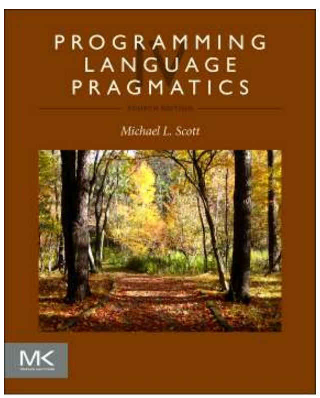
Data Flow Analysis

17-363/17-663: Programming Language Pragmatics



Reading: PLP chapter 17

Prof. Jonathan Aldrich

ELSEVIER

Data Flow Analysis

- Data flow analysis tracks the flow of information across basic block boundaries
- Examples:
  - **Reaching definitions:** which assignments to a variable reach a given program point?
  - **Available expressions:** which expressions are available in a (virtual) register?
  - **Constant propagation:** what variables hold constant values?
  - **Sign analysis:** is a variable positive, negative, zero, or of unknown sign?
  - **Range analysis:** what is the maximum and minimum value of a variable at a program point?

Data Flow Analysis Frameworks

- Many instances of data flow analysis can be cast in the following framework:
  1. four sets for each basic block B, called In_B, Out_B, Gen_B, and $Kill_B$;
  2. values for the *Gen* and *Kill* sets;
  3. an equation relating the sets for any given block B;
  4. an equation relating the *Out* set of a given block to the *In* sets of its successors, or relating the *In* set of the block to the *Out* sets of its predecessors; and (often)
  5. certain initial conditions

Global Redundancy and Data Flow Analysis

- The goal of the analysis is to find a *fixed point* of the equations: a consistent set of *In* and *Out* sets (usually the smallest or the largest) that satisfy both the equations and the initial conditions
  - Some problems have a single fixed point
  - Others may have more than one
    - we usually want either the least or the greatest fixed point (smallest or largest sets)

Global Redundancy and Data Flow Analysis

- In the case of *global common subexpression elimination*, In_B is the set of expressions (virtual registers) guaranteed to be available at the beginning of block *B*
  - These *available expressions* will all have been set by predecessor blocks
  - Out_B is the set of expressions guaranteed to be available at the end of *B*
  - $Kill_B$ is the set of expressions *killed* in *B*: invalidated by assignment to one of the variables used to calculate the expression, and not subsequently recalculated in *B*
  - Gen_B is the set of expressions calculated in *B* and not subsequently killed in *B*

Global Redundancy and Data Flow Analysis

- The data flow equations for available expression analysis are:

