Introduction to Machine-Independent Optimizations - 3

Data-Flow Analysis

Y.N. Srikant

Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Principles of Compiler Design

Outline of the Lecture

- What is code optimization? (in part 1)
- Illustrations of code optimizations (in part 1)
- Examples of data-flow analysis
- Fundamentals of control-flow analysis
- Algorithms for two machine-independent optimizations
- SSA form and optimizations

Available Expression Computation

- Sets of expressions constitute the domain of data-flow values
- Forward flow problem
- Confluence operator is $\cap$
- An expression $x + y$ is *available* at a point p, if every path (not necessarily cycle-free) from the initial node to p evaluates $x + y$, and after the last such evaluation, prior to reaching p, there are no subsequent assignments to x or y
- A block *kills* $x + y$, if it assigns (or may assign) to x or y and does not subsequently recompute $x + y$.
- A block *generates* $x + y$, if it definitely evaluates $x + y$, and does not subsequently redefine x or y

Available Expression Computation - EGEN and EKILL

In other blocks:

d5: b = a+4
d6: f = e+c
d7: e = b+d
d8: d = a+b
d9: a = c+f
d10: c = e+a

B:

d1: a = f + 1
d2: b = a + 7
d3: c = b + d
d4: a = d + c

Set of all expressions = {f+1,a+7,b+d,d+c,a+4,e+c,a+b,c+f,e+a}

EGEN[B] = {f+1,b+d,d+c}
EKILL[B] = {a+4,a+b,e+a,e+c,c+f,a+7}

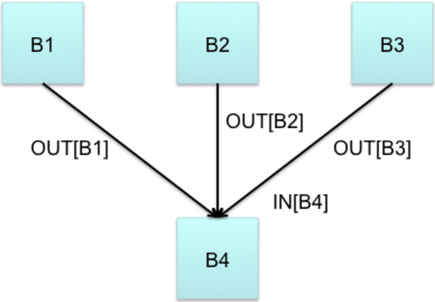
Available Expression Computation - DF Equations (1)

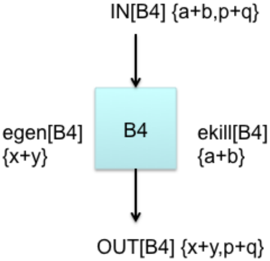
- The data-flow equations

$$\begin{aligned} IN[B] &= \bigcap_{P\ is\ a\ predecessor\ of\ B} OUT[P],\ B\ not\ initial \\ OUT[B] &= e\_gen[B] \bigcup (IN[B] - e\_kill[B]) \\ IN[B1] &= \phi \\ IN[B] &= U,\ for\ all\ B \neq B1\ (initialization\ only) \end{aligned}$$

- $B1$ is the intial or entry block and is special because nothing is available when the program begins execution
- $IN[B1]$ is always ϕ
- U is the universal set of all expressions
- Initializing $IN[B]$ to ϕ for all $B \neq B1$, is restrictive

Available Expression Computation - DF Equations (2)





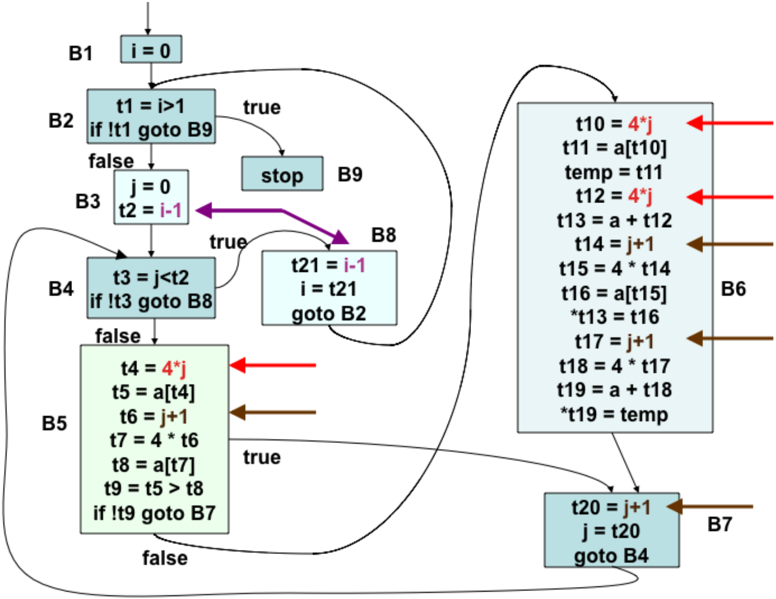
IN[B4] = OUT[B1] $\bigcap$ OUT[B2] $\bigcap$ OUT[B3]

