**Principal Sources of Optimization**

- **Causes of Redundancy**
  - Redundancy is available at the source level
  - Side effect or programs written in a high-level language
    - $A(i,j)$

**Chapter 9: Machine-Independent Optimizations**

**Semantics-Preserving transformations**

```plaintext
quick sort

void l, j, x, v;
if (n<=m) return;
i=m-1;
j=n;
v=a[n];
while 1
  do i=i+1; while (a[i]<v);
  do j=j-1; while (a[j]=v);
  if (i>=j) break;
  x=a[i];
a[i]=a[j];
a[j]=x;
}
x=a[i];
a[i]=a[n];
a[n]=x;
quicksort(m,j);
quicksort(i+1,n);
```
Introduction to Data-Flow Analysis

- Techniques that derive information about the flow of data along program execution paths
- In general, there is an infinite number of possible execution paths
  - Abstract out certain details and keep only the data we need for the purpose of the analysis

The Data-Flow Analysis Schema

- We denote the data-flow values before and after each statement \( s \) by \( \text{IN}[s] \) and \( \text{OUT}[s] \)
- Data-flow program is to find a solution to a set of constraints on the \( \text{IN}[s] \)'s and \( \text{OUT}[s] \)'s.
  - Constraints based on semantics of statements (transfer functions)
    - Forward-flow: \( \text{OUT}[s] = f(s)(\text{IN}[s]) \)
    - Backward-flow: \( \text{IN}[s] = f(s)(\text{OUT}[s]) \)
  - Constraints based on flow of control
  - Within a basic block: \( \text{IN}[s_{i+1}] = \text{OUT}[s_i] \)

Data-flow Schemas on Basic Blocks

- Let \( B=S_1, \ldots, S_n \)
  - \( \text{IN}[B] = \text{IN}[S_1], \text{OUT}[B] = \text{OUT}[S_n] \)
  - Data-flow is forward
    - \( \text{OUT}[B] = f_d(\text{IN}[B]) \), where \( f_d = f_1 \ldots f_n f_1 \)
    - \( \text{IN}[B] = U_S \alpha \text{predecessor of } B \text{OUT}(P) \)
  - Data-flow is backward
    - \( \text{IN}[B] = f_d(\text{OUT}[B]) \)
    - \( \text{OUT}[B] = U_S \alpha \text{successor of } B \text{IN}[S] \)

Reaching Definitions

- A definition \( d \) reaches a point \( p \) if there is a path from \( d \) to \( p \), such that \( d \) is not killed along the path.
  - A definition of \( x \) is a statement that assigns, or may assign, a value to \( x \)
- Allow inaccuracies, but conservative analysis
  - Aliasing
  - Decide whether each path can be taken is undecidable

Transfer Equations for Reaching Definitions

- Consider a definition \( d: u = u + w \)
  - Generates definition for \( u \) and kills other definitions to \( u \)
    - \( f_d(x) = \text{gen}_U \alpha (x - \text{kill}) \)
  - Transfer functions of a basic block is the composition of the ones of statements
    - \( f_d(x) = \text{gen}_2 U \alpha (x - \text{kill}) \)
    - \( f_d(x) = \text{gen}_2 U \alpha (x - \text{kill}) \)
    - \( f_d(x) = \text{gen}_2 U \alpha (x - \text{kill}) \)
    - \( f_d(x) = \text{gen}_2 U \alpha (x - \text{kill}) \)
Transfer Equations for Reaching Definitions

- Consider a block $B$ with $n$ statements

$$ f_a(x) = \text{gen}_a \cup (x - \text{kill}_a) $$

$$ \text{kill}_a = \text{kill}_1 \cup \text{kill}_2 \cup \ldots \cup \text{kill}_n $$

$$ \text{gen}_a = \text{gen}_1 \cup \text{gen}_2 \cup \ldots \cup \text{gen}_n $$

Control-Flow Equations

- If there is an edge from $P$ to $B$
  - $\text{OUT}[P]$ is a subset of $\text{IN}[B]$
  - $\text{IN}[B] = U \{ P \}$, a predecessor of $B$ $\text{OUT}[P]$

