Compiler Design

Spring 2010

Data-Flow Analysis

Sample Exercises and Solutions

Prof. Pedro C. Diniz

USC / Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, California 90292
pedro@isi.edu
Problem 1:

Your goal is to solve the Anticipation Analysis data-flow problem. The idea is to understand how early one could compute an expression in the program before the expression needs to be used.

Definition: *We say an expression $e$ is anticipated at point $p$ if the same expression, computing the same value, occurs after $p$ in every possible execution path starting at $p$.*

This is a necessary condition for inserting a calculation at $p$, although it is not sufficient, since the insertion may not be profitable. Your data-flow analysis must determine whether a particular expression $a + b$ in the program is anticipated at the entry of each basic block. (This can easily be generalized to the analysis of anticipation for every expression in the program).

For the example in the CFG below the expression $a+b$ is anticipated in the beginning of the basic block BB2 but not at the beginning of basic block BB1 since in the later case there is a control path in which the value of the expression changes due to the assignment in basic block BB6.

Describe your approach to anticipation analysis by explaining the following:

1. Direction of the problem, backwards or forward and why?
2. Your representation format and the initial values based on the suggested representation?
3. The definition and rationale for the GEN and KILL sets.
4. The equations that the iterative approach needs to solve.

Solution:

1. This is essentially the inverse version of the available expressions data-flow problem. Here we are asked if a given expression is anticipated at a given point $p$ of the program, i.e., if for all paths starting at $p$ and observe if none of its arguments are redefined. We thus work backwards from the use or creation point of the expression and trace back to see until when are one of its operands redefined. At bifurcation points we will determine if on the other path (along which we did not come) the same expression is also anticipated, i.e. there is a path to a use of the same expression whose operands are not redefined.
2. The lattice of values consists of sets of all the expressions computed in the program, in this case we can number the expressions in the program and use sets of integers to represent which of the expressions at point $p$ are anticipated. The initial values for the OUT of all the basic blocks are the universe, i.e., all expressions are initially anticipated.

3. The GEN set for an instruction and also the basic block, define which set of expressions a given instruction generates. The KILL set is the set of all expressions a given assignment statement eliminates and thus uses the LHS variables and kills all the expressions where the LHS variable appears.

4. Given that this is a backwards problem we have to define the IN of each basic block as a function of its OUT. The meet function is the intersection given that for one expression to be anticipated at a given point $p$ it needs to be anticipated at all paths starting at $p$. To compute the IN of each basic block/instruction we use the equation $\text{IN} = \text{GEN} + (\text{OUT} - \text{KILL})$ where $+$ and $-$ stand for set union and difference respectively.

We illustrate the application of this Data-Flow Analysis to the following program and corresponding CFG where we have omitted the computation of the data-flow solution to all the intermediate program point in each basic block (the so called local phase).
Problem 2:

01         a = 1
02         b = 2
03 L0:   c = a + b
04         d = c - a
05       if c < d goto L2
06 L1:   d = b + d
07       if d < 1 goto L3
08 L2:   b = a + b
09         e = c - a
10       if e = 0 goto L0
11         a = b + d
12         b = a - d
13     goto L4
14 L3:   d = a + b
15         e = e + 1
16     goto L3
17L4:   return

For the code shown above, determine the following:

a) The basic blocks of instructions.
b) The control-flow graph (CFG)

c) For each variable, its corresponding du-chain.
d) The live variables at the end of each basic block. You do not need to determine the live
variables before and after each basic block and justify your answer for the value presented
for the basic block containing instructions at line 6 and 7.
e) Is the live variable analysis a forward or backward data-flow analysis problem? Why and
what does guarantee its termination when formulated as a data-flow analysis iterative
problem?
Solution:

a) b) We indicate the instruction in each basic block and the CFG and dominator tree below.

BB1: \{01, 02\}
BB2: \{03, 04, 05\}
BB3: \{06, 07\}
BB4: \{08, 09, 10\}
BB5: \{11, 12, 13\}
BB6: \{14, 15, 16\}

c) The def-use chains can be obtained by inspection of which definitions are not killed along all path from its definition point to the use points. For example for variable “a” its definition at the instruction 1, denoted by a1 reaches the use points at the instructions 3, 4, 8, 9 and 15, hence the notation \{d1, u3, u4, u8, u9, 15\}. Definition a1 does not reach use u12 because there is another definition at 11 that masks it, hence another du chain for “a” denoted as \{d11, u12\}. The complete list is shown below.

a: \{d1, u3, u4, u8, u9, 15\}
a: \{d11, u12\}
b: \{d2, u3, u4, u6, u14, u8\}
b: \{d8, u11, u8, u14\}
b: \{d12\}
c: \{d3, u4, u5, u9\}
d: \{d4, u5, u6\}
d: \{d6, u7\}
d: \{d14, u6\}
e: \{d9, u10, u15\}
e: \{d14, u15\}

d) At the end of each basic block we can inspect the code and ask if a given variable is still used after the control flow leaves that basic block. The results of this inspection is depicted below:

BB1: \{a, b\}
BB2: \{a, b, c, d, e\}
BB3: \{a, b, c, d, e\}
BB4: \{a, b, d, e\}
BB5: \{\}
BB6: \{a, b, c, e\}
For BB6 the live variable solution at the exit of this basic block has \{a, b, c, e\} as for variable “d” there is no path out of this basic block where the current value of the variable is used. For variable “d” the value is immediately redefined in BB3 without any possibility of being used.

e) The live variable analysis is a backward data-flow problem as we propagate the information about a future use of a variable backward to specific points of the program. If there is a definition at a specific point backward the solution kills all other uses and resets the information associated with that variable. As with many other iterative formulations of data-flow analysis problems termination is guaranteed by the fact that the lattice, in this case the set of variables, has finite cardinality or length. The flow-function, in this case set-union is monotonic.