CSE 5317

Lecture 17: More dataflow analysis
30 Mar 2010

Nate Nystrom
University of Texas at Arlington
Live Variable Analysis

What are the live variables at each program point?

Method:
1. Define sets of live variables
2. Build constraints
3. Solve constraints
Derive Constraints

Constraints for each instruction:

\[ \text{in}[I] = (\text{out}[I] - \text{def}[I]) \cup \text{use}[I] \]

Constraints for control flow:

\[ \text{out}[B] = \bigcup_{B' \in \text{succ}(B)} \text{in}[B'] \]
Derive Constraints

\[
\begin{align*}
L_1 &= L_2 \cup \{c\} \\
L_2 &= L_3 \cup L_{11} \\
L_3 &= (L_4-\{x\}) \cup \{y\} \\
L_4 &= (L_5-\{y\}) \cup \{z\} \\
L_5 &= L_6 \cup \{d\} \\
L_6 &= L_7 \cup L_9 \\
L_7 &= (L_8-\{x\}) \cup \{y,z\} \\
L_8 &= L_9 \\
L_9 &= L_{10}-\{z\} \\
L_{10} &= L_1 \\
L_{11} &= (L_{12}-\{z\}) \cup \{x\}
\end{align*}
\]
Initialization

\begin{align*}
L_1 &= L_2 \cup \{c\} \\
L_2 &= L_3 \cup L_{11} \\
L_3 &= (L_4-\{x\}) \cup \{y\} \\
L_4 &= (L_5-\{y\}) \cup \{z\} \\
L_5 &= L_6 \cup \{d\} \\
L_6 &= L_7 \cup L_9 \\
L_7 &= (L_8-\{x\}) \cup \{y,z\} \\
L_8 &= L_9 \\
L_9 &= L_{10}-\{z\} \\
L_{10} &= L_1 \\
L_{11} &= (L_{12}-\{z\}) \cup \{x\} \\
\end{align*}

\begin{align*}
\text{if (c)} \\
x &= y+1 \\
y &= 2*z \\
\text{if (d)} \\
x &= y+z \\
z &= 1 \\
z &= x \\
\end{align*}

\begin{align*}
L_1 &= \{\} \\
L_2 &= \{\} \\
L_3 &= \{\} \\
L_4 &= \{\} \\
L_5 &= \{\} \\
L_6 &= \{\} \\
L_7 &= \{\} \\
L_8 &= \{\} \\
L_9 &= \{\} \\
L_{10} &= \{\} \\
L_{11} &= \{\} \\
L_{12} &= \{\} \\
\end{align*}
Iteration 1

\[ L_1 = L_2 \cup \{c\} \]
\[ L_2 = L_3 \cup L_{11} \]
\[ L_3 = (L_4 - \{x\}) \cup \{y\} \]
\[ L_4 = (L_5 - \{y\}) \cup \{z\} \]
\[ L_5 = L_6 \cup \{d\} \]
\[ L_6 = L_7 \cup L_9 \]
\[ L_7 = (L_8 - \{x\}) \cup \{y, z\} \]
\[ L_8 = L_9 \]
\[ L_9 = L_{10} - \{z\} \]
\[ L_{10} = L_1 \]
\[ L_{11} = (L_{12} - \{z\}) \cup \{x\} \]

If (c)

\[ x = y + 1 \]
\[ y = 2z \]

If (d)

\[ x = y + z \]

\[ z = 1 \]

\[ z = x \]

\[ L_1 = \{x, y, z, c, d\} \]
\[ L_2 = \{x, y, z, d\} \]
\[ L_3 = \{y, z, d\} \]
\[ L_4 = \{z, d\} \]
\[ L_5 = \{y, z, d\} \]
\[ L_6 = \{y, z\} \]
\[ L_7 = \{y, z\} \]
\[ L_8 = \{\} \]
\[ L_9 = \{\} \]
\[ L_{10} = \{\} \]
\[ L_{11} = \{x\} \]
\[ L_{12} = \{\} \]
Iteration 2

