Preliminary Transformations (contd)

Chapter 4 of Allen and Kennedy
Constant Propagation

- Replace all variables that have constant values at runtime with those constant values.
procedure propagateConst(P);

// P is the procedure in which constants are to be propagated
// valin(w, s) is the best approximate value of input w to s
// valout(v, s) is the best value of output v from s
// \mu(s)(inputs to s) is the result of symbolic interpretation of
//   statement s over the lattice values of its inputs. The
//   output is the lattice value of the output of the statement

for all statements s in the program do begin
  for each output v of s do valout(v, s) := unknown;
  for each input w of s do
    if w is a variable then valin(w, s) := unknown;
    else valin(w, s) := the constant value of w;
end;
Constant Propagation

worklist := \{ all statements of constant form, e.g., \( x = 5 \) \};

\textbf{while} worklist \( \neq \emptyset \) \textbf{do begin}
  choose and remove an arbitrary statement \( x \) from worklist;
  let \( v \) denote the output variable for \( x \);

  // Symbolic interpretation of the statement \( x \)
  newval := \mu(x)(valin(v, x), for all inputs \( v \) to \( x \));

  \textbf{if} newval \( \neq \) valout(v, x) \textbf{then begin}
    valout(x, v) := newval;
    \textbf{for all} (x, y) \( \in \) defuse \textbf{do begin}
      oldval := valin(v, y);
      valin(v, y) := oldval \land valout(v, x);
      \textbf{if} valin(v, y) \( \neq \) oldval \textbf{then} worklist := worklist \cup \{ x \};
    \textbf{end}
  \textbf{end}
\textbf{end}
\textbf{propagateConst}
Forward Expression Substitution

- Example

```
DO I = 1, 100
  K = I + 2
  A(K) = A(K) + 5
ENDDO
```

```
DO I = 1, 100
  A(I+2) = A(I+2) + 5
ENDDO
```
Forward Expression Substitution

• Need definition-use edges and control flow analysis
• Need to guarantee that the definition is always executed on a loop iteration before the statement into which it is substituted
• Data structure to find out if a statement $S$ is in loop $L$
  — Test whether level-$K$ loop containing $S$ is equal to $L$
Induction Variable Substitution

- Definition: an auxiliary induction variable in a DO loop headed by DO I = LB, UB, S is any variable that can be correctly expressed as $c_{expr} \times I + i_{expr_L}$ at every location L where it is used in the loop, where $c_{expr}$ and $i_{expr_L}$ are expressions that do not vary in the loop, although different locations in the loop may require substitution of different values of $i_{expr_L}$. 
Induction Variable Substitution

- Example:

```plaintext
DO I = 1, N
   A(I) = B(K) + 1
   K = K + 4
   ...
   D(K) = D(K) + A(I)
ENDDO
```
Induction Variable Substitution

- More complex example

```fortran
DO I = 1, N, 2
    K = K + 1
    A(K) = A(K) + 1
    K = K + 1
    A(K) = A(K) + 1
ENDDO
```

- Alternative strategy is to recognize region invariance

```fortran
DO I = 1, N, 2
    A(K+1) = A(K+1) + 1
    K = K+1 + 1
    A(K) = A(K) + 1
ENDDO
```
Induction Variable Substitution

• Driver

procedure IVDrive(L);
  // L is the loop being processed, assume SSA graph available
  // IVDrive performs forward substitution and induction variable
  // substitution on the loop L, recursively calling itself where
  // necessary.

  foreach statement S in L in order do
    case(kind(S))
      assignment:
        FS_not_done := ForwardSub(S,L);
        if FS_not_done then IVSub(S,L);
      DO-loop:
        IVDrive(S);
      default:
        end case
    end do
  end IVDrive;
IVSub without loop normalization

\[
\begin{align*}
\text{DO } & I = L, U, S \\
& K = K + N \\
& \ldots = A(K) \\
\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
\text{DO } & I = L, U, S \\
& \ldots = A(K + (I - L + S) / S * N) \\
\text{ENDDO}
\end{align*}
\]

\[
K = K + (U - L + S) / S * N
\]
IVSub without loop normalization

- **Problem:**
  - Inefficient code
  - Nonlinear subscript
IVSub with Loop Normalization

\[ I = 1 \]
\[ \text{DO } J = 1, (U-L+S)/S, 1 \]
\[ \quad K = K + N \]
\[ \quad \ldots = A(K) \]
\[ \quad I = I + 1 \]
\[ \text{ENDDO} \]
IVSub with Loop Normalization

\begin{verbatim}
I = 1
DO J = 1, (U - L + S) / S, 1
    ... = A (K + J * N)
ENDDO
K = K + (U - L + S) / S * N
I = I + (U - L + S) / S
\end{verbatim}
Summary

• Transformations to put more subscripts into standard form
  — Loop Normalization
  — Constant Propagation
  — Induction Variable Substitution

• Do loop normalization before induction-variable substitution

• Leave optimizations to compilers
  — Alternatively, perform preliminary transformations as look-aside analyses (then you’re guaranteed to “do no harm”)
Homework #3 (Written Assignment)

1. Solve exercise 3.6 in book
   — This is case 4 of Lemma 3.3
   — Read Definitions 3.1, 3.2, 3.3 and Lemmas 3.1, 3.2, 3.3 before starting

• Due in class on Thursday, Oct 3rd

• Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates, the teaching assistants and the professor, but you should never misrepresent someone else’s work as your own. If you use any material from external sources, you must provide proper attribution.