OUT[B4] = egen[B4] $\bigcup$ (IN[B4] – ekill[B4])

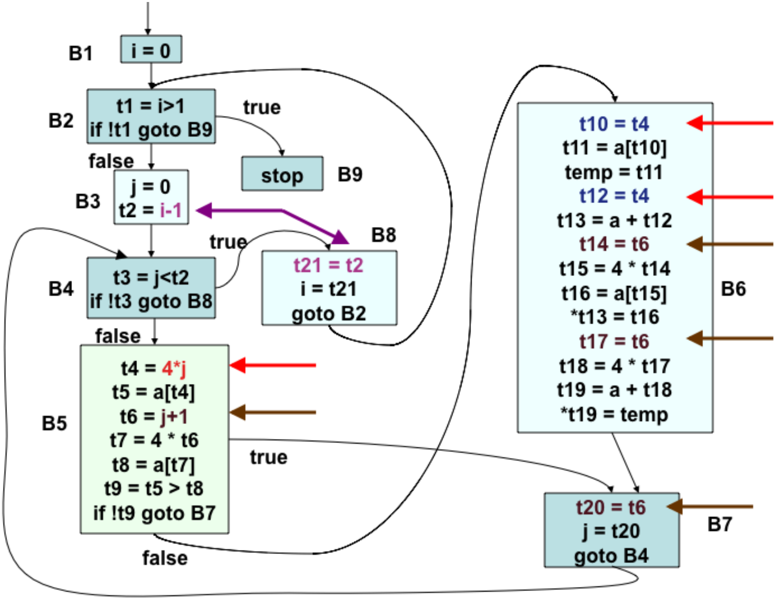
$$IN[B] = \bigcap_{P \text{ is a predecessor of } B} OUT[P], \ B \text{ not initial}$$

$$OUT[B] = e\_gen[B] \bigcup (IN[B] - e\_kill[B])$$

Available Expression Computation - An Example



Available Expression Computation - An Example (2)



An Iterative Algorithm for Computing Available Expressions

for each block $B \neq B1$ do $\{OUT[B] = U - e\_kill[B];\}$
/* You could also do $IN[B] = U$;*/
/* In such a case, you must also interchange the order of */
/* $IN[B]$ and $OUT[B]$ equations below */
$change = true$;
while $change$ do { $change = false$;
  for each block $B \neq B1$ do {

$$\begin{aligned} IN[B] &= \bigcap_{P\ a\ predecessor\ of\ B} OUT[P]; \\ oldout &= OUT[B]; \\ OUT[B] &= e\_gen[B] \bigcup (IN[B] - e\_kill[B]); \end{aligned}$$

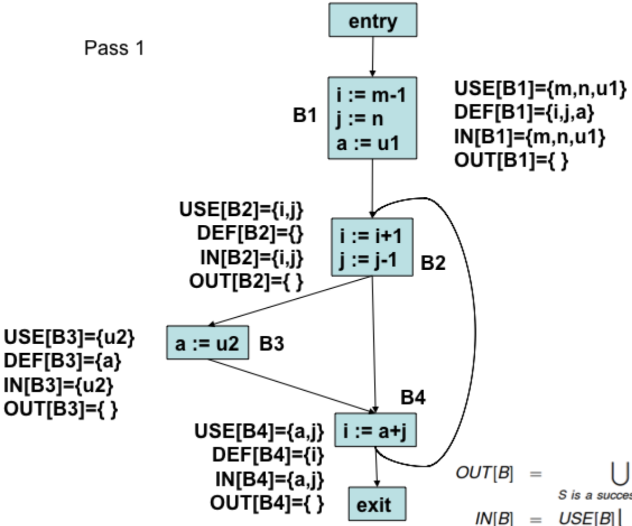
    if ($OUT[B] \neq oldout$) $change = true$;
  }
}

Live Variable Analysis

- The variable x is *live* at the point p, if the value of x at p could be used along some path in the flow graph, starting at p; otherwise, x is *dead* at p
- Sets of variables constitute the domain of data-flow values
- Backward flow problem, with confluence operator $\bigcup$
- $IN[B]$ is the set of variables live at the beginning of B
- $OUT[B]$ is the set of variables live just after B
- $DEF[B]$ is the set of variables definitely assigned values in B, prior to any use of that variable in B
- $USE[B]$ is the set of variables whose values may be used in B prior to any definition of the variable

$$
\begin{aligned}
OUT[B] &= \bigcup_{S \text{ is a successor of } B} IN[S] \\
IN[B] &= USE[B] \bigcup (OUT[B] - DEF[B]) \\
IN[B] &= \phi, \textit{for all } B \ (\textit{initialization only})
\end{aligned}
$$

