CS412/413
Introduction to Compilers
Radu Rugina
Lecture 24: Dataflow Analysis Frameworks
29 Apr 06

Live Variable Analysis
What are the live variables at each program point?
Method:
1. Define sets of live variables
2. Solve constraints

Derive Constraints
Constraints for each instruction:
\begin{align*}
\text{in}[] &= (\text{out}[]) \cup \text{def}[] \\
\text{Constraints for control flow:} &\\text{out}[B] = \bigcup_{B \in \text{out}(B)} \text{in}[B]
\end{align*}

Initialization
\begin{align*}
L_0 &= L_0 \cup \{c\} \\
L_1 &= L_0 \cup L_0 \\
L_2 &= (L_0 - \{a\}) \cup \{y\} \\
L_3 &= (L_0 - \{a\}) \cup \{x\} \\
L_4 &= L_0 \cup \{d\} \\
L_5 &= L_0 \cup \{a\} \\
L_6 &= L_0 - \{a\} \cup \{y,x\} \\
L_7 &= L_0 - \{a\} \cup \{x\} \\
L_8 &= L_0 - \{a\} \\
L_9 &= L_0 - \{a\} \cup \{x\} \\
L_{10} &= L_0 - \{a\} \\
L_{11} &= L_0 - \{a\} \\
L_{12} &= (L_0 - \{a\}) \cup \{x\}
\end{align*}

Iteration 1
\begin{align*}
L_0 &= L_0 \cup \{c\} \\
L_1 &= L_0 \cup L_0 \\
L_2 &= (L_0 - \{a\}) \cup \{y\} \\
L_3 &= (L_0 - \{a\}) \cup \{x\} \\
L_4 &= L_0 \cup \{d\} \\
L_5 &= L_0 \cup \{a\} \\
L_6 &= L_0 - \{a\} \cup \{y,x\} \\
L_7 &= L_0 - \{a\} \cup \{x\} \\
L_8 &= L_0 - \{a\} \\
L_9 &= L_0 - \{a\} \cup \{y,x\} \\
L_{10} &= L_0 - \{a\} \\
L_{11} &= L_0 - \{a\} \\
L_{12} &= (L_0 - \{a\}) \cup \{x\}
\end{align*}
### Iteration 2

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

**if (c)**

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

**if (d)**

\[ z = y+1 \]

\[ L_1 = (x, y, c, a) \]
\[ L_2 = (x, y, c, a) \]
\[ L_3 = (y, c, a) \]
\[ L_4 = (x, y, c, a) \]
\[ L_5 = (x, y, c, a) \]
\[ L_6 = (y, c, a) \]
\[ L_7 = (x, y, c, a) \]
\[ L_8 = (x, y, c, a) \]
\[ L_9 = (x, y, c, a) \]
\[ L_{10} = (x, y, c, a) \]

\[ L_1 = (x, y, c, a) \]
\[ L_2 = (x, y, c, a) \]
\[ L_3 = (y, c, a) \]
\[ L_4 = (x, y, c, a) \]
\[ L_5 = (x, y, c, a) \]
\[ L_6 = (y, c, a) \]
\[ L_7 = (x, y, c, a) \]
\[ L_8 = (x, y, c, a) \]
\[ L_9 = (x, y, c, a) \]
\[ L_{10} = (x, y, c, a) \]

### Fixed-point!

**if (c)**

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

**if (d)**

\[ z = y+1 \]

### Final Result

**if (c)**

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

**if (d)**

\[ z = y+1 \]

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

### Characterize All Executions

**if (c)**

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

**if (d)**

\[ z = x \]

### Generalization

- Live variable analysis and available copies analysis 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
- Methodology:
  - Describe information about the program using an algebraic structure called lattice
  - Build constraints which show how computation 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 \(x \sqsubseteq y, y \sqsubseteq x \implies x = y\)
    3. Transitive \(x \sqsubseteq y, y \sqsubseteq z \implies x \sqsubseteq z\)
- 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 a 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?

