CS412/413

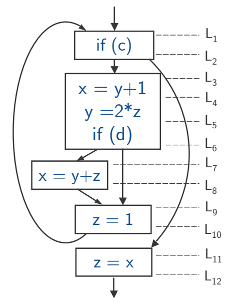
Introduction to Compilers
Radu Rugina

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
1. Build constraints
2. Solve constraints



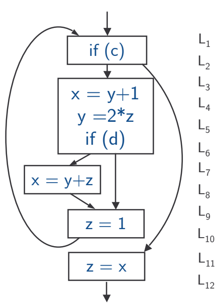
Derive Constraints

Constraints for each instruction:

in[I]=(out[I]-def[I]) ∪ use[I]

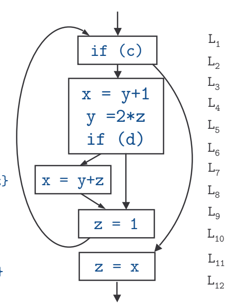
Constraints for control flow:

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



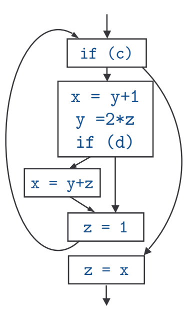
Derive Constraints

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



Initialization

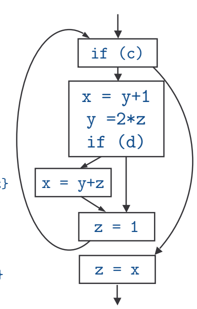
$L_1 = L_2 \cup \{c\}$
$L_2 = L_3 \cup L_{11}$
$L_3 = (L_4 - \{x\}) \cup \{y\}$
$L_4 = (L_5 - \{y\}) \cup \{z\}$
$L_5 = L_6 \cup \{d\}$
$L_6 = L_7 \cup L_9$
$L_7 = (L_8 - \{x\}) \cup \{y,z\}$
$L_8 = L_9$
$L_9 = L_{10} - \{z\}$
$L_{10} = L_1$
$L_{11} = (L_{12} - \{z\}) \cup \{x\}$



$L_1 = \{\}$
$L_2 = \{\}$
$L_3 = \{\}$
$L_4 = \{\}$
$L_5 = \{\}$
$L_6 = \{\}$
$L_7 = \{\}$
$L_8 = \{\}$
$L_9 = \{\}$
$L_{10} = \{\}$
$L_{11} = \{\}$
$L_{12} = \{\}$

Iteration 1

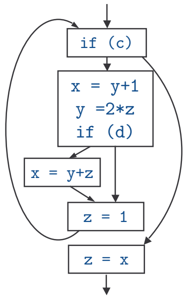
$L_1 = L_2 \cup \{c\}$
$L_2 = L_3 \cup L_{11}$
$L_3 = (L_4 - \{x\}) \cup \{y\}$
$L_4 = (L_5 - \{y\}) \cup \{z\}$
$L_5 = L_6 \cup \{d\}$
$L_6 = L_7 \cup L_9$
$L_7 = (L_8 - \{x\}) \cup \{y,z\}$
$L_8 = L_9$
$L_9 = L_{10} - \{z\}$
$L_{10} = L_1$
$L_{11} = (L_{12} - \{z\}) \cup \{x\}$



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

Iteration 2

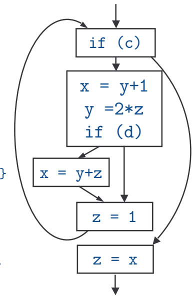
$L_1 = L_2 \cup \{c\}$
$L_2 = L_3 \cup L_{11}$
$L_3 = (L_4-\{x\}) \cup \{y\}$
$L_4 = (L_5-\{y\}) \cup \{z\}$
$L_5 = L_6 \cup \{d\}$
$L_6 = L_7 \cup L_9$
$L_7 = (L_8-\{x\}) \cup \{y,z\}$
$L_8 = L_9$
$L_9 = L_{10}-\{z\}$
$L_{10} = L_1$
$L_{11} = (L_{12}-\{z\}) \cup \{x\}$



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

Fixed-point!

