Data-Flow Analysis

Adapted From Lectures by
Prof. Saman Amarasinghe (MIT)

Outline

• Data-flow analysis
• Available expressions
• Algorithm for calculating available expressions
• Bit sets
• Formulating a data-flow analysis problem
• DU chains
• SSA form

Data-Flow Analysis

• Local Analysis
  – Analyze the effect of each instruction
  – Compose effects of instructions to derive information from beginning of basic block to each instruction
• Data-Flow Analysis
  – Iteratively propagate basic block information over the control-flow graph until no changes
  – Calculate the final value at the beginning of the basic block
• Local Propagation
  – Propagate the information from the beginning of the basic block to each instruction

Data-Flow Analysis

• Overview of data-flow analysis
• Available expressions
• Algorithm for calculating available expressions
• Bit sets
• Formulating a data-flow analysis problem
• DU chains
• SSA form

Example: Available Expression

• An expression is available if and only if
  – All paths of execution reaching the current point pass through the point where the expression was defined
  – No variable used in the expression was modified between the definition point and the current point

Example: Available Expression
Is the Expression Available?

**YES!**

Is the Expression Available?

**YES!**

Is the Expression Available?

**NO!**

Is the Expression Available?

**NO!**

Is the Expression Available?

**NO!**

Is the Expression Available?

**YES!**

Is the Expression Available?

**YES!**

Is the Expression Available?

**NO!**

Is the Expression Available?

**YES!**
Is the Expression Available?

\[ a = b + c \]
\[ d = e + f \]
\[ f = a + c \]

Use of Available Expressions

\[ a = b + c \]
\[ d = e + f \]
\[ f = a + c \]

\[ g = a + c \]
\[ j = a + b + c + d \]
\[ b = a + d \]
\[ h = c + f \]

Use of Available Expressions

\[ a = b + c \]
\[ d = e + f \]
\[ f = a + c \]

\[ g = f \]
\[ j = f + b + d \]
\[ b = a + d \]
\[ h = c + f \]
Use of Available Expressions

- $a = b + c$
- $d = e + f$
- $f = a + c$
- $g = f$
- $j = f + b + d$
- $b = a + d$
- $h = c + f$

Outline

- Overview of data-flow analysis
- Available expressions
- Algorithm for calculating available expressions
- Bit sets
- Formulating a data-flow analysis problem
- DU chains
- SSA form

Example: Available Expression

- Assign a number to each expression

Gen and Kill sets

- **Gen set**
  - If the current basic block (or instruction) creates the definition, it is in the gen set
  - The entry should be in the output no matter what
- **Kill set**
  - If the current basic block (or instruction) redefines a variable in the expression, it is in the kill set
  - Expression is not valid after that

Algorithm for Available Expression

- Assign a number to each expression
- Calculate gen and kill sets for each instruction

Gen and Kill sets
**Gen and Kill sets**

1. \( a = b + c \)
   - gen = \{ \( b + c \) \}
   - kill = \{ any expr with a \}

2. \( d = e + f \)
   - gen = \{ \( e + f \) \}
   - kill = \{ any expr with d \}

3. \( f = a + c \)
   - gen = \{ \( a + c \) \}
   - kill = \{ any expr with f \}

4. \( g = a + c \)

5. \( b = a + d \)

6. \( h = c + f \)

7. \( j = a + b + c + d \)

---

**Algorithm for Available Expression**

- Assign a number to each expression
- Calculate gen and kill sets for each instruction
- Calculate aggregate gen and kill sets for each basic block

---

**Aggregate Gen set**

- The gen set in the current expression should be in the OutGEN set

\[ \text{OutGEN} = \text{gen} \]
Aggregate Gen set

- The gen set in the current expression should be in the OutGEN set.
- Any expression in the InGEN set that is not killed should be in the OutGEN set.