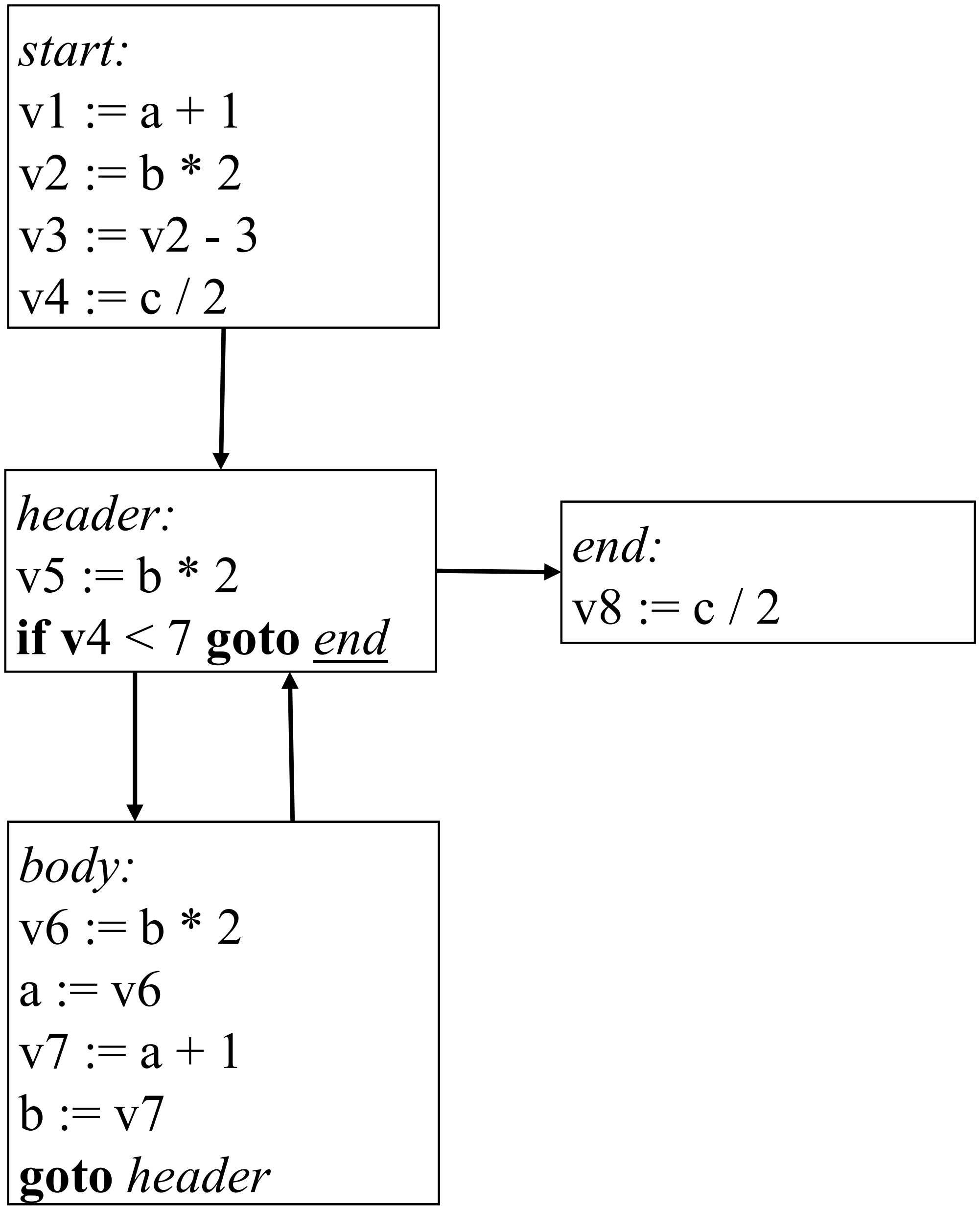
$$\begin{aligned} Out_B &= Gen_B \cup (In_B \setminus Kill_B) \\ In_B &= \bigcap_{\text{predecessors } A \text{ of } B} Out_A \end{aligned}$$

- Our initial condition is $In_1 = \varnothing$: no expressions are available at the beginning of execution

Global Redundancy and Data Flow Analysis

- Available expression analysis is known as a *forward* data flow problem, because information flows forward across branches: the *In* set of a block depends on the *Out* sets of its predecessors
  - We will see an example of a *backward* data flow problem later
- We calculate the desired fixed point of our equations in an inductive (iterative) fashion
- Our equation for In_B uses intersection to insist that an expression be available on all paths into B
  - In our iterative algorithm, this means that In_B can only shrink with subsequent iterations

Example of Available Expressions Analysis



ELSEVIER

Exercise: Apply available expressions analysis to this program

start:
v1 := a*2
v2 := b-1
if v1 < v2 **goto** *else*

then:
a := v2
v1 := a*2
v4 := v1+3
goto *endif*

else:
b := v1
v4 := v1+3

endif:
v1 := a*2
v7 := v1+3
return v7

Live Variable Analysis

- We turn our attention to *live variable analysis* -very important in any subroutine in which global common subexpression analysis has eliminated load instructions
- Live variable analysis is a *backward* flow problem
- It determines which instructions produce values that will be needed in the future, allowing us to eliminate *dead* (useless) instructions
  - in our example we consider only values written to memory and with the elimination of dead stores
  - applied to values in virtual registers as well, live variable analysis can help to identify other dead instructions