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

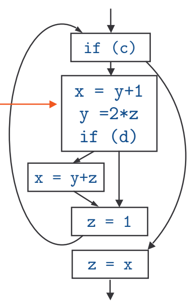


$L_1=\{x,y,z,c,d\}$
$L_2=\{x,y,z,c,d\}$
$L_3=\{y,z,c,d\}$
$L_4=\{x,z,c,d\}$
$L_5=\{x,y,z,c,d\}$
$L_6=\{x,y,z,c,d\}$
$L_7=\{y,z,c,d\}$
$L_8=\{x,y,c,d\}$
$L_9=\{x,y,c,d\}$
$L_{10}=\{x,y,z,c,d\}$
$L_{11}=\{x\}$
$L_{12}=\{\}$

Final Result

x live here !

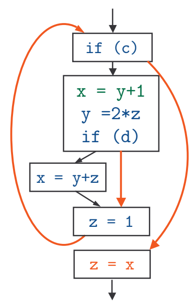
Final result: sets of live variables at each program point



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

Characterize All Executions

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



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

Generalization

- Live variable analysis and 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 $x \sqsubseteq y,\ y \sqsubseteq x \Rightarrow x = y$
    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 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?

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: $x \sqcap y = \mathrm{GLB}(\{x,y\})$
  2. Join operator: $x \sqcup y = \mathrm{LUB}(\{x,y\})$
- 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: $\bot = \mathrm{GLB}(L)$
  2. Top element: $\top = \mathrm{LUB}(L)$
- All finite lattices are complete
- Alternative notation for a lattice: $(L, \sqsubseteq, \sqcap, \top)$

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 $x \sqsubseteq y$ implies $x \sqcap y = x$:
  - 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 $x \sqcap y = x$ implies $x \sqsubseteq y$:
  - By hypothesis, x is a lower bound of {x,y}
  - So x is less than y

Properties of Meet and Join

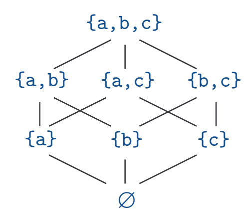
- The meet and join operators are:
  1. Associative $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
  2. Commutative $x \sqcap y = y \sqcap x$
  3. Idempotent: $x \sqcap x = x$
- Property: If "$\sqcap$" is an associative, commutative, and idempotent operator, then the relation "$\sqsubseteq$" defined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

Example Lattice

- Consider S = {a,b,c} and its power set P = {∅, {a}, {b}, {c}, {a,b}, {b,c}, {a,c} {a,b,c}}
- Define partial order as set inclusion: X⊆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 subset $L \subseteq P$, there exists LUB(L) and GLB(L)
- Therefore (P,⊆) is a (complete) lattice

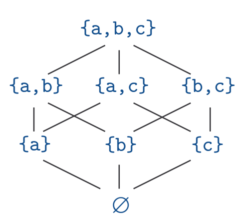
Hasse Diagrams

- Hasse diagram = graphical representation of a lattice where x is below y when $x \sqsubseteq y$ and $x \neq y$



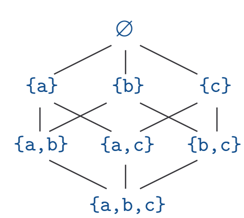
Power Set Lattice

- Partial order: ⊆ (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)



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, ⊇, ∪, ∅)
  - sets of live variables are elements of this lattice
  - Information propagates backward

Lattices in Dataflow Analysis

- 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
  - ($2^{V \times V}$, ⊆, ∩, V x V)
  - sets of available copies are lattice elements
  - information propagates forward

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 $F_n : L \to L$ for each CFG node in the program
- Let `in[n]` be the information before CFG node `n`, and `out[n]` be the information after `n`
- Forward analysis: `out[n] = F`$_n$ `(in[n])`
- Backward analysis: `in[n] = F`$_n$ `(out[n])`
- Transfer functions must be monotonic:
  - For all A, B in L : A ⊑ B implies $F_n(A) \sqsubseteq F_n(B)$

Merge Operation

- Dataflow analysis uses the meet operation to merge dataflow information at split/join points in the control flow
- Forward analysis: `in[n] = ⊓ {out[n'] | n'∈pred(n)}`
- Backward analysis: `out[n] = ⊓ {in[n'] | n'∈succ(n)}`

Dataflow Analysis Framework

- A dataflow analysis framework consists of:
  - A lattice (L, ⊑, ⊓, ⊤) where L is the dataflow information, ⊑ is the ordering, ⊓ is the meet operation, and ⊤ is the top element
    - Lattice must have finite height
  - Transfer functions $F_n : L \to L$ 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