CSC2/455 Software Analysis and Improvement
Partial Redundancy Elimination

Sreepathi Pai

URCS

February 25, 2019
Outline

Review

Partial Redundancy Elimination

Postscript
Outline

Review

Partial Redundancy Elimination

Postscript
Optimizations: Dead Code Elimination

- Find useful operations (backward analysis)
- Find useful conditional branches
  - Reverse Dominance Frontier
- Remove code, and “touch up CFG”
Outline

Review

Partial Redundancy Elimination

Postscript
Redundancy: Fully Redundant

\[ a = b + c \]
\[ b = 7 \]
\[ d = b + c \]
\[ e = b + c \]

\[ t = b + c \]
\[ a = t \]
\[ b = 7 \]
\[ t = b + c \]
\[ d = t \]

\[ e = t \]
Redundancy: Loop Invariant

\[
\begin{align*}
& a = b + c \\
& t = b + c \\
& a = t
\end{align*}
\]
Redundancy: Partial Redundancy

\[ a = b + c \]
\[ d = b + c \]
\[ t = b + c \]
\[ a = t \]
\[ t = b + c \]
\[ d = t \]
Eliminating Redundancy: Complication 1

Can we insert \( t = b + c \) in \( B_3 \)?
Splitting Critical Edges

(Similar to when we were inserting minimal $\phi$-functions.)
Eliminating Redundancy: Complication 2

Note that there is no block where \( t = b + c \) can be introduced without introducing computations not in the original program.
CFG duplication

(Possibility of exponential blowup.)
The Lazy Code Motion Algorithm

- Eliminate all expressions when it will not duplicate code
- Do not perform computations not in original program
  - Although where the computation is performed can change
- Delay computation for as long as possible
  - "Lazy"
  - Helps lower resource (esp. register) usage
Setup

For all blocks $B$ in CFG, compute:

- $e_{use_B}$: set of expressions used in a block
- $e_{kill_B}$: set of expressions killed in block
  - usually by redefining subcomponents

Also, split all critical edges, inserting empty blocks.
Anticipable Expressions

Recall *very busy expressions*. An expression $e$ is anticipable at at block $p$ if:

► ?
Anticipable Expressions

Recall *very busy expressions*. An expression $e$ is anticipable at block $p$ if:

- $e$ is used/computed on all paths leading out of $p$
- And it is not killed before the use
- Implies that $p$ can compute $e$ and all paths could use this result
Anticipable Expressions Analysis

- **Direction**: Backwards
- **Values**: Expressions in programs
- **Meet**: \( \cap \)
- **Transfer Function**
  - \( f_B(x) = e\_use_B \cup (x - e\_kill_B) \)
- **Equations**:
  - \( \text{OUT}[B] = \wedge_{S \in \text{succ}(B)} \text{IN}[S] \)
  - \( \text{IN}[B] = f_B(\text{OUT}[B]) \)
- \( \top = U \)
- \( \text{IN}[\text{EXIT}] = \emptyset \)
Available Expressions

An expression is available at a program point $p$ if:

- it has been computed along all paths leading into $p$
- it has not been killed since being computed until $p$
- (NEW) it is anticipated at $p$
  - we could make it available if it is anticipated
Available Expressions Analysis

- **Direction**: Forwards
- **Meet**: $\cap$
- **Transfer function**
  - $f_B(x) = (e_{\use_B} \cup \text{anticipable}[B].\text{in}) \cup (x - e_{\kill_B})$
- **Equations**
  - $\text{IN}[B] = \wedge_{P \in \text{pred}(B)} \text{OUT}[P]$
  - $\text{OUT}[B] = f_B(\text{IN}[B])$
- $\top = U$
- $\text{OUT}[\text{ENTRY}] = \emptyset$
Positioning Expressions

- When is the earliest an expression can be evaluated?
- When is the latest an expression can be evaluated?
Positioning Expressions: Earliest