\[
\begin{align*}
L_1 &= L_2 \cup \{c\} \\
L_2 &= L_3 \cup L_{11} \\
L_3 &= (L_4-\{x\}) \cup \{y\} \\
L_4 &= (L_5-\{y\}) \cup \{z\} \\
L_5 &= L_6 \cup \{d\} \\
L_6 &= L_7 \cup L_9 \\
L_7 &= (L_8-\{x\}) \cup \{y,z\} \\
L_8 &= L_9 \\
L_9 &= L_{10}-\{z\} \\
L_{10} &= L_1 \\
L_{11} &= (L_{12}-\{z\}) \cup \{x\} \\
L_{12} &= \{} \\
L_1 &= \{x,y,z,c,d\} \\
L_2 &= \{x,y,z,c,d\} \\
L_3 &= \{y,z,c,d\} \\
L_4 &= \{x,z,c,d\} \\
L_5 &= \{x,y,z,c,d\} \\
L_6 &= \{x,y,z,c,d\} \\
L_7 &= \{y,z,c,d\} \\
L_8 &= \{x,y,c,d\} \\
L_9 &= \{x,y,c,d\} \\
L_{10} &= \{x,y,z,c,d\} \\
L_{11} &= \{x\} \\
L_{12} &= \{}
\end{align*}
\]
Fixed-point!

\( L_1 = L_2 \cup \{c\} \)
\( L_2 = L_3 \cup L_{11} \)
\( L_3 = (L_4 - \{x\}) \cup \{y\} \)
\( L_4 = (L_5 - \{y\}) \cup \{z\} \)
\( L_5 = L_6 \cup \{d\} \)
\( L_6 = L_7 \cup L_9 \)
\( L_7 = (L_8 - \{x\}) \cup \{y,z\} \)
\( L_8 = L_9 \)
\( L_9 = L_{10} - \{z\} \)
\( L_{10} = L_1 \)
\( L_{11} = (L_{12} - \{z\}) \cup \{x\} \)

- \( \text{if } (c) \)
  - \( x = y + 1 \)
  - \( y = 2z \)
  - \( \text{if } (d) \)
  - \( x = y + z \)

\( z = 1 \)
\( z = x \)

- \( L_1 = \{x,y,z,c,d\} \)
- \( L_2 = \{x,y,z,c,d\} \)
- \( L_3 = \{y,z,c,d\} \)
- \( L_4 = \{x,z,c,d\} \)
- \( L_5 = \{x,y,z,c,d\} \)
- \( L_6 = \{x,y,z,c,d\} \)
- \( L_7 = \{y,z,c,d\} \)
- \( L_8 = \{x,y,c,d\} \)
- \( L_9 = \{x,y,c,d\} \)
- \( L_{10} = \{x,y,z,c,d\} \)
- \( L_{11} = \{x\} \)
- \( L_{12} = \{\} \)
Final Result

Final result: sets of live variables at each program point

L_1 = \{x, y, z, c, d\}
L_2 = \{x, y, z, c, d\}
L_3 = \{y, z, c, d\}
L_4 = \{x, z, c, d\}
L_5 = \{x, y, z, c, d\}
L_6 = \{x, y, z, c, d\}
L_7 = \{y, z, c, d\}
L_8 = \{x, y, c, d\}
L_9 = \{x, y, c, d\}
L_{10} = \{x, y, z, c, d\}
L_{11} = \{x\}
L_{12} = \{\}

x live here!
Characterize All Executions

The analysis detects that there is an execution which uses the value \( x = y + 1 \)

\[ \begin{align*}
\text{if (c)} & : \\
x &= y + 1 \\
y &= 2 \times z \\
\text{if (d)} & :\\
x &= y + z \\
z &= 1 \\
z &= x
\end{align*} \]

\( L_1 = \{ x, y, z, c, d \} \)
\( L_2 = \{ x, y, z, c, d \} \)
\( L_3 = \{ y, z, c, d \} \)
\( L_4 = \{ x, z, c, d \} \)
\( L_5 = \{ x, y, z, c, d \} \)
\( L_6 = \{ x, y, z, c, d \} \)
\( L_7 = \{ y, z, c, d \} \)
\( L_8 = \{ x, y, c, d \} \)
\( L_9 = \{ x, y, c, d \} \)
\( L_{10} = \{ x, y, z, c, d \} \)
\( L_{11} = \{ x \} \)
\( L_{12} = \{ \} \)
Generalization

- Live variable analysis and detection of available copies are similar:
  - Define some information that they need to compute
  - Build constraints for the information
  - Solve constraints iteratively:
    - The information always “increases” during iteration
    - Eventually, it reaches a fixed point.