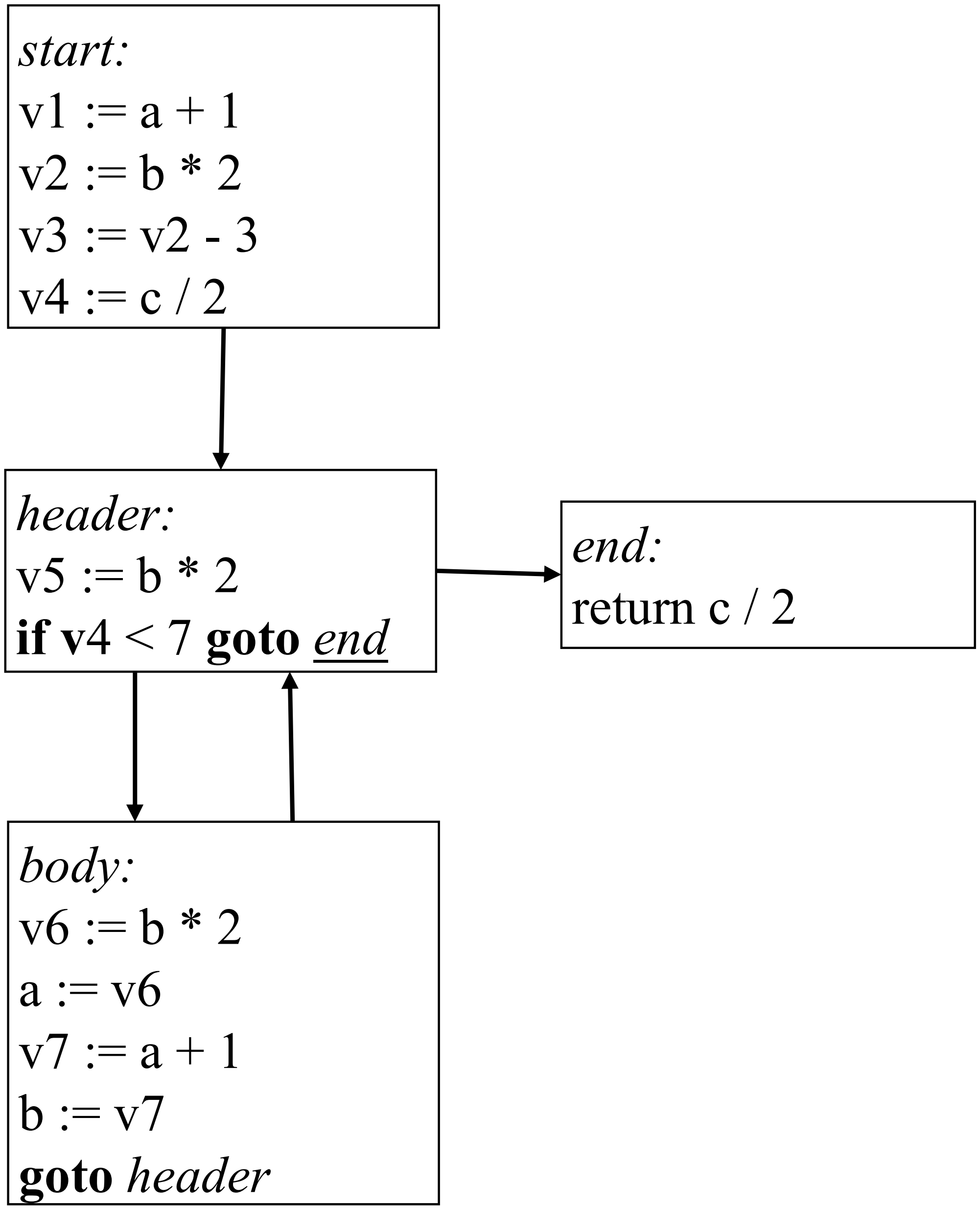
Live Variable Analysis

- For this instance of data flow analysis
  - In_B is the set of variables live at the beginning of block B
  - Out_B is the set of variables live at the end of the block
  - Gen_B is the set of variables read in B without first being written in B
  - $Kill_B$ is the set of variables written in B without having been read first
- The data flow equations are:

$$
\begin{aligned}
In_B &= Gen_B \cup (Out_B \setminus Kill_B) \\
Out_B &= \bigcup_{\text{successors } C \text{ of } B} In_C
\end{aligned}
$$

ELSEVIER

Example of Live Variable Analysis and Dead Code Elimination



Exercise: Apply live variable analysis and dead code elimination to this program