- When is the earliest an expression can be evaluated?
  - When it anticipated, but not available
  - $\text{earliest}[B] = \text{anticipable}[B].in - \text{available}[B].in$
  - Observe notation for results of different analyses
Anticipable + (Not) Available = Earliest

c = 2
a = b + c
d = b + c
e = b + c
Positioning Expressions: Latest

- When is the latest an expression can be evaluated?
  - When it can no longer be postponed
- “Postponed”: expression pushed down from earliest placement

- When can we push down an expression into the next block?
Postponable Expressions

An expression \( e \) is postponable to a block \( p \) if:

- \( e \) could be placed in block \( b \) before \( p \) (earliest is before \( p \))
- Such that it is available on all paths leading to \( p \) from ENTRY
- But \( e \) is not used after block \( b \) (and before \( p \))
Postponable Expressions Analysis

- Direction: Forwards
- Values: Expressions
- Meet: $\cap$
- Transfer functions
  - $f_B(x) = (\text{earliest}[B] \cup x) - e\text{-}_use_B$
- Equations
  - $\text{OUT}[B] = f_B(\text{IN}[B])$
  - $\text{IN}[B] = \land_{P \in \text{pred}(B)} \text{OUT}[P]$
- $\top = U$
- $\text{OUT}[\text{ENTRY}] = \emptyset$
Postponable

c = 2

a = b + c
d = b + c
e = b + c
Postponement Frontier

A block \( p \) is on the *postponement frontier* for an expression \( e \) if

- \( e \) can be postponed to \( p \)
- \( e \) cannot be placed at entry to a successor \( s \) of \( p \)
  - \( e \) is used in \( p \)
  - \( e \) is not postponable from some predecessor of \( s \)
  - \( e \) is not in \( \text{earliest}[S] \)

\[
\text{latest}[B] = \left( \text{earliest}[B] \cup \text{postponable}[B].\text{in} \right) \cap \left( \text{\_use}_B \cup (\cap_{S \in \text{succ}(B)}(\text{earliest}[S] \cup \text{postponable}[S].\text{in}))^C \right)
\]

(Note: \( A^C \) means the complement of set \( A \))
Used Expressions

An expression $e$ in block $p$ is used if:

- Some block $q$ uses $e$
- There exists a path from $p$ to $q$ that does not invalidate $e$
  - I.e. recompute $e$ or invalidate its operands
**Used Expressions Analysis**

- **Direction**: Backwards
- **Values**: Expressions
- **Meet**: $\cup$
- **Transfer function**
  - $f_B(x) = (x \cup e_{use_B}) - latest[B]$
- **Equations**
  - $IN[B] = f_B(OUT[B])$
  - $OUT[B] = \land_{s \in succ(B)} IN[s]$
  - $T = \emptyset$
  - $IN[EXIT] = \emptyset$
Putting it all together - I

- Compute *anticipable*[\(B\)].\textit{in}, *available*[\(B\)].\textit{in}
- Compute *earliest*[\(B\)]
- Compute *postponable*[\(B\)].\textit{in}
- Compute *latest*[\(B\)]
- Compute *used*[\(B\)].\textit{out}
For each expression $x + y$ in program:

- Create $t = x + y$ (where $t$ is a unique temporary)
- Place $t = x + y$ at the beginning of all blocks $B$ such that
  - $x + y$ is in $\text{latest}[B] \cap \text{used}[B].out$
  - i.e. $B$ is the last block where $x + y$ can be placed, and $x + y$ is used after $B$
- Replace all $x + y$ with $t$ in all block $B$ where:
  - $x + y \in (\text{e}_\text{use}_B \cap (\text{latest}[B]^C \cup \text{used}[B].out))$
  - i.e., $x + y$ is in $\text{e}_\text{use}_B$, and
  - $x + y$ is NOT in $\text{latest}[B]$, or
  - $x + y$ is in $\text{used}[B].out$

Algorithm 9.36 in the Dragon Book.
Final result

c = 2

\[ t = b + c \]

a = t

d = t

e = t
Outline

Review

Partial Redundancy Elimination

Postscript
References

- Chapter 9 of the Dragon Book
  - Section 9.5