果你不希望使用 Intellij IDEA，例如你是忠实的 Eclipse 信徒，那么我想如何根据 `Makefile` 和 `run.sh` 构造一个 Eclipse 项目对你来说并不是一件难事 :P

如果你是纯终端选手，那么你需要适当调整 `Makefile` 和 `run.sh` 内容，将其中的 `javac` 和 `java` 所在路径修改为你本地的配置。

实验内容

本节中包含了本次实验全部你需要工作的相关内容。

你需要完成的部分全部位于 `src/submit` 中，也即 `package submit`。

1. 完成 `submit.MySolver`。它位于 `src/submit/MySolver.java`。使用已经提供好的 `flow.ConstantProp` 和 `flow.Liveness` 作为在这一 Solver 上运行的分析算法来测试其正确性；期望的输出分别位于 `test/Test*.cp.out` 和 `test/Test*.lv.out`。
2. 完成 `submit.ReachingDefs`。它位于 `src/submit/ReachingDefs.java`。在 `submit.MySolver` 上运行时它应输出和 `test/Test*.rd.out` 相同的正确结果。

3. 完成 `submit.Faintness` 以使其正确运行 Faint Variable Analysis。它位于 `src/submit/Faintness.java`。
4. 在 `submit.TestFaintness` 中添加用于测试 Faint Variable Analysis 的代码。在每个方法的注释中注明该方法中所有的 faint variable 及原因。`test/TestFaintness.out` 中提供了初始的测试代码的正确输出，但你应当继续添加新的测试代码以覆盖尽可能多的情形。

Faintness Analysis

Faintness Analysis 本身是 Living Variable Analysis 的进一步扩展。faint variable 的定义是：非活跃（dead）的或只用于计算 faint variable 的变量。举例来说：

```
int foo()
{
    int x = 1;
    int y = x + 2;
    int z = x + y;
    return x;
}
```

这一函数中，`x` 被用于返回值，所以是活跃变量；`y` 虽然被用于 `z` 的计算，但 `z` 是非活跃变量，因此 `y` 是 faint 的。

在 JoeQ 中，为了简化情形，我们只对 `Operator.Move` 和 `Operator.Binary` 两类指令做 faintness 传递。也即，如果有

```
ADD_I    T12 int, R4 int, R5 int
```

这里 `ADD_I` 是一个二元运算。不同于 Living Variable Analysis 中直接将 R4 和 R5 视作活跃变量，这里我们认为如果 T12 是不活跃的，则 R4 和 R5 也不应被视作活跃。

除了这两类操作符之外的指令使用的变量，我们认为被用于了不明用途（例如函数调用，包括输出），即使没有用到其输出，也可能有其它不可忽视的副作用，不可消去，应当是活跃的。

为了实现这一分析，你可以从已经提供的 `flow.Liveness` 出发，思考进行哪些修改。

提交方式

直接在实验机上对 `~/exp1/src/submit/` 进行修改即可。确保在实验机上它能够正确地工作。

实验机运行 CentOS 7.5，架设在助教的 i7-8700K PC 上的 Hyper-V 虚拟机上，最大内存 2048 MB，2 处理器核心可用（如果发现性能不足请及时联系助教处理）。在校园网内以你的学号为用户名使用 SSH 连接实验机：

```
ssh YOUR_STUDENT_NUMBER@166.111.68.163 -p 2273
```

> Beginner's Hint：如果你使用的是 Windows 10 build 1803 及更新版本、任意版本的 Linux 或我也不知道什么版本的 MacOS，你应该（说应该是因为我对 mACos 一无所知）可以在系统默认的终端中直接使用 `ssh` 命令；如果你使用的是 Windows 10 build 1709，参考这篇 blog 启用 OpenSSH Client；如果是更早版本的 Windows 或你不喜欢使用 Windows 自带的 console host，推荐安装 XShell 或 MobaXTerm 等 SSH 终端工具。

