SAARLAND UNIVERSITY
COMPUTER SCIENCE

Compiler Construction (WS 2017/18)

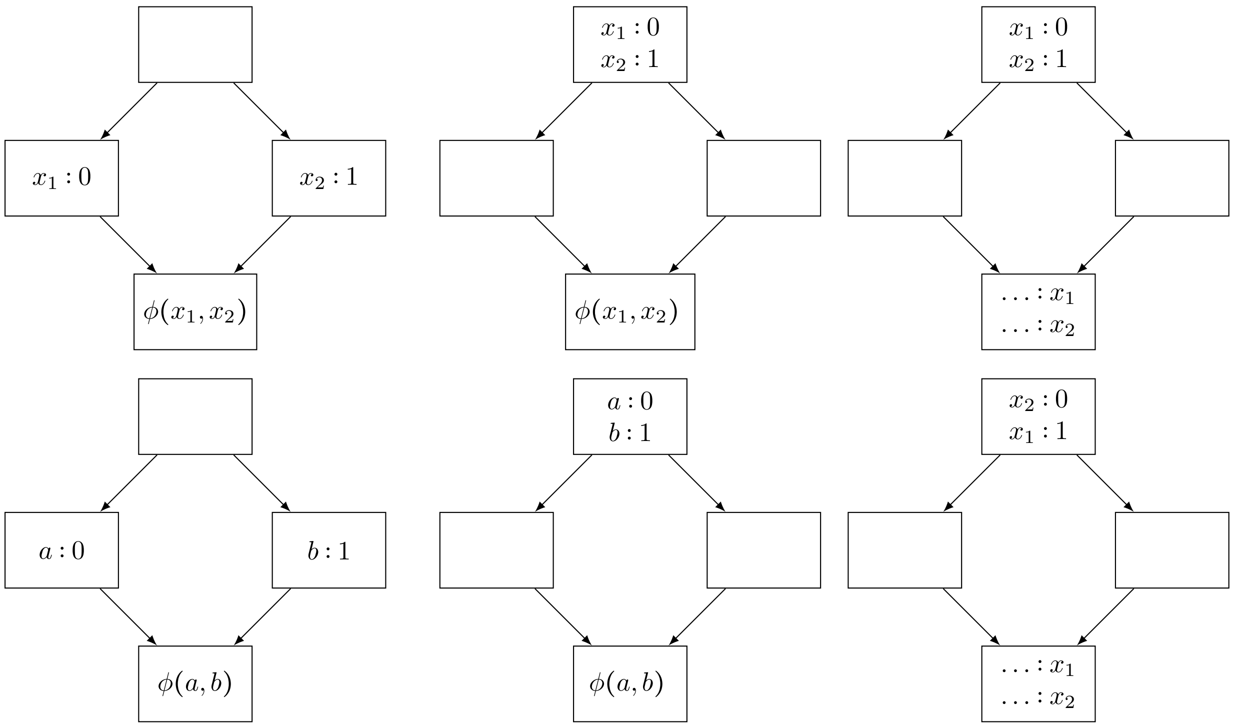
Exercise Sheet 8

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

Note: Neither the theoretical exercises nor the project assignment from this exercise sheet are for doing "over the holidays". You are encouraged to work on the assignments on the official working days before and after the holidays only.

Exercise 8.1 SSA-Property

Which of the following control flow graphs represent valid SSA-form programs? Justify your answer.



Exercise 8.2 Static Single Assignment (SSA) Form and Sparse Conditional Constant Propagation (SCCP)

Consider the following program P:

```
x = 1;
y = 1;
while (...) {
    if (x != 1)
        y = 2;
    z = x;
    x = 2;
    if (y == 1)
        x = z;
}
print(x);
```

1. Construct the abstract syntax tree (AST) from P.

2. Compute SSA form for P by using Cytron's algorithm:

(a) Construct the control flow graph (CFG) from P. Label each basic block with an uppercase latter.
(b) Construct the dominance tree for the CFG.
(c) Calculate the dominance frontier DF and the iterated dominance frontier DF^+ for each basic block.
(d) Give each definition a unique name. Furthermore, create names for the results of comparisons.
(e) Insert ϕ functions and rewire all uses.

3. Compute SSA form for P by using Braun et. al.'s simple algorithm:

   (a) Traverse the AST by depth-first search (visit the children from left to right). This corresponds to traversing the program text from top to bottom.
   (b) Successively create the CFG and fill it with instructions in SSA form. Note that you don't create "copy-instructions" when using this algorithm. Remember to create names for the result of a comparison. *Seal* basic blocks as early as possible. Insert ϕ functions on the fly.

4. Why does Cytron's algorithm insert more ϕ functions than Braun et. al.'s algorithm?

5. SCCP[1] is a combination of three data-flow analyses: Constant Propagation Analysis, Reachability Analysis and Constant Branch Analysis. The SCCP lattice is therefore the cartesian product of the lattices of each individual analysis and the transformer can use information from all three domains. Answer the following questions and perform SCCP on P in SSA form:

   (a) Write down all lattices (reachability, general constants, boolean constants for conditions).
   (b) Write down the transfer function for each basic block and each ϕ function.
   (c) Draw a table with one row for each SSA variable and one row for each basic block.
   (d) Initialize all values in the first column.
   (e) Now perform the algorithm until a fixed point is reached. Record updates to a value in a new column. Perform the updates in a smart order to minimize work.
   (f) Consider a program P with $|V|$ variables and $|B|$ basic blocks ($\mathcal{O}(|V|) = \mathcal{O}(|B|) = \mathcal{O}(|P|)$). Estimate (using $\mathcal{O}$-notation) the memory consumption (the size of the necessary tables) by the classic (non-sparse) and SSA-based sparse conditional constant propagation analyses.
   For the SSA estimation, let $|\phi|$ be the number of ϕ functions and $|def|$ be the number of assignments in P and consider these two cases:
      i. Suppose $\mathcal{O}(|\phi|) = \mathcal{O}(|def|)$.
      ii. Presume there are the maximum number of ϕ functions.

Project task F Intermediate Representation

- Install LLVM 5.0.0. Follow the instructions from the "Introduction to LLVM" slide set to do so.
- 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 */
```

[1] "Sparse Conditional Constant Propagation", consider `https://dl.acm.org/citation.cfm?id=103136` for more details (accessible from the university network)

- The soft deadline for this milestone is 2018-01-19.
- Keep it simple!