- We would like a general framework
  - Framework applicable to many other analyses
  - Live variable/copy propagation = instances of the framework
Dataflow Analysis Framework

- **Dataflow analysis** = a common framework for many compiler analyses
  - Computes some information at each program point
  - The computed information characterizes all possible executions of the program

- **Basic methodology:**
  - Describe information about the program using an algebraic structure called **lattice**
  - Build constraints which show how instructions and control flow modify the information in the lattice
  - Iteratively solve constraints
Lattices and Partial Orders

- Lattice definition uses the concept of partial order relation

- A partial order \((P, \sqsubseteq)\) consists of:
  - A set \(P\)
  - A partial order relation \(\sqsubseteq\) which is:
    1. Reflexive \(x \sqsubseteq x\)
    2. Anti-symmetric \(x \sqsubseteq y, y \sqsubseteq x \Rightarrow x = y\)
    3. Transitive: \(x \sqsubseteq y, y \sqsubseteq z \Rightarrow x \sqsubseteq z\)

- Called “partial order” because not all elements are comparable
Lattices and Lower/Upper Bounds

- Lattice definition uses the concept of lower and upper bounds

- If \((P, \sqsubseteq)\) is a partial order and \(S \subseteq P\), then:
  1. \(x \in P\) is a lower bound of \(S\) if \(x \sqsubseteq y\), for all \(y \in S\)
  2. \(x \in P\) is an upper bound of \(S\) if \(y \sqsubseteq x\), for all \(y \in S\)

- There may be multiple lower and upper bounds of the same set \(S\)
LUB and GLB

• Define least upper bounds (LUB) and greatest lower bounds (GLB)

• If \((P, \sqsubseteq)\) is a partial order and \(S \subseteq P\), then:
  1. \(x \in P\) is GLB of \(S\) if:
     a) \(x\) is an lower bound of \(S\)
     b) \(y \sqsubseteq x\), for any lower bound \(y\) of \(S\)

  2. \(x \in P\) is a LUB of \(S\) if:
     a) \(x\) is an upper bound of \(S\)
     b) \(x \sqsubseteq y\), for any upper bound \(y\) of \(S\)

• ... are GLB and LUB unique?
Lattices

• A pair \((L, \subseteq)\) is a lattice if:
  1. \((L, \subseteq)\) is a partial order
  2. Any finite subset \(S \subseteq L\) has a LUB and a GLB

• Can define two operators in lattices:
  1. Meet operator: \(x \sqcap y = \text{GLB}\{x, y\}\)
  2. Join operator: \(x \sqcup y = \text{LUB}\{x, y\}\)

• Meet and join are well-defined for lattices
Complete Lattices

- A pair \((L, \sqsubseteq)\) is a complete lattice if:
  1. \((L, \sqsubseteq)\) is a partial order
  2. Any subset \(S \subseteq L\) has a LUB and a GLB

- Can define meet and join operators

- Can also define two special elements:
  1. Bottom element: \(\bot = \text{GLB}(L)\)
  2. Top element: \(\top = \text{LUB}(L)\)

- All finite lattices are complete
Example Lattice

• Consider $S = \{a,b,c\}$ and its power set $P = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{b,c\}, \{a,c\} \{a,b,c\}\}$

• Define partial order as set inclusion: $X \subseteq Y$
  – Reflexive $X \subseteq Y$
  – Anti-symmetric $X \subseteq Y, Y \subseteq X \Rightarrow X = Y$
  – Transitive $X \subseteq Y, Y \subseteq Z \Rightarrow X \subseteq Z$

• Also, for any two elements of $P$, there is a set which includes both and another set which is included in both

• Therefore $(P, \subseteq)$ is a (complete) lattice
Hasse Diagrams

- **Hasse diagram** = graphical representation of a lattice where $x$ is below $y$ when $x \subseteq y$ and $x \neq y$
Power Set Lattice

