Dataflow Analysis Frameworks

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

Constraints for each instruction:

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

Constraints for control flow:

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

Derive Constraints

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

Derive Constraints

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

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

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

\[ L_1 = \{ x, y, z, c, d \} \]
\[ L_2 = L_3 \cup L_3 \]
\[ L_3 = \{ x, y, z, c, d \} \]
\[ L_4 = \{ x, y, z, c, d \} \]
\[ L_5 = \{ x, y, z, c, d \} \]
\[ L_6 = \{ x, y, z, c, d \} \]
\[ L_7 = \{ x, y, z, c, d \} \]
\[ L_8 = \{ x, y, z, c, d \} \]

Iteration 1

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

\[ L_1 = \{ x, y, z, c, d \} \]
\[ L_2 = \{ x, y, z, c, d \} \]
\[ L_3 = \{ x, y, z, c, d \} \]
\[ L_4 = \{ x, y, z, c, d \} \]
\[ L_5 = \{ x, y, z, c, d \} \]
\[ L_6 = \{ x, y, z, c, d \} \]

Iteration 2

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

\[ L_1 = \{ x, y, z, c, d \} \]
\[ L_2 = \{ x, y, z, c, d \} \]
\[ L_3 = \{ x, y, z, c, d \} \]
\[ L_4 = \{ x, y, z, c, d \} \]
\[ L_5 = \{ x, y, z, c, d \} \]
\[ L_6 = \{ x, y, z, c, d \} \]

Fixed-point!

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

\[ L_1 = \{ x, y, z, c, d \} \]
\[ L_2 = \{ x, y, z, c, d \} \]
\[ L_3 = \{ x, y, z, c, d \} \]
\[ L_4 = \{ x, y, z, c, d \} \]
\[ L_5 = \{ x, y, z, c, d \} \]
\[ L_6 = \{ x, y, z, c, d \} \]
Final Result

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

Final result: sets of live variables at each program point

Characterize All Executions

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

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

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 a lattice
  - Build constraints that show how instructions and control flow influence the information in terms of values in the lattice
  - Iteratively solve constraints

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
Partial Order Relations

• Lattice definition builds on the concept of a partial order relation

• A partial order \((P, \sqsubseteq)\) consists of:
  - A set \(P\)
  - A partial order relation \(\sqsubseteq\) that 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 a “partial order” because not all elements are comparable, in contrast with a total order, in which:
  -4. Total \(x \sqsubseteq y \text{ or } y \sqsubseteq x\)

Example

• \(P\) is \((\text{red, blue, yellow, purple, orange, green})\)

• \(\sqsubseteq\)
  - red \(\sqsubseteq\) purple, red \(\sqsubseteq\) orange
  - blue \(\sqsubseteq\) purple, blue \(\sqsubseteq\) green
  - yellow \(\sqsubseteq\) orange, blue \(\sqsubseteq\) green
  - red \(\sqsubseteq\) red, blue \(\sqsubseteq\) blue
  - yellow \(\sqsubseteq\) yellow
  - purple \(\sqsubseteq\) purple
  - orange \(\sqsubseteq\) orange
  - green \(\sqsubseteq\) green

Hasse Diagrams

• A graphical representation of a partial order, where
  - \(x\) and \(y\) are on the same level when they are incomparable
  - \(x\) is below \(y\) when \(x \sqsubseteq y\) and \(x \neq y\)
  - \(x\) is below \(y\) and connected by a line when \(x \sqsubseteq y, x \neq y,\) and there is no \(z\) such that \(x \sqsubseteq z, z \sqsubseteq y, x \neq z,\) and \(y \neq z\)

Lower/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 \(S\)
Example, cont.

- red is lower bound for {purple, orange}
- blue is lower bound for {purple, green}
- yellow is lower bound for {orange, green}
- no lower bound for {red, blue}
- no lower bound for {red, yellow}
- no lower bound for {blue, yellow}, etc.

- purple is upper bound for {red, blue}
- orange is upper bound for {red, yellow}
- green is upper bound for {purple, orange}
- no upper bound for {red, blue, yellow}
- no upper bound for {purple, orange}
- no upper bound for {orange, green}
- etc.