Live Variable Analysis: An Example - Pass 1

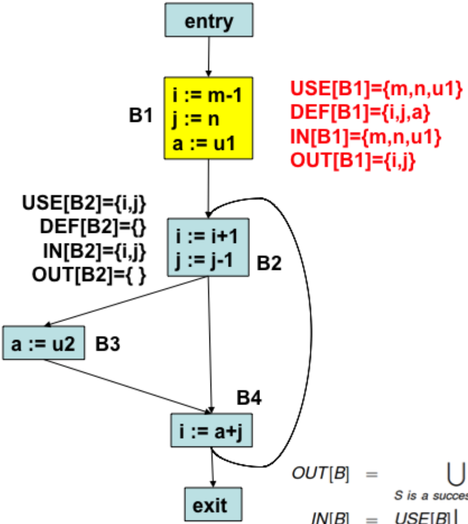


$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

Live Variable Analysis: An Example - Pass 2.1

Pass 2

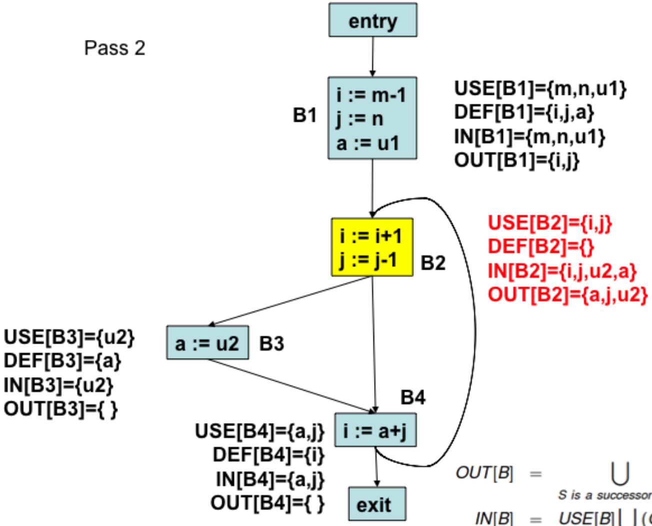


$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

Live Variable Analysis: An Example - Pass 2.2

Pass 2

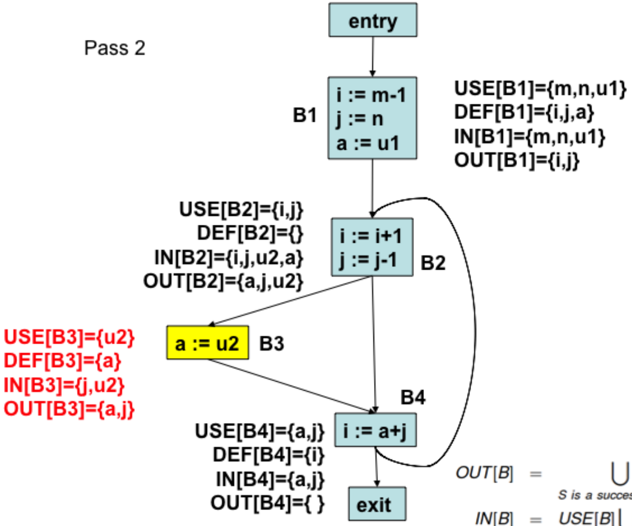


$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

Live Variable Analysis: An Example - Pass 2.3

Pass 2

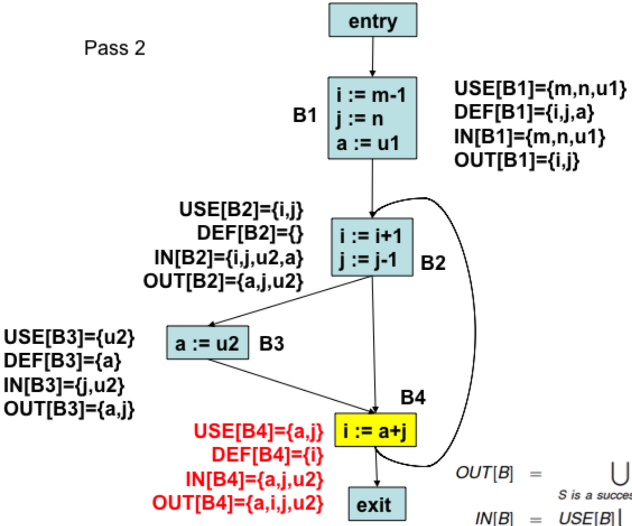


$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

Live Variable Analysis: An Example - Pass 2.4

Pass 2

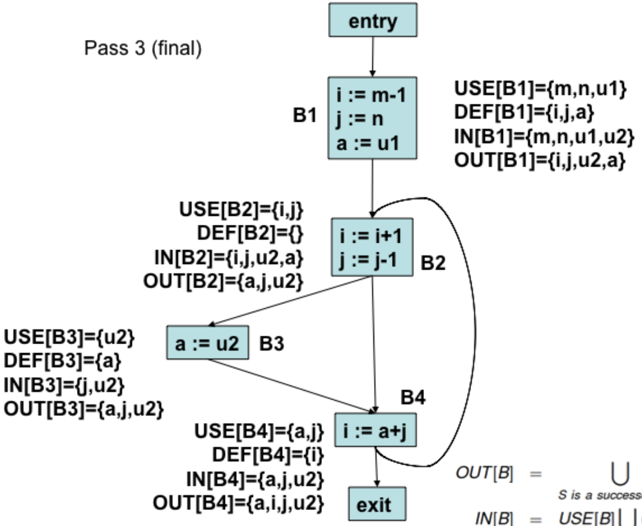


$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

Live Variable Analysis: An Example - Final pass

Pass 3 (final)



$$OUT[B] = \bigcup_{S \text{ is a successor of } B} IN[S]$$

$$IN[B] = USE[B] \bigcup (OUT[B] - DEF[B])$$

Data-flow Analysis: Theoretical Foundations

Y.N. Srikant

Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Principles of Compiler Design

Foundations of Data-flow Analysis

- Basic questions to be answered
  1. In which situations is the iterative DFA algorithm correct?
  2. How precise is the solution produced by it?
  3. Will the algorithm converge?
  4. What is the meaning of a "solution"?
- The above questions can be answered accurately by a DFA framework
- Further, reusable components of the DFA algorithm can be identified once a framework is defined
- A DFA framework $(D, V, \wedge, F)$ consists of