- Partial order: \( \subseteq \)
  (set inclusion)
- Meet: \( \cap \)
  (set intersection)
- Join: \( \cup \)
  (set union)
- Top element: \( \{a,b,c\} \)
  (whole set)
- Bottom element: \( \emptyset \)
  (empty set)
Reversed Lattice

- **Partial order**: \( \supseteq \)
  (set inclusion)
- **Meet**: \( \cup \)
  (set union)
- **Join**: \( \cap \)
  (set intersection)
- **Top element**: \( \emptyset \)
  (empty set)
- **Bottom element**: \( \{a,b,c\} \)
  (whole set)
Relation To Dataflow Analysis

- Information computed by live variable analysis and available copies can be expressed as elements of lattices

- **Live variables**: if \( V \) is the set of all variables in the program and \( P \) the power set of \( V \), then:
  - \((P, \subseteq)\) is a lattice
  - sets of live variables are elements of this lattice
Relation To Analysis of Programs

• Copy Propagation:
  - V is the set of all variables in the program
  - V x V the cartesian product representing all possible copy instructions
  - P the power set of V x V

• Then:
  - (P, ⊆) is a lattice
  - sets of available copies are lattice elements
More About Lattices

• In a lattice \((L, \sqsubseteq)\), the following are equivalent:
  1. \(x \sqsubseteq y\)
  2. \(x \sqcap y = x\)
  3. \(x \sqcup y = y\)

• Note: meet and join operations were defined using the partial order relation
Proof

- Prove that $x \subseteq y$ implies $x \cap y = x$:
  - $x$ is a lower bound of $\{x,y\}$
  - All lower bounds of $\{x,y\}$ are less than $x,y$
  - In particular, they are less than $x$

- Prove that $x \cap y = x$ implies $x \subseteq y$:
  - $x$ is a lower bound of $\{x,y\}$
  - $x$ is less than $x$ and $y$
  - In particular, $x$ is less than $y$
Proof

• Prove that $x \subseteq y$ implies $x \cup y = y$:
  – $y$ is an upper bound of \{x,y\}
  – All upper bounds of \{x,y\} greater than $x,y$
  – In particular, they are greater than $y$

• Prove that $x \cup y = y$ implies $x \subseteq y$:
  – $y$ is a upper bound of \{x,y\}
  – $y$ is greater than $x$ and $y$
  – In particular, $y$ is greater than $x$
Properties of Meet and Join

- The meet and join operators are:
  1. **Associative**: \((x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)\)
  2. **Commutative**: \(x \sqcap y = y \sqcap x\)
  3. **Idempotent**: \(x \sqcap x = x\)

- **Property**: If \(\sqcap\) is an associative, commutative, and idempotent operator, then the relation \(\sqsubseteq\) defined as \(x \sqsubseteq y\) iff \(x \sqcap y = x\) is a partial order.

- Above property provides an alternative definition of a partial orders and lattices starting from the meet (join) operator.
Using Lattices

• Assume information we want to compute in a program is expressed using a lattice L

• To compute the information at each program point we need to:
  – Determine how each instruction in the program changes the information in the lattice
  – Determine how lattice information changes at join/split points in the control flow
Transfer Functions

- Dataflow analysis defines a transfer function $F : L \rightarrow L$ for each instruction in the program.
- Describes how the instruction modifies the information in the lattice.
- Consider $\text{in}[I]$ is information before $I$, and $\text{out}[I]$ is information after $I$.
  - Forward analysis: $\text{out}[I] = F(\text{in}[I])$
  - Backward analysis: $\text{in}[I] = F(\text{out}[I])$
Basic Blocks

- Can extend the concept of transfer function to basic blocks using function composition

- Consider:
  - Basic block $B$ consists of instructions $(I_1, \ldots, I_n)$ with transfer functions $F_1, \ldots, F_n$
  - $\text{in}[B]$ is information before $B$
  - $\text{out}[B]$ is information after $B$

- **Forward analysis:** $\text{out}[B] = F_n(\ldots(F_1(\text{in}[B])))$
- **Backward analysis:** $\text{in}[I] = F_1(\ldots(F_n(\text{out}[I])))$
Split/Join Points

- Dataflow analysis uses meet/join operations at split/join points in the control flow

- Consider in[B] is lattice information at beginning of block B and out[B] is lattice information at end of B

