SAARLAND
UNIVERSITY
COMPUTER SCIENCE

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

Compiler Construction WS15/16

Exercise Sheet 6

Exercise 6.1. Dominance and Data Flow

Develop a data flow analysis to compute the set of dominators of each block in a control flow graph. A control flow graph (CFG) consists only of basic blocks and control flow edges between them. Further, a CFG has a unique start node and each node is reachable from the start. A block A dominates a block B, if all paths from the start node to B must go through A. A basic block is a maximal sequence of instructions, which start with a label and end in a conditional or unconditional branch and does not contain any other label or branch. (The actual instructions do not matter for dominance.) Consider the following aspects:

- What is the domain, which is a complete lattice, of the analysis? In particular, describe $\perp$ and $\top$.
- What is the join operator $\sqcup$?
- If a block A is dominated by a block B, then all predecessors of A are also dominated by B.
- Each block dominates itself.
- What does it mean that information is (un-)safe for this analysis?
- Is the analysis performed forwards (along the control flow) or backwards (against the control flow)?
- What is the initialisation at each block?
- What would be the set of dominators of an unreachable block?

Draw the CFG for the following program and perform dominance analysis. Label each basic block with an uppercase letter.

```
void f(void) {
  if (...) {
    while (...) {
      if (...)
        break;
    }
  } else {
  }
}
```

Exercise 6.2. Monotinicity and Ascending Chains

In the lecture we saw the following two properties of a function f:

- Monotinicity: $x \leq y \rightarrow f(x) \leq f(y)$
- Ascending Chain: $x \leq f(x)$

Show that, in general, neither implies the other.

Project task E. Semantic Analysis

Implement semantic analysis.

- You can perform this either during parsing and AST construction or as a separate phase.
- Semantic analysis augments `--parse` and `--print-ast`.
- Major parts are name and type analysis. Name analysis associates identifiers with declarations. Type analysis associates expressions with types. It encompasses the Constraints and Semantics clauses.
- If you delayed certain syntactic checks (e.g. rejecting `a || b = c`), perform them now.
- The only *null pointer constant*, which you need to support, is literal `0`.
- The previous restriction and the restricted language subset ensure, that it is not necessary to evaluate the value of any expressions during semantic analysis.
- It is not necessary to accept programs, which contain functions, that return a struct or have one as parameter. Similarily, it is not necessary to accept programs, which contain assignments of struct type.
- It is not necessary to accept programs, which contain anonymous structs.
- You do not need to handle `__func__`.
- Use `int` for the types `ptrdiff_t` and `size_t`.
- Due to the restricted language subset, type compatibility degenerates to equality.
- For the error location use the location of the (first) terminal of the syntactic construct, where the error was detected. E.g. for adding two pointers, show the location of the `+`. For an `if`, whose condition is not scalar, show the location of the keyword `if`.
- If you are uncertain about some aspect, ask!