**Data Flow Analysis**

Static (compile-time) analysis of how data flows during execution

Undecidable in general for real execution

Prove small facts about program

*Solve system of data flow equations over flow graph*

Determine legality of specific optimizations

*Need conservative answers*

*Want most optimistic solution*

---

**Data Flow Information**

Ex. 

(S1) \( Y := 1 \)

(S2) \( X := K \)

(S3) \( T := \text{FOO}(Z) \cdot Y + \text{FOO}(Z) \)

(S4) \( Y := Y + K \)

Determine if \( X \) a constant in loop

Determine if \( \text{FOO} \) modifies \( Z \)

**Determine if expression** \( \text{FOO}(Z) \cdot Y + \text{FOO}(Z) \)

already computed

Determine if \( Y \) not used later in program
Data Flow Equations

**Local information:**
- $\text{Gen}(B)$: Info generated in block $B$
- $\text{Pres}(B)$: Info preserved through block $B$

**Data Flow Equations:**
$$\text{OUT}(B) = \text{Gen}(B) \cup (\text{In}(B) \cap \text{Pres}(B))$$

Intuitively, info at end of basic block $B$ is either
- Generated within block $B$, or
- Enters at beginning, and is not killed as control flows thru $B$

**Typically assumes all control flow paths may be taken**

**Variations:**
- Info flow forward or backward in CFG
- How Gen, Pres are defined
- How In, Out are initialized

---

Optimizer Structure

```
control flow analysis + CFG + DFA + SSA-based Transformations
```

**Typical**

```
Ast to CFG Analysis
```

---

```
optimized IL
```

---


Reaching Definitions Problem

**Def:** A *definition* of a variable is a possible assignment to the variable. We say that the statement containing the definition *defines* the variable.

**Def:** If a definition *always* assigns to the variable, it *kills* all other definitions. Otherwise, it *preserves* them.

**Ex.** \( X := A \) is a definition, and kills all other defs, of \( X \)

**Def:** A definition \( D \) of \( X \) at node \( B_1 \) *reaches* node \( B_2 \) if there is a path \( p \) from \( B_1 \) to \( B_2 \) such that \( D \) is preserved on path \( p \).

\[
\text{Reachin}(B) = \text{set of defs that reach the entry of node } B
\]

Reaching Definitions Example

---

Diagram:

```
      B1
   /     \
B2-X:=2- B4-X:=3
      |  \
      |   \
      |    B5
```

Def of \( X \) at node \( B_2 \) reaches \( B_4 \) (not \( B_1, B_3, B_5 \))
Def of \( X \) at node \( B_4 \) reaches \( B_5 \) and kills defs at \( B_2, B_3 \)

*Useful for constant propagation, code motion*
Solving Reaching Definitions Problem

Local Analysis:
- \( \text{Gen}(B) \): set of local defs that reach the end of \( B \)
- \( \text{Pres}(B) \): set of defs preserved through \( B \)

Data Flow Equations:
- \( \text{Reachin}(B) \): set of defs that reach entry of \( B \)
- \( \text{Reachout}(B) \): set of defs that reach exit of \( B \)

Initialize to \( \emptyset \) (empty set)

\[
\begin{align*}
\text{Reachin}(B) &= \bigcup_{P \in \text{pred of } B} \text{Reachout}(P) \\
\text{Reachout}(B) &= \text{Gen}(B) \cup (\text{Reachin}(B) \setminus \text{Pres}(B))
\end{align*}
\]

Solution Method: Iterate until no change in Reach sets. (fixed point)

Most optimistic: \( \emptyset \)
Most pessimistic: all defs (satisfies the equations!)

Reaching Defs Example

\begin{tabular}{|c|c|c|c|c|}
\hline
Line & (1) L1:Read(N) & (2) Call FOO(N) & (3) I := 1 & (4) Repeat Until(I > N) \\
\hline
Gen & N1 & N2 & I3 & I6 \\
Kill & N2 & N1 & I4 & I6 \\
Reachin & A5 & A5 & A5 & A5 \\
Reachout & I6 & I6 & I6 & I6 \\
\hline
\end{tabular}

With sets or bit vectors

Takes time to propagate info, entire pass with no change

Termination? Complexity?
Iterative Algorithm for RD Problem

Local Sets:  Compute Gen(B), Pres(B)
Initialization:  Initialize In(B), Out(B) to empty
Worklist Algorithm:
Worklist := Set of all nodes
While(Worklist ≠ 0)
    Remove node N from worklist
    OldOut := Out(N)
    In(N) := \bigcup_{P \text{ pred of } N} Out(P)
    Out(N) := Gen(N) \bigcup \{ In(N) \setminus Pres(N) \}
    if (Out(N) ≠ OldOut) then add N to Worklist

Live Definitions Problem

Def:  A def D of X is live at node N if there is a path p from N to Exit with a use of X that can use the value defined at D. Otherwise, the def is dead.
Def:  The variable X is live at node B if there is def D of X that is live at B.