- Forward analysis: \( \text{in}[B] = \sqcap \{ \text{out}[B'] \mid B' \in \text{pred}(B) \} \)

- Backward analysis: \( \text{out}[B] = \sqcap \{ \text{in}[B'] \mid B' \in \text{succ}(B) \} \)

- Can alternatively use join operation \( \sqcup \) (equivalent to using the meet operation \( \sqcap \) in the reversed lattice)
Lecture 18: More dataflow analysis
30 Mar 2010

Nate Nystrom
University of Texas at Arlington
Dataflow analysis

Collection of technique for compile-time reasoning about the runtime flow of values

Used to prove safety and to identify optimization opportunities

Usually formulated as a set of simultaneous equations

- lattice elements (usually sets) associated with program points

Desired result is usually meet over all paths solution

- “what is try on every path from entry?”
- “can this happen on any path from the entry?”
Set up

Build control flow graph

Define equations over a \textit{lattice} (technically a \textit{semilattice} will do)

Equations treated as \textit{transfer functions}

Properties of lattice and transfer functions determine termination and correctness

Solve equations to compute dataflow information at each point in the program
Classifying dataflow analyses

Forward vs. backward
- Forward: information flows from function entry to function exit
- Backward: information flows from function exit to function entry

May vs. must
- “this fact may be true”
- “this fact must be true”

- Analysis must be conservative—cannot transform code in a way that breaks semantics
- Depending on what analysis is used, need to know if something may be true or or if something must be true
Classifying dataflow analyses

Is liveness analysis forward or backward?
Copy propagation analysis?

Is liveness analysis a may-analysis or a must-analysis?
Copy propagation analysis?
Set augmented with a partial relation $\sqsubseteq$
- Each subset has a LUB ($\sqcup$) and GLB ($\sqcap$)
- Can define: meet $\sqcap$, join $\sqcup$, top $\top$, bottom $\bot$

Use lattice in compiler to express information about the program

To compute: build equations that describe how the lattice information changes
- effects of instructions: transfer functions
- effects of control flow: meet operation ($\sqcap$)
Transfer functions

Let $L =$ dataflow information lattice

For each instruction $I$: $F_I : L \rightarrow L$
- describes how $I$ modifies the lattice
- if $\text{in}[I]$ is info before $I$ and $\text{out}[I]$ is info after $I$:
  - forward analysis: $\text{out}[I] = F_I(\text{in}[I])$
  - backward analysis: $\text{in}[I] = F_I(\text{out}[I])$

For each basic block $B$: $F_B : L \rightarrow L$
- composition of transfer functions for each instruction in $B$
  - $F_B(x) = F_{I_1}(F_{I_2}(...(F_{I_n}(x))))$
Properties of transfer functions

Monotonicity
- $x \subseteq y$ implies $F(x) \subseteq F(y)$

Distributivity
- $F(x \cap y) = F(x) \cap F(y)$

Theorem:
- $F$ is monotonic iff $F(x \cap y) \subseteq F(x) \cap F(y)$
- Hence, $F$ distributive $\implies F$ monotonic

Easy to show that $L_1 \times L_2$ preserves monotonicity and distributivity
Meet operation

Models how to combine information at join points in the CFG

If in[B] is info before B  out[B] is info after B, then:

- forward analysis: in[B] = \{ out[B'] | B' ∈ pred(B) \}
- backward analysis: out[B] = \{ in[B'] | B' ∈ succ(B) \}
Forward analysis

CFG G with entry (start) node BS

Lattice \((L, \sqsubseteq)\) represents information at each program

- meet operator \(\sqcap\), top element

Monotonic transfer function

- Transfer function for each instruction: \(F_I: L \rightarrow L\)
- Can derive \(F_B\) for basic blocks

Goal: compute information at each program point, given information at entry of BS is \(X_0\)

Require the solution to satisfy:

- \(\text{out}[B] = F_B(\text{in}[B])\) for all \(B\)
- \(\text{in}[B] = \sqcap \{ \text{out}[B'] \mid B' \in \text{pred}(B) \}\), for all \(B\)
- \(\text{in}[B_S] = X_0\)
Backward analysis