D	:	A direction of the dataflow, either forward or backward
V	:	A domain of values
$\wedge$	:	A meet operator; $(V, \wedge)$ form a semi-lattice
F	:	A family of transfer functions, $V \longrightarrow V$ F includes constant transfer functions for the ENTRY/EXIT nodes as well

Semi-Lattice

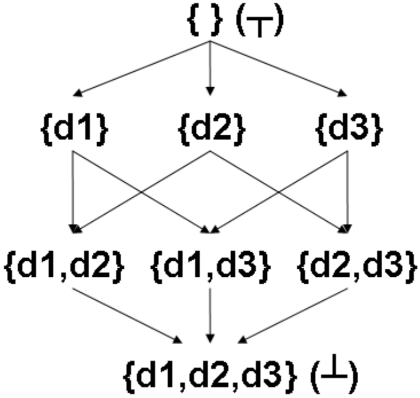
- A semi-lattice is a set V and a binary operator $\wedge$, such that the following properties hold
  1. V is closed under $\wedge$
  2. $\wedge$ is idempotent ($x \wedge x = x$), commutative ($x \wedge y = y \wedge x$), and associative ($x \wedge (y \wedge z) = (x \wedge y) \wedge z$)
  3. It has a *top* element, $\top$, such that $\forall\ x \in V,\ \top \wedge x = x$
  4. It may have a *bottom* element, $\perp$, such that $\forall x \in\ V,\ \perp \wedge\, x = \perp$
- The operator $\wedge$ defines a partial order $\leq$ on V, such that $x \leq y$ *iff* $x \wedge y = x$

Semi-Lattice of Reaching Definitions

- 3 definitions, {d1,d2,d3}
- V is the set of all subsets of {d1,d2,d3}
- $\wedge$ is $\cup$
- The diagram (next slide) shows the partial order relation induced by $\wedge$ (i.e., $\cup$)
- Partial order relation is $\supseteq$
- An arrow, $y \rightarrow x$ indicates $x \supseteq y$ ($x \leq y$)
- Each set in the diagram is a data-flow value
- Transitivity is implied in the diagram ($a \rightarrow b$ & $b \rightarrow c$ imples $a \rightarrow c$)
- An ascending chain: $(x_1 < x_2 < ... < x_n)$
- Height of a semi-lattice: largest number of '<' relations in any ascending chain
- Semi-lattices in our DF frameworks will be of finite height

