SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Sebastian Hack
Johannes Doerfert, B.Sc.

Compiler Construction WS15/16

Exercise Sheet 8

Exercise 8.1. Static Single Assignment Form

- Which two criteria have to be fulfilled to assume that a given program is in SSA form?
- Transform the following program to SSA form.

```
x1 = 10;
x2 = 1;
while (x1 > 0) {
  x2 = x2 * 2;
  x1 = x1 - 1;
}
x2 = x2 + 3;
```

Exercise 8.2. Interference Graphs

1. Consider the following code snippet.

```
s1 = 47;
s2 = 42;
s3 = s1 + s2;
do {
    s3 = s3 - s1;
    s4 = s3 + 2;
} while (s3 > s2);
s5 = s2 * s4;
s6 = s3 / s2;
write s6 - s5;
```

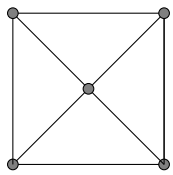
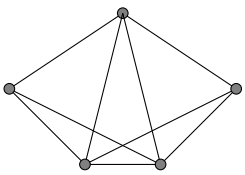
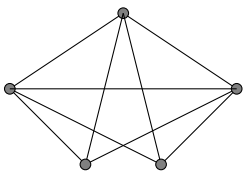
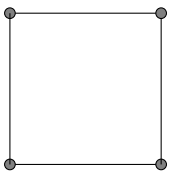
   a) Draw the interference graph.

   b) Assign actual registers to the symbolic registers by coloring the interference graph using Chaitin's local-colorability criterion (degree of node is smaller than k). Assume an overall number of 4 registers to be available.

2. Craft an algorithm to test interference of two definitions in an SSA-form program without constructing the interference graph. You can assume that the dominator tree is given and that the position in the dominator tree of each definition and use can be looked up. Additionally, you can assume that you have a list of all uses of every variable.

Exercise 8.3. Chordal Graphs

Are the following graphs chordal or not? Justify your claims!

Exercise 8.4. Colorability

Show that Chaitin's algorithm finds a k-coloring for every k-colorable chordal graph.
Hint: Every chordal graph with at least 2 nodes has 2 simplicial nodes.

Project task F. Intermediate Representation

- Install LLVM 3.7. Get it from your package manager or `http://llvm.org/releases/download.html#3.7`.
- Merge the master branch of the project *template* to get an updated Makefile to build the project with LLVM:

```
cd c4
git fetch --all
git merge template/master
```

- Download and study the provided example programs to gain some intuition about the LLVM API. The examples show how to directly construct LLVM IR representation for small but relevant code fragments.
- Implement a systematic LLVM IR construction from your AST. The compiler must only perform this, if `-compile` or no explicit compilation mode is given.
- The output shall be human readable LLVM code into a file. When the input is `foo/bar.c` the output file is `bar.ll` in the current directory. To output into a file perform the following:

```
#include "llvm/Support/FileSystem.h"
#include "llvm/Support/raw_ostream.h"
...
std::error_code EC;
raw_fd_ostream stream(filename, EC, llvm::sys::fs::OpenFlags::F_Text);
M.print(stream, nullptr); /* M is a llvm::Module */
```