CFG $G$ with exit node $B_e$

Lattice $(L, \sqsubseteq)$ represents information at each program point
- meet operator $\sqcap$, top element

Monotonic transfer function
- Transfer function for each instruction: $F_I : L \rightarrow L$
- Can derive $F_B$ for basic blocks

Goal: compute information at each program point, given information at entry of $B_e$ is $X_0$

Require the solution to satisfy:
- $\text{in}[B] = F_B(\text{out}[B])$ for all $B$
- $\text{out}[B] = \sqcap \{ \text{in}[B'] \mid B' \in \text{succ}(B) \}$, for all $B$
- $\text{out}[B_e] = X_0$
Solving dataflow equations

Given equations:
- out[B] = F_{B}(in[B]) for all B
- in[B] = \cap \{ out[B'] | B' \in \text{pred}(B) \}, for all B
- in[B_s] = X_0

Solve equations using iterative algorithm:
- initialize in[B] = out[B] = T
- initialize in[B_s] = X_0
- Repeatedly apply rules
- Stop when fixed point reached
Solving dataflow equations

Given equations:
- $\text{out}[B] = F_B(\text{in}[B])$ for all $B$
- $\text{in}[B] = \bigcap \{ \text{out}[B'] \mid B' \in \text{pred}(B) \}$, for all $B$
- $\text{in}[B_S] = X_0$

Solve equations using iterative algorithm:
- Initialize $\text{in}[B_S] = X_0$
- Initialize $\text{out}[B] = \top$ for all $B$
- Repeatedly apply rules
- Stop when *fixed point* reached
Algorithm

\[
in[B_s] = X_0 \\
\text{out}[B] = T \text{ for all } B
\]

Repeat until no change:

\[
\text{for each } B \neq B_s: \\
in[B] = \cap \{ \text{out}[B'] | B' \in \text{pred}(B) \}
\]

\[
\text{for each } B: \\
\text{out}[B] = F_B(\text{in}[B])
\]
Algorithm is inefficient

Transfer function for a block re-evaluated even if in[B] not changed

Better: re-evaluate only if necessary

Worklist algorithm:
- Keep list of blocks to evaluate
- Initialize worklist to set of all basic blocks
- If out[B] changes after evaluating out[B] = F_B(in[B]), add succ(B) to the worklist
Worklist algorithm

\[ \text{in}[B_s] = X_0 \]
\[ \text{out}[B] = \top \text{ for all } B \]
\[ \text{worklist} = [ B_1, \ldots, B_n ] \]

while (worklist \neq []):

    B = remove first element from worklist
    \[ \text{in}[B] = \cap \{ \text{out}[B'] \mid B' \in \text{pred}(B) \} \]
    \[ \text{out}[B] = F_B(\text{in}[B]) \]
    if out[B] changed:
        worklist += succ(B)
Observation:
- At each iteration, information decreases in the lattice
- $\text{in}_{k+1}[B] \sqsubseteq \text{in}_k[B]$ 
- $\text{out}_{k+1}[B] \sqsubseteq \text{out}_k[B]$

Information forms a \textit{chain}: $\text{in}_1[B] \sqsupseteq \text{in}_2[B] \sqsupseteq \ldots \sqsupseteq \text{in}_n[B]$
Chains

A chain in a lattice $L$ is a totally ordered subset $S$ of $L$:
- $x \sqsubseteq y$ or $y \sqsubseteq x$ for any $x, y$ in $S$

Height of lattice = size of its largest chain
Lattice with finite height => only finite chains

If transfer functions are monotonic, algorithm will iterate at most the size of the largest chain
=> algorithm will terminate if lattice has finite height
Even better

Compute strongly connected components (SCCs) first

Before processing a node in an SCC, ensure all predecessors have been processed first

Can do this by visiting nodes in reverse postorder

- Only looping will be within an SCC
- Will never revisit an SCC
Multiple solutions

Solution to dataflow equations might not be

Example: live variables

Equations:
- $I_1 = I_2 - \{y\}$
- $I_3 = (I_4 - \{x\}) \cup \{y\}$
- $I_2 = I_1 \cup I_3$
- $I_4 = \{x\}$