OutGEN = gen ∪ (InGEN - kill)
aggregate gen set

\[ a = b + c \]

\[ \text{gen} = \{1\} \]
\[ \text{kill} = \{3, 4, 5, 7\} \]
\[ \text{InGEN} = \{1\} \]

\[ e + f \]
\[ \text{gen} = \{2\} \]
\[ \text{kill} = \{3, 7\} \]
\[ \text{OutGEN} = \{2\} \cup \left(\{1\} \cdot \{5, 7\}\right) \]

\[ a + c \]
\[ \text{gen} = \{3\} \]
\[ \text{kill} = \{2, 6\} \]
\[ \text{OutGEN} = \text{gen} \cup \left(\text{InGEN} - \text{kill}\right) \]
Aggregate Kill set

\[ a = b + c \]
\[ \text{gen} = \{ 1 \} \]
\[ \text{kill} = \{ 3, 4, 5, 7 \} \]

OutKILL set

OutKILL = kill

Aggregate Kill set

- The kill set in the current expression should be in the OutKILL set
- Any set in the InKILL should be in OutKILL

OutKILL = kill \cup \text{InKILL}
Aggregate Kill set

\[ \text{InKILL} = \{ \} \]
\[ a = b + c \]
\[ \text{OutKILL} = \{ 3, 4, 5, 7 \} \]
\[ d = e + f \]
\[ \text{InKILL} = \{ 3, 4, 5, 7 \} \]
\[ f = a + c \]
\[ \text{gen} = \{ 1 \} \]
\[ \text{kill} = \{ 3, 4, 5, 7 \} \]
\[ \text{OutKILL} = \text{kill} \cup \text{InKILL} \]

\[ \text{gen} = \{ 2 \} \]
\[ \text{kill} = \{ 3, 7 \} \]
\[ \text{OutKILL} = \{ 3, 7 \} \cup \{ 3, 4, 5, 7 \} \]

\[ \text{gen} = \{ 3 \} \]
\[ \text{kill} = \{ 2, 6 \} \]
Algorithm for Available Expression

- Assign a number to each expression
- Calculate gen and kill sets for each instruction
- Calculate aggregate gen and kill sets for each basic block
- Initialize available set at each basic block to be the entire set

Iteratively propagate available expression set over the CFG
Propagate available expression set

- If the expression is generated (in the gen set) then it is available at the end
  - should be in the OUT set

\[ \text{OUT} = \text{gen} \]

Aggregate Gen and Kill sets

\[ \text{IN} = \bigcap \text{OUT} \]
\[ \text{OUT} = \text{gen} \cup (\text{IN} - \text{kill}) \]

IN = \{ 1, 2, 3, 4, 5, 6, 7 \}
\[ \text{Gen} = \{ 1, 3 \} \]
\[ \text{Kill} = \{ 2, 3, 4, 5, 6, 7 \} \]
\[ \text{IN} = \bigcap \text{OUT} \]
\[ \text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \} \]

\[ \text{IN} = \{ \} \]
\[ \text{Gen} = \{ 1, 3 \} \]
\[ \text{Kill} = \{ 2, 3, 4, 5, 6, 7 \} \]
\[ \text{IN} = \bigcap \text{OUT} \]
\[ \text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \} \]
Aggregate Gen and Kill sets

\[ a = b + c \]
\[ d = e + f \]
\[ f = a + c \]
\[ g = a + c \]
\[ j = a + b + c + d \]
\[ b = a + d \]
\[ h = c + f \]

\[ \text{Gen} = \{ 1, 3 \} \]
\[ \text{Kill} = \{ 2, 3, 4, 5, 6, 7 \} \]
\[ \text{Gen} = \{ 4 \} \]
\[ \text{Kill} = \{ \} \]
\[ \text{Gen} = \{ 5, 6 \} \]
\[ \text{Kill} = \{ 1, 7 \} \]
\[ \text{Gen} = \{ 7 \} \]
\[ \text{Kill} = \{ \} \]

\[ \text{IN} = \{ 1, 2, 3, 4, 5, 6, 7 \} \]
\[ \text{OUT} = \{ 1, 3 \} \]
\[ \text{IN} = \{ 2, 3, 4, 5, 6, 7 \} \]
\[ \text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \} \]

\[ \text{IN} = \{ 1, 3, 4 \} \]
\[ \text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \} \]
Aggregate Gen and Kill sets

IN = \cap OUT
OUT = gen \cup (IN - kill)

IN = \{ 1, 3 \}
OUT = \{ 1, 2, 3, 4, 5, 6, 7 \}

\text{IN} = \{ 1, 3 \}
\text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \}

\text{IN} = \{ 1, 3 \}
\text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \}

\text{IN} = \{ 1, 3 \}
\text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \}

\text{IN} = \{ 1, 3 \}
\text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \}

\text{IN} = \{ 1, 3 \}
\text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \}

\text{IN} = \{ 1, 3 \}
\text{OUT} = \{ 1, 2, 3, 4, 5, 6, 7 \}
Algorithm for Available Expression