  We refer to union as the meet operator for reaching definitions

Iterative Algorithm for Reaching Definitions

$$ \text{OUT}[\text{ENTRY}] = \emptyset $$

For (each $B$ other than $\text{ENTRY}$)

$$ \text{OUT}[B] = \emptyset $$

While (change to any OUT occur)

for (each $B$ other than $\text{ENTRY}$)

$$ \text{IN}[B] = U \{ P \}$, a predecessor of $B$ $\text{OUT}[P]$

$$ \text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) $$

Live-Variable Analysis

- In live-variable analysis we wish to know for $x$ and point $p$ whether the value of $x$ at $p$ could be used along some path in the flow graph starting at $p$
  - If so, $x$ is live at $p$; otherwise $x$ is dead at $p$
  - Can be used in register allocation
  - $\text{def}_a$ is the set of vars defined in $B$ prior to any use of that var in $B$, and
  - $\text{use}_a$ is the set of vars used in $B$ prior to any definition of that var in $B$
Available Expressions

• Used for detecting global common subexpressions
• An expression $x+y$ is available at $p$ if
  - every path from entry to $p$ evaluates $x+y$
  - after last such evaluation prior to $p$, no subsequent assignment to $x$ or $y$
• A block
  - Kills $x+y$ if it (may) assigns $x$ or $y$ and does not subsequently recompute $x+y$
  - Generates $x+y$ if it evaluates $x+y$ and does not subsequently redefine $x$ or $y$

Iterative Algorithm to Compute Available Expressions

\[
\text{OUT}[\text{ENTRY}]=\Phi
\]
\[
\text{OUT}[B]= \text{All}
\]
\[
\text{While} \ (\text{change to any OUT occur})
\]
\[
\text{for} \ (\text{each B other than ENTRY})
\]
\[
\text{IN}[B]=\bigcap \ P \ a \ predecessor \ of \ B \ \text{OUT}[P]
\]
\[
\text{OUT}[B]=\text{use}_B \bigcap \ (\text{IN}[B]-\text{def}_B)
\]

Foundations of Data-Flow Analysis

• Under what circumstances is the iterative algorithm used in data-flow analysis correct?
• How precise is the solution obtained by the iterative algorithm
• Will the iterative algorithm converge?
• What is the meaning of the solution to the equations?
Framework

• A data-flow analysis framework \((D,V,\land,F)\) consists of
  – A direction of the data flow \(D\), either forward or backward
  – A semilattice that includes values \(V\) and a meet operator \(\land\)
  – A family of transfer functions from \(V\) to \(V\)

Semilattices

• A semilattice is a set \(V\) and a binary meet operator \(\land\) such that for all \(x,y,z\) in \(V\)
  – Idempotent: \(x \land x = x\)
  – Commutative: \(x \land y = y \land x\)
  – Associative: \(x \land (y \land z) = (x \land y) \land z\)
• A top element \(\top\), for all \(x\) in \(V\), \(\top \land x = x\)
• Optionally, a bottom element \(\bot\), for all \(x\) in \(V\), \(\bot \land x = \bot\)

Partial Orders

• A relation \(\leq\) is a partial order if for all \(x,y,z\)
  – \(x \leq x\) (reflexive)
  – If \(x \leq y\) and \(y \leq x\), then \(x=y\) (antisymmetric)
  – If \(x \leq y\) and \(y \leq z\), then \(x \leq z\) (transitive)
• Partial order for a semilattice
  – \(x \leq y\) if and only if \(x \land y = x\)
• A greatest lower bound (glb) of \(x\) and \(y\) is \(g\)
  – \(g \leq x\), \(g \leq y\), and
  – If \(z \leq x\) and \(z \leq y\), then \(z \leq g\)

Iterative Algorithm for a Forward Data-Flow Problem

\[
\text{OUT[ENTRY]} = \Phi
\]
For (each \(B\) other than ENTRY)
\[
\text{OUT}[B] = \top
\]
While (change to any OUT occur)
for (each \(B\) other than ENTRY)
\[
\text{IN}[B] = \land \ P \text{ a predecessor of } B \text{ OUT}[P]
\]
\[
\text{OUT}[B] = f_B \ (\text{IN}[B])
\]

Iterative Algorithm for a Backward Data-Flow Problem

\[
\text{IN[EXIT]} = \Phi
\]
For (each \(B\) other than ENTRY)
\[
\text{IN}[B] = \top
\]
While (change to any IN occur)
for (each \(B\) other than EXIT)
\[
\text{OUT}[B] = \land \ S \text{ a successor of } B \text{ IN}[S]
\]
\[
\text{OUT}[B] = f_B \ (\text{OUT}[B])
\]