**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**

\[
\begin{align*}
(1) & \quad Y = 1 \\
(2) & \quad X := K \\
(3) & \quad T := \text{FOO}(Z) \cdot Y + \text{FOO}(Z) \\
(4) & \quad Y := Y + K
\end{align*}
\]

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

Typical

Ast to CFG | Analysis + SSA | SSA-based Transformations

Scale
**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 \( X \). \( \text{Foo}(Z) \) may define \( X \) but preserves all 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**

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:
- Gen(B): set of local defs that reach the end of B
- Pres(B): set of defs preserved through B

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

Initialize to \( \emptyset \) (empty set)
- Reachin(B) = \( \bigcup_{P \in pred(B)} \text{Reachout}(P) \)
- Reachout(B) = Gen(B) \( \cup \) (Reachin(B) \( \setminus \) Pres(B))

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

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

Reaching Defs Example

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
<th>Reachin</th>
<th>Reachout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L1: Read(N)</td>
<td>N1</td>
<td>N2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Call FOO()</td>
<td>N2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>I := 1</td>
<td>I3</td>
<td>I6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Repeat Until(i &gt; N)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A0 := A0 + 1</td>
<td>A5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>I := I + 1</td>
<td>I5</td>
<td>I3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Endrepeat</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Print(A(N))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Goto L1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

\[ \text{Worklist} := \text{Set of all nodes} \]

\[ \text{While}(\text{Worklist} \neq \emptyset) \]

Remove node \( N \) from worklist
\[ \text{OldOut} := \text{Out}(N) \]
\[ \text{In}(N) := \bigcup_{P \in \text{pred of } N} \text{Out}(P) \]
\[ \text{Out}(N) := \text{Gen}(N) \bigcup \text{In}(N) \bigcap \text{Pres}(N) \]

if \( \text{Out}(N) \neq \text{OldOut} \) 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

Liveout(B) = set of variables live on exit from node \( B \)

Ex:
Solving Live Variables Problem

Local Analysis:
- $Gen(B)$: set of variables used in B (before def.)
- $Pres(B)$: set of variables NOT always redefined in B

Data Flow Equations:
Initialize to $\phi$

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

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

Solution Method: Iterate until no change in Live sets.

Most optimistic: $\phi$
Most pessimistic: all defs

How different from Reaching Defs problem?

Live Vars Example

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

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

\[
\begin{align*}
\text{In}(B) &= \bigcup \text{Out}(P) \\
\text{Out}(B) &= \text{Gen}(B) \bigcup (\text{In}(B) \cap \text{Pres}(B))
\end{align*}
\]

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

\[
\begin{align*}
\text{In}(B) &= \text{Gen}(B) \bigcup (\text{Out}(B) \cap \text{Pres}(B)) \\
\text{Out}(B) &= \bigcup \text{In}(S) \\
&\quad \text{S succ of B}
\end{align*}
\]

Symmetry of Liveness/Reaching Defs

Swap In and Out, Backward and Forward

**Reaching Defs:**

\[
\begin{align*}
\text{In}(B) &= \bigcup \text{Out}(P) \\
\text{Out}(B) &= \text{Gen}(B) \bigcup (\text{In}(B) \cap \text{Pres}(B))
\end{align*}
\]

**Liveness:**

\[
\begin{align*}
\text{In}(B) &= \text{Gen}(B) \bigcup (\text{Out}(B) \cap \text{Pres}(B)) \\
\text{Out}(B) &= \bigcup \text{In}(S) \\
&\quad \text{S succ of B}
\end{align*}
\]
Other Data Flow Problems

Available Expressions:
An expression e is available at B if every path to B contains a computation of e fromdefs that are live at B.

Forward, 1 bit per expression

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

Backward, 1 bit per expression

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

Bidirectional, 1 bit per expression

Available Expressions

Local Analysis:
Gen(B): set of expressions generated in B
Kill(B): set of expressions killed in B

Data Flow Equations:
In(B): set of expressions that reach entry of B
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>desired info</th>
<th>small set</th>
<th>Must: large set</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>larger</td>
<td>smaller</td>
</tr>
<tr>
<td>Gen</td>
<td>all may</td>
<td>only must</td>
</tr>
<tr>
<td>Kill</td>
<td>guaranteed</td>
<td>might be</td>
</tr>
<tr>
<td>merge</td>
<td>∪</td>
<td>∩</td>
</tr>
<tr>
<td>initialization</td>
<td>empty set</td>
<td>all</td>
</tr>
</tbody>
</table>

What is common?

Flow graph: CFG
Local Analysis: Gen, Pres
Meet Function: ∪
Direction in flow graph: forward, backward
Data Flow Equations:

\[ ln(B) = \bigcup_P OUT(P) \]
\[ out(B) = GEN(B) \bigcup (ln(B) \bigcap pres(B)) \]

General case function:

\[ OUT(B) = (f_B ln(B)) \]