Dataflow Analysis

Hal Perkins
Winter 2009

Agenda
- Initial example: dataflow analysis for common subexpression elimination
- Other analysis problems that work in the same framework

Available Expressions

Goal: use dataflow analysis to find common subexpressions
Idea: calculate available expressions at beginning of each basic block
Avoid re-evaluation of an available expression - use a copy operation
- Simple inside a single block; more complex dataflow analysis used across blocks

Available Expression Sets

For each block \( b \), define
- \( \text{AVAIL}(b) \) - the set of expressions available on entry to \( b \)
- \( \text{NKILL}(b) \) - the set of expressions not killed in \( b \)
- \( \text{DEF}(b) \) - the set of expressions defined in \( b \) and not subsequently killed in \( b \)

“Available” and Other Terms

An expression \( e \) is defined at point \( p \) in the CFG if its value is computed at \( p \)
- Sometimes called definition site
An expression \( e \) is killed at point \( p \) if one of its operands is defined at \( p \)
- Sometimes called kill site
An expression \( e \) is available at point \( p \) if every path leading to \( p \) contains a prior definition of \( e \) and \( e \) is not killed between that definition and \( p \)

Computing Available Expressions

\( \text{AVAIL}(b) \) is the set
\[
\text{AVAIL}(b) = \bigcap_{x \in \text{preds}(b)} (\text{DEF}(x) \cup (\text{AVAIL}(x) \cap \text{NKILL}(x)))
\]
- \( \text{preds}(b) \) is the set of \( b \)'s predecessors in the control flow graph
- This gives a system of simultaneous equations - a dataflow problem
Computing Available Expressions

- **Big Picture**
  - Build control-flow graph
  - Calculate initial local data – DEF(b) and NKILL(b)
    - This only needs to be done once
  - Iteratively calculate AVAIL(b) by repeatedly evaluating equations until nothing changes
  - Another fixed-point algorithm

- **Computing DEF and NKILL (1)**
  - For each block \( b \) with operations \( o_1, o_2, \ldots, o_k \)
    - KILLED = \( \emptyset \)
    - DEF(b) = \( \emptyset \)
    - for \( i = k \) to 1
      - assume \( o_i \) is “\( x = y + z \)”
      - if (\( y \notin \) KILLED and \( z \notin \) KILLED)
        - add “\( y + z \)” to DEF(b)
        - add \( x \) to KILLED

- **Computing DEF and NKILL (2)**
  - After computing DEF and KILLED for a block \( b \)
    - NKILL(b) = \{ all expressions \}
    - for each expression \( e \)
      - for each variable \( v \in e \)
        - if \( v \in \) KILLED then
          - NKILL(b) = NKILL(b) - \( e \)

- **Computing Available Expressions**
  - Once DEF(b) and NKILL(b) are computed for all blocks \( b \)
    - Worklist = \{ all blocks \( b \) \}
    - while (Worklist != \( \emptyset \))
      - remove a block \( b \) from Worklist
      - recompute AVAIL(b)
      - if AVAIL(b) changed
        - Worklist = Worklist \( \cup \) successors(b)

Dataflow analysis

- Available expressions are an example of a dataflow analysis problem
- Many similar problems can be expressed in a similar framework
- Only the first part of the story - once we’ve discovered facts, we then need to use them to improve code

Characterizing Dataflow Analysis

- All of these algorithms involve sets of facts about each basic block \( b \)
  - \( \text{IN}(b) \) – facts true on entry to \( b \)
  - \( \text{OUT}(b) \) – facts true on exit from \( b \)
  - \( \text{GEN}(b) \) – facts created and not killed in \( b \)
  - \( \text{KILL}(b) \) – facts killed in \( b \)
  - These are related by the equation
    - \( \text{OUT}(b) = \text{GEN}(b) \cup (\text{IN}(b) \setminus \text{KILL}(b)) \)
    - Solve this iteratively for all blocks
    - Sometimes information propagates forward; sometimes backward
Efficiency of Dataflow Analysis

- The algorithms eventually terminate, but the expected time needed can be reduced by picking a good order to visit nodes in the CFG
  - Forward problems - reverse postorder
  - Backward problems - postorder

Example: Available Expressions

- This is the analysis we did to detect redundant expression evaluation
- Equation:
  \[
  \text{AVAIL}(b) = \bigcap_{x \in \text{preds}(b)} (\text{DEF}(x) \cup (\text{AVAIL}(x) \cap \text{NKILL}(x)))
  \]

Example: Live Variable Analysis

- A variable \( v \) is live at point \( p \) iff there is any path from \( p \) to a use of \( v \) along which \( v \) is not redefined
- Uses
  - Register allocation - only live variables need a register (or temporary)
  - Eliminating useless stores
  - Detecting uses of uninitialized variables

Equations for Live Variables

- Sets
  - \( \text{USED}(b) \) - variables used in \( b \) before being defined in \( b \)
  - \( \text{NOTDEF}(b) \) - variables not defined in \( b \)
  - \( \text{LIVE}(b) \) - variables live on exit from \( b \)
- Equation
  \[
  \text{LIVE}(b) = \bigcup_{s \in \text{succ}(b)} \text{USED}(s) \cup (\text{LIVE}(s) \cap \text{NOTDEF}(s))
  \]

Example: Reaching Definitions

- A definition \( d \) of some variable \( v \) reaches operation \( i \) iff \( i \) reads the value of \( v \) and there is a path from \( d \) to \( i \) that does not define \( v \)
- Uses
  - Find all of the possible definition points for a variable in an expression

Equations for Reaching Definitions

- Sets
  - \( \text{DEFOUT}(b) \) - set of definitions in \( b \) that reach the end of \( b \) (i.e., not subsequently redefined in \( b \))
  - \( \text{SURVIVED}(b) \) - set of all definitions not obscured by a definition in \( b \)
  - \( \text{REACHES}(b) \) - set of definitions that reach \( b \)
- Equation
  \[
  \text{REACHES}(b) = \bigcup_{p \in \text{preds}(b)} \text{DEFOUT}(p) \cup (\text{REACHES}(p) \cap \text{SURVIVED}(p))
  \]
Example: Very Busy Expressions

- An expression $e$ is considered *very busy* at some point $p$ if $e$ is evaluated and used along every path that leaves $p$, and evaluating $e$ at $p$ would produce the same result as evaluating it at the original locations.

- Uses
  - Code hoisting – move $e$ to $p$ (reduces code size; no effect on execution time)

Equations for Very Busy Expressions

- Sets
  - $\text{USED}(b)$ – expressions used in $b$ before they are killed
  - $\text{KILLED}(b)$ – expressions redefined in $b$ before they are used
  - $\text{VERYBUSY}(b)$ – expressions very busy on exit from $b$

- Equation
  $$\text{VERYBUSY}(b) = \bigcap_{s \in \text{succ}(b)} \text{USED}(s) \cup (\text{VERYBUSY}(s) - \text{KILLED}(s))$$

And so forth...

- General framework for discovering facts about programs
  - Although not the only possible story
  - And then: facts open opportunities for code improvement

- To be continued...
  - CSE 501!