Exercise Sheet Solutions 9

Compiler Design Lab
Prof. Dr. Sebastian Hack
Fabian Ritter, B.Sc.

This is a solution proposal. It does not necessarily describe complete solutions but gives the initial ideas to solve the theoretical exercises and presents the results.

Exercise 9.1 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;
```

Solution:

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

  Every variable has exactly one point where it is defined.

  Furthermore for every use of a variable x at program position ℓ the following has to hold:

  If it is used as the i^{th} position within a ϕ, then the i^{th} predecessor block of ℓ has to be dominated by the definition of x.

  If it is used outside of a ϕ, then the definition of x has to dominate ℓ on all incoming paths.

- We get the following program in SSA form:

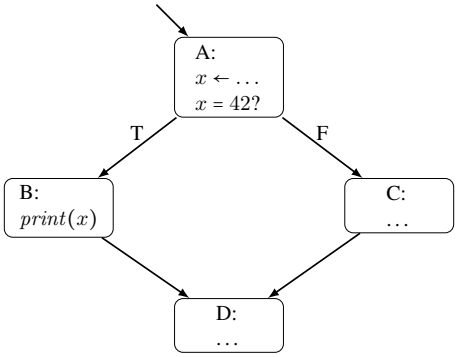
```
// first BB
x1,1 = 10;
x2,1 = 1;

// condition BB
loop_cond:
x1,2 = φ(x1,1, x1,3);
x2,2 = φ(x2,1, x2,3);
if (x1,2 > 0) {

  // loop body BB
  x2,3 = x2,2 * 2;
  x1,3 = x1,2 - 1;
  jump loop_cond;

}

// after loop BB
x2,4 = x2,2 + 3;
```

Exercise 9.2 SSA Form and Flow-Sensitivity

A commonly encountered argument for using the SSA form for program analysis is that analyses on SSA programs need not be flow-sensitive (i.e. they do not need to have a mapping from program locations to some abstract states) to achieve the same precision as flow-sensitive analyses.

Show that this argument is wrong by giving an example program in SSA form and a program analysis that gives better results on this input if performed flow-sensitively.
Hint: Which operations can influence the analysis information for a variable x other than an assignment to x?

Solution:



With a flow-sensitive constant propagation, we can find out that x has the value 42 at basic block B. In a flow-insensitive setting, the information $x \mapsto 42$ would clearly be wrong, as x can have different values at other basic blocks.

Exercise 9.3 Interference Graphs

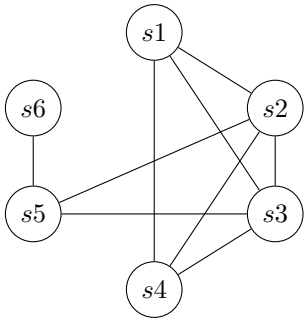
Consider the following code snippet.

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

1. Draw the interference graph.
2. Assign actual registers to the symbolic registers by coloring the interference graph using Chaitin's local-colorability criterion. Assume an overall number of 4 registers to be available.

Solution:

1. We construct the following interference graph:

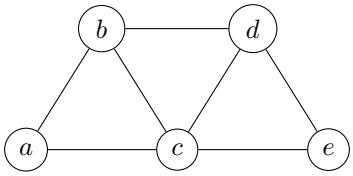


2. As the interference graph contains a clique of size 4, we need at least 4 colors to color the graph. As the local colorability criterion is $degree(n) < k$ we can remove the nodes from the graph in any order that only removes one of the nodes S2 or S3 after having removed at least one of its neighbors first.

   For the sake of simplicity we use the order S6, S5, S4, S3, S2, S1. Inserting and coloring them is then done in the reverse order and we assign the registers R1, R2, R3, R4, R1, R2.

Exercise 9.4 More Interference Graphs

Consider the following interference graph.



Give an SSA form program without control flow that has the above graph as an interference graph. You may use assignments of constants to variables, assignments of results of binary arithmetic operations to variables, and stores of variable values to some (fixed) memory address in your program.

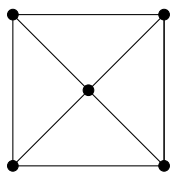
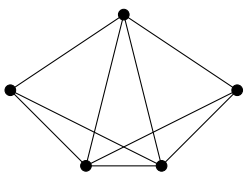
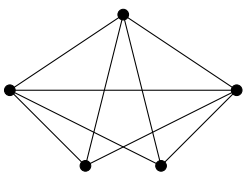
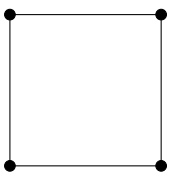
Solution:

The following program is in SSA form (each variable is defined once and each use is dominated by its definition) and has the given interference graph:

```
a = 1
b = 1
c = 1
store(a)
d = 1
store(b)
e = 1
store(c)
store(d)
store(e)
```

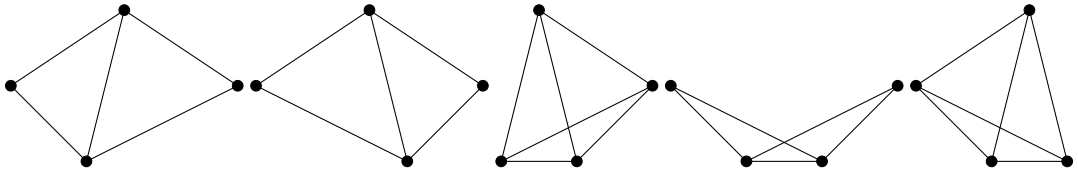
Exercise 9.5 Chordal Graphs

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

Solution:

1. Not chordal: The subgraph that is induced by the four outermost nodes is a cycle of length 4.
2. Chordal. We show that it is chordal by looking at all induced subgraphs of length ≥ 4 and checking whether any of them is a cycle (this method obviously does not scale well to larger graphs).
   - The full graph is the only induced subgraph of length ≥ 5. It is not a cycle as it contains chords.
   - The following induced subgraphs of length 4 exist:

     All of them contain chords and are therefore no cycles.

   As we enumerated all potential candidates for induced cycles of length ≥ 4 and verified that they are in fact no cycles, we have shown that there is no such induced cycle, therefore the graph is not chordal.
3. Chordal: It is the same graph as the previous one (just differently laid out).
4. Not chordal: The graph itself is a cycle of length 4 (i.e. it is an induced cycle that is not a triangle).