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
- Alternative notation for a lattice: \((L, \sqsubseteq, \sqcap, \sqcup)\)

More About Lattices

- In a lattice \((L, \sqsubseteq)\), the following are equivalent:
  1. \(x \sqsubseteq y\)
  2. \(x \sqcup y = y\)
  3. \(x \sqcap y = x\)
- Note: meet and join operations were defined using the partial order relation
Proof

- Prove that $x \subseteq y$ implies $x \cap y = x$:
  - $y$ is a lower bound of $(x,y)$ because:
    - $y$ is less than $y$ by reflexivity
    - $x$ is less than $y$ by hypothesis
  - Take another lower bound $z$ of $(x,y)$:
    - Then $z$ is less than $x$, $y$
    - In particular, $z$ is less than $x$
    - So $x$ is the least upper bound
- Prove that $x \cap y = x$ implies $x \subseteq y$:
  - By hypothesis, $x$ is a lower bound of $(x,y)$
  - So $x$ is less than $y$

Properties of Meet and Join

- The meet and join operators are:
  1. Associative $(x \cap y) \cap z = x \cap (y \cap z)$
  2. Commutative $x \cap y = y \cap x$
  3. Idempotent: $x \cap x = x$
- Property: If "$\subseteq$" is an associative, commutative, and idempotent operator, then the relation "$\sqsubseteq$" defined as $x \sqsubseteq y$ iff $x \cap y = y$ is a partial order

Example Lattice

- Consider $S = \{a, b, c\}$ and its power set $P = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, b, c\}\}$
- Define partial order as set inclusion: $X \sqsubseteq Y$
  - Reflexive $X \sqsubseteq Y$
  - Anti-symmetric $X \sqsubseteq Y, Y \sqsubseteq X \Rightarrow X = Y$
  - Transitive $X \sqsubseteq Y, Y \sqsubseteq Z \Rightarrow X \sqsubseteq Z$
- Also, for any subset $L \subseteq P$, there exists LUB$(L)$ and GLB$(L)$
- Therefore $(P, \sqsubseteq)$ is a (complete) lattice

Hasse Diagrams

- Hasse diagram = graphical representation of a lattice where $x$ is below $y$ when $x \sqsubseteq y$ 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)
Lattices in Dataflow Analysis

- Information computed by live variable analysis and available copies can be expressed as elements of lattices
- Live variables:
  - \( V \) is the set of all variables in the program
  - \( P \) the power set of \( V \)
  - Lattice: \( (2^V, \supseteq, \cup, \emptyset) \)
  - sets of live variables are elements of this lattice
  - Information propagates backward

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 the lattice information changes:
  - At each CFG node, due to the computation in that node
  - At join/split points in the control flow

Transfer Functions

- Dataflow analysis defines a transfer function \( F_n : L \rightarrow L \) for each CFG node in the program
- Let \( \text{in}[n] \) be the information before CFG node \( n \), and \( \text{out}[n] \) be the information after \( n \)
- Forward analysis:
  \[ \text{out}[n] = F_n(\text{in}[n]) \]
- Backward analysis:
  \[ \text{in}[n] = F_n^{-1}(\text{out}[n]) \]
- Transfer functions must be monotonic:
  - For all \( A, B \) in \( L : A \subseteq B \) implies \( F_n(A) \subseteq F_n(B) \)

Merge Operation

- Dataflow analysis uses the meet operation to merge dataflow information at split/join points in the control flow
- Forward analysis:
  \[ \text{in}[n] = \bigcap \{\text{out}[n] \mid n\in\text{pred}(n)\} \]
- Backward analysis:
  \[ \text{out}[n] = \bigcap \{\text{in}[n] \mid n\in\text{succ}(n)\} \]

Dataflow Analysis Framework

- A dataflow analysis framework consists of:
  - A lattice \( (L, \subseteq, \cap, \top) \) where \( L \) is the dataflow information, \( \subseteq \) is the ordering, \( \cap \) is the meet operation, and \( \top \) is the top element
    - Lattice must have finite height
  - Transfer functions \( F_n : L \rightarrow L \) for each CFG node \( n \)
    - Transfer functions must be monotonic
  - Boundary dataflow information \( d_0 \)
    - Before CFG entry node for a forward analysis
    - After CFG exit node for a backward analysis