Static Program Analysis
SS 2011
Graded Project 1

In this project you are to implement your own program analyzer/optimizer!
The goal of this project is implement a little program that can perform some simple static analyses on programs gi-
ven in our toy language from the lecture. Furthermore, this program shall be able to optimize the input program by
applying program transformations enabled by the analysis results. You are allowed to work in groups of 2 students
on the projects. You can use Java, C, C++, Lisp or Haskell as your implementation language. Your program has to
be submitted before Friday, June 17th, 16.00 hrs. Later submissions will not be accepted. Keep in mind that your
projects will be graded.

Exercise 1.1: Input

Your analyzer/optimizer expects a single input file specifying which analysis it shall perform, which fixed point
algorithm it shall use, what output it shall generate, and, of course, what program it shall analyze/optimize. Your
implementation shall support at least the following analyses: available expressions analysis, truly live variables
analysis, and interval analysis. Furthermore, your implementation shall be able to compute all analyses either
using round-robin iteration, the worklist algorithm or the recursive algorithm. Supported outputs are the analysis
results or a transformed/optimized program (depending on the input).

A formal specification as well as an example of the input language is given in the following subsection.

Exercise 1.2: Grammar: Input

\[
\begin{align*}
IN & \rightarrow \text{ANALYSIS} \\
\text{AN} & \rightarrow \text{ANALYSIS} \\
\text{AN} & \rightarrow \text{ALGORITHM} \\
\text{ALG} & \rightarrow \text{WORKLIST} | \text{ROUND ROBIN} | \text{RECURSIVE} \\
\text{OUT} & \rightarrow \text{ANALYSIS} | \text{TRANSFORMATION} \\
\text{PROG} & \rightarrow \text{EDG}^+ \\
\text{EDG} & \rightarrow \text{NODE Label Node}
\end{align*}
\]

where Node denotes the name of a node and label a program statement from the following list:

- \text{;}
- \text{Pos(e)}
- \text{Neg(e)}
• \( x = e; \)
• \( x = M[e]; \)
• \( M[e_1] = e_2; \)

The grammar for expressions \( e, e_1, \) and \( e_2 \) is given by

\[
\begin{align*}
\text{EXPR} & \rightarrow \text{UOP ATOM} \mid \text{ATOM BOP ATOM} \mid \text{ATOM} \\
\text{ATOM} & \rightarrow \text{var} \mid \text{constant} \\
\text{UOP} & \rightarrow + \mid - \\
\text{BOP} & \rightarrow + \mid - \mid * \mid / \mid < \mid > \mid <= \mid >= \mid == \mid !=
\end{align*}
\]

where \( \text{var} \) is a valid identifier and \( \text{constant} \) an integer.

Notes

• The truly live variables analysis shall assume that \( X \), the set of variables (truly) live at the end of the program, is the empty set.

• The interval analysis shall use widening/narrowing.

Example

ANALYSIS
Available_Expressions

ALGORITHM
Worklist

OUTPUT
Transformation

PROGRAM
\begin{verbatim}
START x = 2; P1
P1 y = 36; P2
P2 t = x * 21; P3
P3 Pos(y > 0) P4
P3 Neg(y > 0) P5
P4 y = y - 1; P6
P6 t = x * 21; P7
P7 ; P3
P5 x = 5; P8
P8 M[0] = x + t; END
\end{verbatim}

Exercise 1.3: Output

Write the output of your implementation to standard out (stdout). To allow for some automated testing of your implementations, please generate your output according to the grammar given in the following subsection.
Exercise 1.4: Grammar: Output

\[
OP \rightarrow \text{ANALYSIS\_RESULTS}
\]

\[
NRES^+ | \text{TRANSFORMATION\_RESULT}
\]

\[
PROG
\]

\[
NRES \rightarrow \text{Node} : \text{RESULT}
\]

where \text{RESULT} is of form

- If analysis was available expressions analysis:
  
  \[
  \text{RESULT} \rightarrow \{\} | \{\text{expr EXPLIST}\}
  \]
  
  \[
  \text{EXPLIST} \rightarrow \text{, expr EXPLIST} | \epsilon
  \]

  Please sort the sets of expressions lexicographically and do not do any normalization of expressions.

- If analysis was truly live variables analysis:
  
  \[
  \text{RESULT} \rightarrow \{\} | \{\text{var VARLIST}\}
  \]
  
  \[
  \text{VARLIST} \rightarrow \text{, var VARLIST} | \epsilon
  \]

  Please sort the sets of variables lexicographically.

- If analysis was interval analysis:
  
  \[
  \text{RESULT} \rightarrow \{\text{INTT}^*\}
  \]
  
  \[
  \text{INTT} \rightarrow (\text{var}, [\text{lowerbound}, \text{upperbound}])
  \]

  where \text{lowerbound} is either an integer or -INF and \text{upperbound} either an integer or +INF. Please sort the sets of variable/interval pairs according the lexicographical order of the variable names.

Example

ANALYSIS\_RESULTS

\[
\text{START} : \{\}
\]

\[
\text{P1} : \{\}
\]

\[
\text{P2} : \{\}
\]

\[
\text{P3} : \{x \times 21\}
\]

\[
\text{P4} : \{x \times 21, y > 0\}
\]

\[
\text{P5} : \{x \times 21, y > 0\}
\]

\[
\text{P6} : \{x \times 21\}
\]

\[
\text{P7} : \{x \times 21\}
\]

\[
\text{P8} : \{y > 0\}
\]

\[
\text{END} : \{x + t, y > 0\}
\]