CS412/413
Introduction to Compilers
Radu Rugina

Lecture 25: Dataflow Analysis Frameworks
02 Apr 04

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:
in[I] = (out[I] - def[I])
\cup use[I]

Constraints for control flow:
out[B] = \bigcup_{I : F \rightarrow B} in[B]

Initialization
if (c)
L_1 = L_2 \cup \{x\}
L_2 = L_2 \cup L_3
L_3 = L_3 \cup \{y\}
x = y + 1
y = 2z
if (d)
L_1 = L_1 \cup \{z\}
L_2 = L_2 \cup \{x\}
L_3 = L_3 \cup \{y\}
L_4 = L_4 \cup \{z\}
L_5 = L_5 \cup \{x\}
L_6 = L_6 \cup \{y\}
L_7 = L_7 \cup \{z\}
L_8 = L_8 \cup \{x\}
L_9 = L_9 \cup \{y\}
L_{10} = L_{10} \cup \{z\}
L_{11} = L_{11} \cup \{x\}
L_{12} = L_{12} \cup \{y\}

Iteration 1
if (c)
L_1 = L_1 \cup \{x,y,z,c,d\}
L_2 = L_2 \cup \{x\}
L_3 = L_3 \cup \{y\}
L_4 = L_4 \cup \{z\}
L_5 = L_5 \cup \{x\}
L_6 = L_6 \cup \{y\}
L_7 = L_7 \cup \{z\}
L_8 = L_8 \cup \{x\}
L_9 = L_9 \cup \{y\}
L_{10} = L_{10} \cup \{z\}
L_{11} = L_{11} \cup \{x\}
L_{12} = L_{12} \cup \{y\}

**Iteration 2**

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

**Fixed-point!**

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

**Final Result**

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

**Characterize All Executions**

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

**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:
  1. A set \(P\)
  2. 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 \((L, \sqsubseteq)\) is a lattice if:
  1. \((L, \sqsubseteq)\) 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: \(x \sqcap y = GLB\{x,y\}\)
  2. Join operator: \(x \sqcup y = LUB\{x,y\}\)
- 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: \(\bot = GLB(L)\)
  2. Top element: \(\top = LUB(L)\)
- 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: \(X \sqsubseteq Y\)
  - Reflexive \(X \subseteq Y\)
  - 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 two elements of \(P\), there is a set which includes both and another set which is included in both
- Therefore \((P, \sqsubseteq)\) 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

\[
\begin{array}{c}
\{a,b,c\} \\
\{a,b\} \quad \{a,c\} \quad \{b,c\} \\
\{a\} \quad \{b\} \quad \{c\} \\
\emptyset
\end{array}
\]

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)

\[
\begin{array}{c}
\{a,b,c\} \\
\{a,b\} \quad \{a,c\} \quad \{b,c\} \\
\{a\} \quad \{b\} \quad \{c\} \\
\emptyset
\end{array}
\]

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 \times V the cartesian product representing all possible copy instructions
  - P the power set of V \times V

- Then:
  - (P, \subseteq) is a lattice
  - sets of available copies are lattice elements

More About Lattices

- In a lattice (L, \subseteq), the following are equivalent:
  1. x \subseteq y
  2. x \cap y = x
  3. x \cup y = y

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

- Prove that $x \sqsubseteq y$ 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 $x \sqsubseteq y$:
  - $x$ is a lower bound of $\{x, y\}$
  - $x$ is less than $x$ and $y$
  - In particular, $x$ is less than $y$

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

Proof

- Prove that $x \sqsubseteq y$ implies $x \sqcup y = y$:
  - $y$ is an upper bound of $\{x, y\}$
  - All upper bounds of $\{x, y\}$ greater than $x, y$
  - In particular, they are greater than $x$

- Prove that $x \sqcup y = y$ implies $x \sqsubseteq y$:
  - $y$ is a upper bound of $\{x, y\}$
  - $y$ is greater than $x$ and $y$
  - In particular, $y$ is greater than $x$

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 $(I_1, ..., I_n)$ with transfer functions $F_1, ..., F_n$
  - $\text{in}[B]$ 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 \( \text{in}[B] \) is lattice information at beginning of block \( B \) and \( \text{out}[B] \) is lattice information at end of \( B \).

- **Forward analysis:** \( \text{in}[B] = \bigwedge \{ \text{out}[B'] | B' \in \text{pred}(B) \} \)

- **Backward analysis:** \( \text{out}[B] = \bigwedge \{ \text{in}[B'] | B' \in \text{succ}(B) \} \)

- Can alternatively use join operation \( \sqcup \) (equivalent to using the meet operation \( \sqcap \) in the reversed lattice).