- Assign a number to each expression
- Calculate gen and kill sets for each instruction
- Calculate aggregate gen and kill sets for each basic block
- Initialize available set at each basic block to be the entire set
- Iteratively propagate available expression set over the CFG
- Propagate within the basic block

Propagate within the basic block

- Start with the IN set of available expressions
- Linearly propagate down the basic block
- same as data-flow step
- single pass since no back edges

\[
\text{OUT} = \text{gen} \cup (\text{IN} - \text{kill})
\]

Available Expressions

- \(a = b + c\)
- \(\text{gen} = \{1\}\)
- \(\text{kill} = \{3, 4, 5, 7\}\)

Outline

- Overview of data-flow analysis
- Available expressions
- Algorithm for calculating available expressions
- Bit sets
- Formulating a data-flow analysis problem
- DU chains
- SSA form

Bitsets

- Assign a bit to each element of the set
  - Union \(\Rightarrow\) bit OR
  - Intersection \(\Rightarrow\) bit AND
  - Subtraction \(\Rightarrow\) bit NEGATE and AND
- Fast implementation
  - 32 elements packed to each word
  - AND and OR are single instructions

Kill Set vs. Preserve Set

- Kill Sets
  - \(\text{OUT} = \text{gen} \cup (\text{IN} - \text{kill})\)
  - Using bit vectors: \(\text{OUT} = \text{gen} \lor (\text{IN} - \text{kill})\)
  - Subtraction \(\Rightarrow\) bit NEGATE and AND
  - \(\text{OUT} = \text{gen} \lor (\text{IN} \land \neg \text{kill})\)
- Preserve Sets
  - \(\text{PRSV} = \text{Entire Set} - \text{KILL}\)
  - \(\text{OUT} = \text{gen} \lor (\text{IN} \land \text{prsv})\)
  - \(\text{OUT} = \text{gen} \lor (\text{IN} \land \neg \text{prsv})\)

(Refer to Muchnick book)
Formulating a data-flow analysis problem

- Lattice
  - Abstract quantities over which the analysis will operate
    - Example: sets of available expressions

- Flow functions
  - how each control-flow and computational construct affects the abstract quantities
    - Example: the OUT equation for each statement

Lattice

- A lattice \( L \) consists of
  - a set of values
  - two operations meet (\( \wedge \)) and join (\( \vee \))
  - a top value (1) and a bottom value (\( \perp \))
Lattice

- Example: the lattice for the reaching definition problem when there are only 3 definitions

\[ T = \{ \{ d1, d2, d3 \}, \{ d1, d3 \}, \{ d2 \}, \{ \} \} \]

Meet and Join Operations

- Meet and Join forms a closure
  - For all \( a, b \in L \) there exist a unique \( c \) and \( d \in L \) such that
    \[ a \land b = c \quad a \lor b = d \]
- Meet and Join are commutative
  - \( a \land b = b \land a \)
  - \( a \lor b = b \lor a \)
- Meet and Join are associative
  - \( (a \land b) \land c = b \land (a \land c) \quad (a \lor b) \lor c = b \lor (a \lor c) \)
- There exist a unique top element (\( T \)) and bottom element (\( \bot \)) in \( L \) such that
  \[ a \land \bot = \bot \quad a \lor T = T \]
Meet and Join Operations

\{ d_1, d_2 \} \lor \{ d_3 \} = ???

T = \{ d_1, d_2, d_3 \}
\{ d_1, d_3 \}
\{ d_2 \}
\{ d_1, d_2 \} \lor \{ d_3 \} = ???

Meet and Join Operations

• Meet Operation
  – Set Intersection
  – Follow the lines downwards from the two elements in the lattice until they meet at a single unique element

• Join Operation
  – Set Union
  – There is a unique element in the lattice from where there is a downwards path (with no shared segment) to both elements
**Partial Order**

- Define $a \subseteq b$ if and only if $a \land b = b$
- Properties
  - Reflexive: $a \subseteq a$
  - Antisymmetric: $a \subseteq b$ and $b \subseteq a$ $\Rightarrow$ $a = b$
  - Transitive: $a \subseteq b$ and $b \subseteq c$ $\Rightarrow$ $a \subseteq c$

**Lattice Height**

- The height of the lattice is the longest ascending chain in it
  - $(T, a, b, c, \ldots, \bot)$