Example, cont.

- red' is also a lower bound for {purple, orange}

LUB and GLB

- Define least upper bound (LUB) and greatest lower bound (GLB) as follows:
- 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?
Example

- red is GLB for (purple, orange)
- green is GLB for (orange, green)
- purple is LUB for (red, blue)
- orange is LUB for (red, yellow)
- green is LUB for (orange, green)
- purple is LUB for (red', blue)
- orange is LUB for (red', yellow)

Lattices

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

Example

- $L$ is natural numbers \{0, 1, 2, 3, ...\}
- $\sqsubseteq$ is $\leq$

Every finite subset of $L$ has a LUB
Every subset of $L$ has a GLB
Therefore $(L, \sqsubseteq)$ is a lattice

Complete Lattices

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

- Can identify and name two special elements:
  1. Bottom element: \(\bot = \text{GLB}(L)\)
  2. Top element: \(\top = \text{LUB}(L)\)
- All finite lattices are complete
Example

- L is natural numbers \{0, 1, 2, 3, \ldots\}
- \(\subseteq\) is \(\leq\)

Every finite subset of L has a GLB and LUB
Therefore (L, \(\subseteq\)) is a lattice
Every infinite subset of L has a LUB
Therefore (L, \(\subseteq\)) is a complete lattice
However, L has infinite ascending chains

Meet and Join

- By definition, for any lattice L, GLBs and LUBs are defined for finite sets
- Define operators meet (\(\sqcap\)) and join (\(\sqcup\)) as
  - \(x \sqcap y = \text{GLB}(\{x,y\})\)
  - \(x \sqcup y = \text{LUB}(\{x,y\})\)
- For any finite set \(S \subseteq L\)
  - \(\sqcap S = \text{GLB}(S)\)
  - \(\sqcup S = \text{LUB}(S)\)

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 \(X \subseteq X\)
  - 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 that includes both and another set that is included in both
- Therefore (\(P, \subseteq\)) is a (complete) lattice
Power Set Lattice
• Partial order: ⊆ (set inclusion)
• Meet: ∩ (set intersection)
• Join: ∪ (set union)
• Top element: \{a,b,c\} (whole set)
• Bottom element: ∅ (empty set)

Reversed Lattice
• Partial order: ⊇ (set inclusion)
• Meet: ∪ (set union)
• Join: ∩ (set intersection)
• Top element: ∅ (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 Dataflow Analysis
• 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
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
  - Determine how 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

- 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:
    \[ \text{out}[B] = F_n(\ldots(F_1(\text{in}[B]))) = F_n \circ \ldots \circ F_1(\text{in}[B]) \]

  - Backward analysis:
    \[ \text{in}[I] = F_1(\ldots(F_n(\text{out}[I]))) = F_1 \circ \ldots \circ F_n(\text{out}[B]) \]

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

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

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

- Let \( L_1, \ldots, L_n \) be sets

- Cartesian product of \( L_1, \ldots, L_n \) is
  \[ \{ <x_1, \ldots, x_n> \mid x_i \in L_i \} \]

- If \( L_1, \ldots, L_n \) are (complete) lattices then their Cartesian product is a (complete) lattice, where \( \sqsubseteq \) is defined by
  \[ <x_1, \ldots, x_n> \sqsubseteq <y_1, \ldots, y_n> \text{ iff for all } i, x_i \sqsubseteq y_i \]

Information as Cartesian Product

- Consider a program analysis in which \( n \) program analysis variables range over lattice \( L \)
- We view the analysis as computing an \( n \)-tuple of \( L \)-values, i.e., a point in the \( n \)-ary Cartesian product of \( L \)
- Each change of one program analysis variable changes one component of the \( n \)-tuple
- Analyses will terminate because we will only consider
  - Lattices with no infinite descending chains
  - "Monotonic" transfer functions that move us down (or not at all) in the lattice

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 (1 & 2)

- Prove that \( x \sqsubseteq y \) implies \( x \sqcap y = x \):
  - \( x \) 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 \( x \sqcap y = x \) 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.