Soln 1: $I_1 = \{\}$, $I_2 = \{y\}$, $I_3 = \{y\}$, $I_4 = \{x\}$
Soln 2: $I_1 = \{x\}$, $I_2 = \{x,y\}$, $I_3 = \{y\}$, $I_4 = \{x\}$
Sets of live variables must include every variable that will be used later in the program
... but may also include variables that won’t be used

Analysis is safe if it takes into account all possible executions of the program
- may include cases that never occur in any execution
- conservative approximation of all possible executions
Precision

Dataflow equations guarantee a **safe** solution

Precision: a (safe) solution is more **precise** if it is less conservative

Liveness:
- More precise if sets of live variables is smaller
- A solution that reports all variables are live at all points is safe, but not precise

For any lattice \( L \):
- less precise if lower in the lattice
- \( S_1 \sqsubseteq S_2 \Rightarrow S_1 \) is **less precise** than \( S_2 \)
Maximal fixed point solution

Among all possible solutions of the equations, the solution computed by the iterative algorithm is most precise.

Intuition:
- start at top (most precise but unsafe) and move down, stopping when safe solution found

Called the **maximal fixed point solution (MFP)**

For any solution FP of the dataflow equations, $FP \subseteq MFP$
More precise than MFP?

If we consider all paths through the program
And compute a solution for each path rather than each program point

Can get a more precise solution than MFP

- MFP ⊑ MOP

Called meet over paths (MOP)

But exponential number of paths in the program (and infinite with loops) => intractable
But...

If transfer functions are distributive:

- $\text{MOP} = \text{MFP}$

Is MOP the ideal solution?

No, because it considers paths that might not occur at run time. Ideal solution is undecidable.
Summary

Dataflow analysis

- set up equations
- compute MFP solution
- terminates if transfer function monotonic, lattice finite height

MFP is safe solution. There are other safe solutions:

- $FP \subseteq MFP \subseteq MOP \subseteq IDEAL$

MFP = MOP if transfer functions distributive

Solving MOP and IDEAL directly intractable
Examples

Liveness

- backward
- may-analysis
- \( L = \) sets of live variables
- \( \mathcal{E} = \emptyset \)
- \( \square = \cup \)
- \( \top = \emptyset \)
Examples

Copy propagation analysis
- forward
- must-analysis
- $L = \text{sets of pairs of variables}$
- $\exists = \subseteq$
- $\cap = \cap$
- $\top = \text{set of all pairs of variables}$
Available expressions

An expression is *available* on entry to B if along every path from the entry node to B, it has been computed and none of its subexpressions have been modified.
Using available expressions

Common subexpression elimination:

If \( x+y \in \text{Avail}(I) \), then compiler can replace any occurrence of \( x+y \) in \( I \) with a reference to the previously computed value

[Cocke, 1970]
Can reuse previously computed \( x+y \) if it’s available:
Available expressions

Forward or backward?
Available expressions

Forward or backward?
- forward
Available expressions

Forward or backward?
- forward

May or must?
Available expressions

Forward or backward?
  - forward

May or must?
  - must
Available expressions

Forward or backward?
- forward

May or must?
- must

Lattice set L?
Available expressions

Forward or backward?
- forward

May or must?
- must

Lattice set L?
- L = set of expressions
Available expressions

Forward or backward?
- forward

May or must?
- must

Lattice set $L$?
- $L = \text{set of expressions}$

Ordering $\sqsubseteq$?
Available expressions

Forward or backward?
- forward

May or must?
- must

Lattice set L?
- \( L = \text{set of expressions} \)

Ordering \( \sqsubseteq \)?
- \( \sqsubseteq = \subseteq \)
Available expressions

Forward or backward?
- forward

May or must?
- must

Lattice set L?
- \( L = \) set of expressions

Ordering \( \sqsubseteq \)?
- \( \sqsubseteq = \subseteq \)