Live Variables Problem: What variables are live at B?
Useful for storage reuse, register allocation
Live(B) = set of variables live on exit from node B
Ex:
Solving Live Variables Problem

Local Analysis:

- \( \text{Gen}(B) \): set of variables used in B (before def.)
- \( \text{Pres}(B) \): set of variables NOT always redefined in B

Data Flow Equations:

Initialize to \( \phi \)

\[
\text{Liveout}(B) = \bigcup_{S \text{ succ of } B} \text{Livein}(S)
\]

\[
\text{Livein}(B) = \text{Gen}(B) \cup (\text{Liveout}(B) \cap \text{Pres}(B))
\]

Solution Method: Iterate until no change in Live sets.

Most optimistic: \( \phi \)
Most pessimistic: all defs

How different from Reaching Defs problem?

<table>
<thead>
<tr>
<th>Live Vars Example</th>
<th>Gen</th>
<th>Kill</th>
<th>Livein</th>
<th>Liveout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Read(N)</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. B := 2</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. I := 1</td>
<td>I</td>
<td></td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>4. Repeat Until(I &gt; N)</td>
<td>I, N</td>
<td></td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>5. A(I) := A(I) + 1</td>
<td>A, I</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. I := I + 1</td>
<td>I</td>
<td></td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>7. Endrepeat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Print(A(N))</td>
<td>A, N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Print B * N</td>
<td>B, N</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Direction of Flow

**Forward:** Information at node depends on what happens later in flow graph

\[
\text{In}(B) = \bigcup_{P \text{ pred of } B} \text{Out}(P) \\
\text{Out}(B) = \text{Gen}(B) \bigcup (\text{In}(B) \cap \text{Pres}(B))
\]

**Backward:** Information at node depends on what happens earlier in flow graph

\[
\text{In}(B) = \text{Gen}(B) \bigcup (\text{Out}(B) \cap \text{Pres}(B)) \\
\text{Out}(B) = \bigcup_{S \text{ succ of } B} \text{In}(S)
\]

Symmetry of Liveness/Reaching Defs

Swap In and Out, Backward and Forward

**Reaching Defs:**

\[
\text{In}(B) = \bigcup_{P \text{ pred of } B} \text{Out}(P) \\
\text{Out}(B) = \text{Gen}(B) \bigcup (\text{In}(B) \cap \text{Pres}(B))
\]

**Liveness:**

\[
\text{In}(B) = \text{Gen}(B) \bigcup (\text{Out}(B) \cap \text{Pres}(B)) \\
\text{Out}(B) = \bigcup_{S \text{ succ of } B} \text{In}(S)
\]
Other Data Flow Problems

Available Expressions:
An expression \( e \) is \textit{available} at \( B \) if every path to \( B \) contains a computation of \( e \) from \( \text{def} \)s that are live at \( B \).

\textit{Forward}, 1 bit per expression

Upwards Exposed Uses:
Set of uses that may not be defined.

\textit{Backward}, 1 bit per expression

Partially Redundant Expressions:
Set of expressions appearing at least twice on some path, without its \( \text{operands} \) being modified between occurrences of the expression.

\textit{Bidirectional}, 1 bit per expression

Available Expressions

\textbf{Local Analysis:}

\( \text{Gen}(B) \): set of expressions generated in \( B \)

\( \text{Kill}(B) \): set of expressions killed in \( B \)

\textbf{Data Flow Equations:}

\( \text{In}(B) \): set of expressions that reach entry of \( B \)

\( \text{Out}(B) \): set of expressions that reach exit of \( B \)

Initialize to

\( \text{In}(B) = \)

\( \text{Out}(B) = \)
Must vs May Information

Must: Implies a guarantee
May: Identifies possibility

Liveness is may: there is a path on which variable is live
Reaching Def? Available Exp?

<table>
<thead>
<tr>
<th>Must:</th>
<th>May:</th>
</tr>
</thead>
<tbody>
<tr>
<td>desired info safe</td>
<td>small set larger</td>
</tr>
<tr>
<td>Gen Kill</td>
<td>all may guaranteed</td>
</tr>
<tr>
<td>merge initialization</td>
<td>empty set all</td>
</tr>
<tr>
<td>large set smaller only must might be</td>
<td></td>
</tr>
</tbody>
</table>