Lattice Diagram of Reaching Definitions

$y \rightarrow x$ indicates $x \supseteq y$ ($x \leq y$)



Transfer Functions

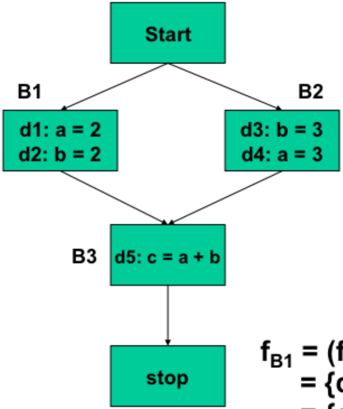
$F: V \rightarrow V$ has the following properties

1. F has an identity function, $I(x) = x$, for all $x \in V$
2. F is closed under composition, *i.e.,* for $f, g \in F,\ f.g \in F$

Example: Again considering the R-D problem

- Assume that each quadruple is in a separate basic block
- $OUT[B] = GEN[B] \cup (IN[B] - KILL[B])$
- In its general form, this becomes $f(x) = G \cup (x - K)$
- F consists of such functions f, one for each basic block
- Identity function exists here (when both G and K (GEN and $KILL$) are empty)

Reaching Definitions Framework - Example



Transfer functions:

$f_{d1}(x) = \{d1\} \cup (x - \{d4\})$
$f_{d2}(x) = \{d2\} \cup (x - \{d3\})$
$f_{d3}(x) = \{d3\} \cup (x - \{d2\})$
$f_{d4}(x) = \{d4\} \cup (x - \{d1\})$
$f_{d5}(x) = \{d5\} \cup (x - \Phi)$

Transfer functions for start and stop blocks are identity functions

$$\begin{aligned} f_{B1} &= (f_{d2}.f_{d1})(x) \\ &= \{d2\} \cup (\{d1\} \cup (x - \{d4\}) - \{d3\}) \\ &= \{d1,d2\} \cup (x - \{d3,d4\}) \\ f_{B2} &= (f_{d4}.f_{d3})(x) \\ &= \{d3,d4\} \cup (x - \{d1,d2\}) \\ f_{B3} &= f_{d5} = \{d5\} \cup x \end{aligned}$$

Monotone and Distributive Frameworks

- A DF framework $(D, F, V, \wedge)$ is monotone, if $\forall x, y \in V,\ f \in F,\ x \leq y \Rightarrow f(x) \leq f(y)$, OR $f(x \wedge y) \leq f(x) \wedge f(y)$
- The reaching definitions framework is monotone
- A DF framework is distributive, if $\forall x, y \in V,\ f \in F,\ f(x \wedge y) = f(x) \wedge f(y)$
- Distributivity $\Rightarrow$ monotonicity, but not vice-versa
- The reaching definitions lattice is distributive

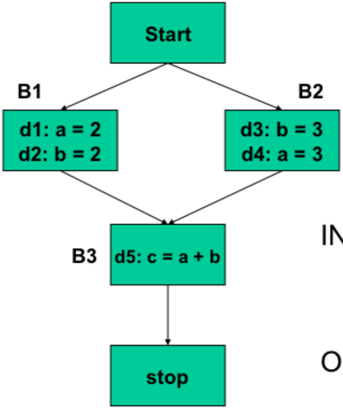
Iterative Algorithm for DFA (forward flow)

{$OUT[B1] = v_{init}$;
for each block $B \neq B1$ do $OUT[B] = \top$;
while (*changes to any OUT occur*) do
  for each block $B \neq B1$ do {

$$\begin{aligned} IN[B] &= \bigwedge_{P \text{ a predecessor of } B} OUT[P]; \\ OUT[B] &= f_B(IN[B]); \end{aligned}$$

  }
}

Reaching Definitions Framework - Example contd.



f_{B1} = {d1,d2} U (x – {d3,d4})

f_{B2} = {d3,d4} U (x – {d1,d2})

f_{B3} = {d5} U x

$$IN[B] = \bigwedge_{P,\ \text{a predecessor of } B} OUT[P]$$

$$= \bigcup_{P,\text{a predecessor of } B} OUT[P]$$

$$OUT[B] = f_B(IN[B])$$

Needs 2 iterations to converge
IN[B1] = IN[B2] = Φ; OUT[B1] = {d1,d2}; OUT[B2] = {d3,d4}
IN[B3] = OUT[B1] U OUT[B2] = {d1,d2,d3,d4}
OUT[B3] = {d5} U IN[B3] = {d1,d2,d3,d4,d5}