各位的用户并没有设置密码，请在登入后第一时间使用如下命令修改密码：

```
passwd
```

如果在你修改密码前就有坏人登入了你的用户身份、修改了密码，请联系助教重置你的用户。

> 如果有经验，你也可以为你的账户设置一个超级恶心的密码然后使用 ssh key 登入。没有经验的同学也可以了解学习一下 ;)
>
> 同时，觉得每次输入那么一串东西太麻烦的话，可以创建一个文件（如果没有）`~/.ssh/config`，在其中添加：
>
> ```
> Host WHATEVERNAMEYOUWANT
>     HostName 166.111.68.163
>     Port 2273
>     User YOUR_STUDENT_NUMBER
> ```
>
> 并通过
>
> ```
> ssh WHATEVERNAMEYOUWANT
> ```
>
> 代替先前的命令连接。

需要从实验机上下载文件/文件夹时，使用 SFTP：

```
$ sftp -P 2273 YOUR_STUDENT_NUMBER@166.111.68.163
Connected to *******.
sftp> ls /opt
/opt/jdk-1_5_0_22-linux-amd64.bin        /opt/jdk-1_5_0_22-windows-amd64.exe
/opt/jdk1.5.0_22
sftp> get /opt/jdk-1_5_0_22-linux-amd64.bin
Fetching /opt/jdk-1_5_0_22-linux-amd64.bin to jdk-1_5_0_22-linux-amd64.bin
/opt/jdk-1_5_0_22-linux-amd64.bin    100%   42MB  42.1MB/s   00:00
sftp> get -r exp1 exp1
( ... downloading directory exp1 ... omitted ... )
```

> 助教会每天对虚拟机进行 Checkpoint，请不要妄图搞事，谢谢。

Hints

> 如果你十分自信，可以不看这一节做本实验，复现你们助教踩过的所有坑 :P 不算很深

1. 这一代码框架中，数据流分析的基本单位选取了 Quads 而非 BasicBlock。这一设计存在一定缺陷，但不影响基本的使用。主要的问题会是一个过程/函数/方法的结束 Quads 不只有一个，这与 BasicBlock 组成的 CFG 中始终设置一个 Exit Block 不同；为了解决这个问题，你需要想办法找出所有会使函数退出的 Quads，例如从 Exit Block（`ControlFlowGraph.exit()`）开始递归地跳过空 BasicBlock 直到找到所有连接 Exit Block 的 Quads，或遍历所有 Quads 并检查其 successor（`Quads.successor()`）是否包含 `null`（这表示其后继有 Exit Block）。

2. `Flow.DataflowObject` 是用来表示状态的。也就是说，它是 IN 和 OUT 集合的类型。参考已有的 Analysis 实现，观察其正确使用方式。需要注意的是，它需要实现 `equal` 方法以保证你能够正确地比较一个状态是否发生了变化（从而在正确的时机结束迭代算法）。
3. `Flow.Analysis` 的食用方法：
    1. 在 `isForward()` 中设定好方向。
    2. 当想要对一个 Quad 根据其 IN/OUT 计算 OUT/IN 时，使用 `setIn` 或 `setOut` 接口设置好输入状态，调用 `processQuad` 接口完成处理，然后就可以通过 `getOut` 或 `getIn` 接口获取输出状态。注意，计算完成的状态会保存在 `analysis` 对象中。
    3. Entry 状态对应于入口点 Quad 的 IN，Exit 状态对应于所有出口点 Quads 的 OUT。在处理前，你需要根据方向从 Entry 或 Exit 初始化入口或出口点的 IN 或 OUT，处理完后也需要相应地根据出口点或入口点对 Entry 和 Exit 赋值。判断入口/出口点的方法依然是检查 `Quads.successor()` / `Quads.predecessor()` 是否包含 `null`。
4. 在遍历 Quads 时，善用 `QuadIterator`。构造 `QuadIterator` 时，你不仅可以提供输入的 CFG，还能够指定遍历方向：`true` 代表从 Entry 到 Exit，`false` 反之。正向遍历时使用 `next()` 移动到下一 Quad，反向时使用 `previous()` 移动到前一 Quad；事实上，方向参数只影响初始化位置在头还是尾。
5. 实现 Faintness Analysis 时，善用 `QuadVisitor`。你可以参照 `flow.ConstantProp` 中是如何分类处理不同的指令的。

做不下去了的时候翻一翻 `joeq.jar` 中的 Javadoc，会有帮助的。