SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Reinhard Wilhelm
Prof. Dr. Sebastian Hack
Dipl.-Inform. Daniel Grund
Michael Jacobs, M.Sc.

Compiler Construction WS11/12

Exercise Sheet 8

Please hand in the solutions to the theoretical exercises until the beginning of the last lecture before the Christmas brake, 2011-12-23, 12:00. Please write the number of your tutorial group or the name of your tutor on the first sheet of your solution.

Exercise 8.1 Constant Propagation Analysis (Points: 2+4)

Consider the following program S of the While-language.

```
[x := 1]^1;
while [y > 0]^2 do (
  [x := 2-x]^3;
  [y := y-1]^4
)
```

- Provide the control flow graph (abstract flowchart) for program S following the representation by Nielson, Nielson and Hankin as introduced on the lecture slides.
- Perform the Constant Propagation Analysis on program S. In particular, provide the corresponding equation system for program S and perform a fixed point iteration.

Exercise 8.2 MFP and MOP (Points: 5+5)

We learned in the lecture that the MFP is a (least) solution to the constraint system of an analysis that is a monotone framework (compare slide 62 of the slide set on Data Flow Analysis by Nielson, Nielson, Hankin). So the following constraints hold for all $\ell \in \text{LAB}_\star$:

$$\begin{aligned} \text{MFP}_\circ(\ell) &\sqsupseteq \bigsqcup\{\text{MFP}_\bullet(\ell') \mid (\ell',\ell) \in F\} \sqcup \iota^\ell_E \\ &\text{where } \iota^\ell_E = \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases} \\ \text{MFP}_\bullet(\ell) &\sqsupseteq f_\ell(\text{MFP}_\circ(\ell)) \end{aligned}$$

The definition of the MOP solution is given on slides 77 and 78.
Prove the following Lemmas.

- In a monotone framework the following statements hold for all $\ell \in \text{LAB}_\star$:

$$\begin{aligned} \text{MOP}_\circ(\ell) &\sqsubseteq \text{MFP}_\circ(\ell) \\ \text{MOP}_\bullet(\ell) &\sqsubseteq \text{MFP}_\bullet(\ell) \end{aligned}$$

- In a distributive framework and under the assumption that all program instructions are reachable in the CFG from the program start ($\mathit{path}_\circ(\ell) \neq \emptyset$ for all $\ell \in \text{LAB}_\star$) the following statements hold for all $\ell \in \text{LAB}_\star$:

$$\begin{aligned} \text{MOP}_\circ(\ell) &= \text{MFP}_\circ(\ell) \\ \text{MOP}_\bullet(\ell) &= \text{MFP}_\bullet(\ell) \end{aligned}$$

Exercise 8.3 Extreme Pointer Analysis (Points: 4+4+4+2+2)

In this exercise you should develop an analysis that works on the CMa code of a single function and determines a safe upper bound on the extreme pointer for that function. The analysis should find a safe upper bound on the stack height for each point in the CMa code. A safe upper bound on the extreme pointer is then simply the maximum of the upper bounds for all points in the function.

- Consider the following C-function and transform it to a CMa program using the rules of code generation as presented in the lecture.

```
int sumParts (int number)
{
  int sum = 0;
  int i = 0;
  while (i < number) {
    sum = sum + part(i);
    i = i + 1;
  }
  return sum;
}
```

- Design your Extreme Pointer Analysis. Specify a domain for your analysis by giving a set and a partial order relation that forms a complete lattice on this set. Define a least upper bound operator to merge the information of different elements of your domain. Provide the transfer functions for all atomic CMa instructions introduced in the lecture slides. You can give one generic transfer function for all binary operators and one generic transfer function for all unary operators. Macro instructions, like `storea`, do not have to be considered.

  Make sure that your analysis is precise enough to deliver the same result as the precalculation of the extreme pointer for CMa code generated from C-code using the transformation rules presented in the lecture. Further make sure that your analysis terminates even if it cannot detect a finite upper bound on the extreme pointer for a given sequence of CMa instructions.

- Test your analysis on the CMa code you generated in the first part of this exercise. Provide a control flow graph of the CMa program. Derive the equation system and perform a fixed point iteration.

- Give an example for a sequence of CMa instructions that does only need a finite stack height on execution. Chose the example in such a way that your analysis will not be able to detect a finite upper bound for the extreme pointer. You do not need to perform the analysis.

- Explain why such a situation cannot happen with CMa code generated from a C-function according to the code generation rules introduced in the lecture.

Exercise 8.4 Sad but true (Bonus-Points: 4)

Formally prove the following claim. You cannot construct an optimal compiler which translates every given input program to a program with the minimal number of instructions.