Meet operator \( \sqcap \)?
Available expressions

Forward or backward?
- forward

May or must?
- must

Lattice set L?
- L = set of expressions

Ordering $\sqsubseteq$?
- $\sqsubseteq = \subseteq$

Meet operator $\sqcap$?
- $\sqcap = \cap$
Available expressions

Forward or backward?
- forward

May or must?
- must

Lattice set $L$?
- $L = \text{set of expressions}$

Ordering $\sqsubseteq$?
- $\sqsubseteq = \subseteq$

Meet operator $\sqcap$?
- $\sqcap = \cap$

Top element $\top$?
Available expressions

Forward or backward?
- forward

May or must?
- must

Lattice set $L$?
- $L = \text{set of expressions}$

Ordering $\sqsubseteq$?
- $\sqsubseteq = \subseteq$

Meet operator $\sqcap$?
- $\sqcap = \cap$

Top element $\top$?
- $\top = \text{set of all expressions}$
Available expressions

AvailIn[B] = \bigcap \{ \text{AvailOut}[B'] \mid B' \in \text{pred}(B) \} 

AvailOut[I] = (\text{AvailIn}[I] \cup \text{Gen}[I]) - \text{Kill}[I] 

Kill[I] = \text{set of expressions killed in I} 

Gen[I] = \text{set of expressions generated at I} 

Gen[x = y + z] = \{ y + z \} 

Gen[x = y] = \{ \} 

Kill[x = \ldots] = \{ e \mid e \text{ contains } x \}
Very busy expressions

An expression $x+y$ is very busy at point $p$ if every path from $p$, there is an expression $x+y$ before the redefinition of either $x$ or $y$.

i.e., guaranteed that the expression will be computed later

Why useful?
- code hoisting
- if very busy and invariant in a loop, move it out of the loop
Very busy expressions
Very busy expressions

Forward or backward?
Very busy expressions

Forward or backward?
  ▪ backward
Very busy expressions

Forward or backward?

- backward

May or must?
Very busy expressions

Forward or backward?
- backward

May or must?
- must
Very busy expressions

Forward or backward?
- backward

May or must?
- must

Lattice set L?
Very busy expressions

Forward or backward?
  - backward

May or must?
  - must

Lattice set $L$?
  - $L = \text{set of expressions}$
Very busy expressions

Forward or backward?
- backward

May or must?
- must

Lattice set $L$?
- $L = \text{set of expressions}$

Ordering $\sqsubseteq$?
Very busy expressions

Forward or backward?
- backward

May or must?
- must

Lattice set L?
- \( L = \text{set of expressions} \)

Ordering \( \sqsubseteq \)?
- \( \sqsubseteq = \subseteq \)
Very busy expressions

Forward or backward?
- backward

May or must?
- must

Lattice set L?
- \( L = \text{set of expressions} \)

Ordering \( \sqsubseteq \)?
- \( \sqsubseteq = \subseteq \)

Meet operator \( \sqcap \)?
Very busy expressions

Forward or backward?
  ▪ backward

May or must?
  ▪ must

Lattice set $L$?
  ▪ $L = \text{set of expressions}$

Ordering $\sqsubseteq$?
  ▪ $\sqsubseteq = \subseteq$

Meet operator $\sqcap$?
  ▪ $\sqcap = \cap$
Very busy expressions

Forward or backward?
- backward

May or must?
- must

Lattice set L?
- \( L = \) set of expressions

Ordering \( \sqsubseteq \)?
- \( \sqsubseteq = \subseteq \)

Meet operator \( \sqcap \)?
- \( \sqcap = \cap \)

Top element \( \top \)?
Very busy expressions

Forward or backward?
- backward

May or must?
- must

Lattice set L?
- \( L = \text{set of expressions} \)

Ordering \( \sqsubseteq \)?
- \( \sqsubseteq = \subseteq \)

Meet operator \( \sqcap \)?
- \( \sqcap = \cap \)

Top element \( \top \)?
- \( \top = \text{set of all expressions} \)
Very busy expressions

\[ VBOut[B] = \cap \{ \text{VBIn}[B'] \mid B' \in \text{succ}(B) \} \]

\[ \text{VBIn}[I] = (VBOut[I] \cup \text{Gen}[I]) - \text{Kill}[I] \]

\( \text{Kill}[I] \) = set of expressions killed in \( I \)

\( \text{Gen}[I] \) = set of expressions generated at \( I \)

\[ \text{Gen}[x = y + z] = \{ y + z \} \]

\[ \text{Gen}[x = y] = \{ \} \]

\[ \text{Kill}[x = \ldots] = \{ e \mid e \text{ contains } x \} \]