**Flow Functions**

- Example: $OUT = f(IN)$
- $f: L \rightarrow L$ where $L$ is a lattice
- Properties
  - Monotone: $\forall a,b \in L$ $a \subseteq b$ $\Rightarrow$ $f(a) \subseteq f(b)$
- Fixed Point
  - A fixed point is an element $a \in L$ such that $f(a) = a$

**Intuition about Termination**

- Data-flow analysis starts assuming most optimistic values ($T$)
- Each stage applies a flow function
  - $V_{\text{new}} \subseteq V_{\text{prev}}$
  - Moves downwards in the lattice
- Until stable (values don’t change)
  - A fixed point is reached at every basic block
- Lattice has a finite height $\Rightarrow$ should terminate
Outline
• Overview of data-flow analysis
• Available expressions
• Algorithm for calculating available expressions
• Bit sets
• Formulating a data-flow analysis problem
• DU chains
• SSA form

Def-Use and Use-Def Chains
• Def-Use (DU) Chain
  – Connects a definition of each variable to all the possible uses of that variable
• Use-Def (UD) Chain
  – Connects a use of a variable to all the possible definitions of that variable

DU Chain Data-Flow Problem Formulation
• Lattice: The set of definitions
  – Bitvector format: a bit for each definition in the procedure
• Flow direction: Forward Flow
• Flow Functions:
  – gen = \{ b_0 \ldots b_n \mid b_k = 1 \text{ if the } k\text{th definition} \}
  – kill = \{ b_0 \ldots b_n \mid b_k = 1 \text{ if } k\text{th variable is redefined} \}
  – OUT = gen \cup (IN - kill)
  – IN = \bigcup OUT

Formulate the UD Chain Data-Flow Problem
• Lattice:
  – Bitvector format:
• Flow direction: Forward/Backward Flow
• Flow Functions:
  – gen = \{ b_0 \ldots b_n \mid b_k = 1 \}
  – kill = \{ b_0 \ldots b_n \mid b_k = 1 \}
  – OUT =
  – IN =

DU Example

DU Example
```
DU Example
entry
i = 1
j = 2

j = j * 2
k = true
i = i + 1

print j
i = i + 1

exit

i < n

gen = {1, 2, 3}
kill = {4, 5, 6, 7}

OUT = {1, 2, 3}
IN = {1, 2, 3}
```

DU Example

entry

OUT = \{ \}

k = false
i = 1
j = 2
j = j * 2
k = true
i = i + 1
print j
i = i + 1

exit

i < n

gen = \{ 1, 2, 3 \}
kill = \{ 4, 5, 6, 7 \}
gen = \{ 4, 5, 6 \}
kill = \{ 1, 2, 3, 7 \}
gen = \{ \}
kill = \{ \}
gen = \{ \}
kill = \{ \}
gen = \{ 7 \}
kill = \{ 2, 6 \}

OUT = \{ 1, 2, 3 \}
IN = \{ 1, 2, 3, 4, 5, 6 \}
OUT = IN = \{ \}
OUT = IN = \{ 1, 2, 3 \}
OUT = IN = \{ 1, 2, 3 \}
OUT = IN = \{ 4, 5, 6 \} OUT = \{ 1, 2, 3 \} OUT = \{ 1, 3, 7 \}
IN = \{ 1, 2, 3, 7 \}
DU Example

```plaintext
k = false
i = 1
j = 2

j = j * 2
k = true
i = i + 1

print j
```

DU Chains

- At each use of a variable, points to all the possible definitions
  - Very useful information
  - Used in many optimizations

- Incorporate this information in the representation
  - SSA Form

Outline

- Overview of control-flow analysis
- Available expressions
- Algorithm for calculating available expressions
- Bit sets
- Formulating a data-flow analysis problem
- DU chains
- SSA form

Static Single Assignment (SSA) Form

- Each definition has a unique variable name
  - Original name + a version number

- Each use refers to a definition by name

- What about multiple possible definitions?
  - Add special merge nodes so that there can be only a single definition ($\Phi$ functions)
Static Single Assignment (SSA) Form

a = 1
b = a + 2
c = a + b
a = a + 1
d = a + b

Static Single Assignment (SSA) Form

a = 1
c = a + 2
b = 1
j = j + 2
k = true
i = i + 1
print j
exit

i < n

DU Example

copy
k = false
i = 1
j = 2

j = j * 2
k = true
i = i + 1
print j
exit

i < n