SAARLAND UNIVERSITY
COMPUTER SCIENCE

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

Compiler Construction WS15/16

Exercise Sheet 5

Exercise 5.1. Constant Propagation Analysis

Consider the following program S.

```
x ← 1;
while y > 0 do
  x ← 2−x;
  y ← y−1;
done
```

- Provide the control flow graph for program S.
- Constant Propagation Analysis maps a variable to a constant abstract value if and only if it the variable at that program point is known to always evaluate to that constant value. Define a complete lattice (e.g., use a Hasse diagram) suitable for Constant Propagation Analysis as well as the abstract transformer(s) for binary expressions.
- Perform the Constant Propagation Analysis on program S. In particular, provide the corresponding equation system for program S and perform a fixed-point iteration.
- Could the initial state be choosen differently? How will the results of your analysis change if so? Which advantages and disadvantages do you gain?

Project task D. Pretty Printing

Up to now your compiler can already lex and parse input. In this project phase you are to add a pretty printer to your compiler, i.e., functionality to generate nicely formatted source code from the parse tree. Therefore, you have to construct an abstract syntax tree (AST). Each AST node should be able to dump itself.

For a syntactically correct input program, `c4 --print-ast [file]` must print the output to stdout. This new switch must *augment* the `c4 --parse` functionality, i.e., its acceptance and rejection behavior as well as the respective return codes. Download the example file from `http://www.cdl.uni-saarland.de/teaching/cc/2015/exercises/ex05_pretty.c`. Invoking `c4 --print-ast ex05_pretty.c` should emit a char-exact copy of the source file. You can use the tool `diff` to compare your output to the reference file. Once again: *Your output must exactly match the reference output*.

You can use the pretty printer to debug your compiler. For example, the expression `a * b + d + e` should be dumped as `(((a * b) + d) + e)`. If you see a different output, your compiler handles associativity/operator precedence incorrectly.

In general, follow the reference file for placing newlines, spacing and so forth. The example mostly follows K&R style. Additionally, consider the following guidelines for your pretty printer:

- Print everything in the original order of the source file.
- Use a tab character (`'\t'`) for one level of indentation; do not use spaces.
- Do not wrap long lines.
- Do not print digraphs. Print the regular punctuator instead.
- Each (sub-)*expression* and each (sub-)*declarator*/*abstract-declarator* must be fully parenthesized, except for atoms consisting of at most one token like `0`, `'c'`, `"abc"`, *identifier*s and so forth. In particular, original parentheses are discarded. Do not re-associate any subexpressions.
- Do not hesitate to ask questions in the Forum.