Program Analysis

Data Flow Analysis (Part 3)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2020/2021

Outline

- First example: Available expressions
- Basic principles
- More examples ←
- Solving data flow problems
- Inter-procedural analysis
- Sensitivities

Data Flow Analyses

- **Seen previously**
  - Available expressions
- **Next**
  - Reaching definitions
  - Very busy expressions
  - Live variables
- **Course project**
  - Taint analysis

Reaching Definitions Analysis

Goal: For each program point, compute which assignments may have been made and may not have been overwritten

- Useful in various program analyses
- E.g., to compute a data flow graph

Example

```
var x = 5;
var y = 1;
while (x > 1) {
  y = x * y;
  x = x - 1;
}
```

Example

```
var x = 5;
var y = 1;
while (x > 1) {
  y = x * y;
  x = x - 1;
}
```

Definition reaches entry of this statement

Example

```
var x = 5;
var y = 1;
while (x > 1) {
  y = x * y;
  x = x - 1;
}
```

All definitions reach the entry of this statement

Example

```
var x = 5;
var y = 1;
while (x > 1) {
  y = x * y;
  x = x - 1;
}
```

Three definitions reach entry of this statement

Defining the Analysis

- **Domain: Definitions (i.e., assigments) in the code**
  - Set of pairs (v, s) of variables and stmts.
  - (v, s) means a definition of v at s
- **Direction: Forward**
- **Meet operator: Union**
  - Because we care about definitions that *may* reach a program point

Defining the Analysis (2)

- **Transfer function:**

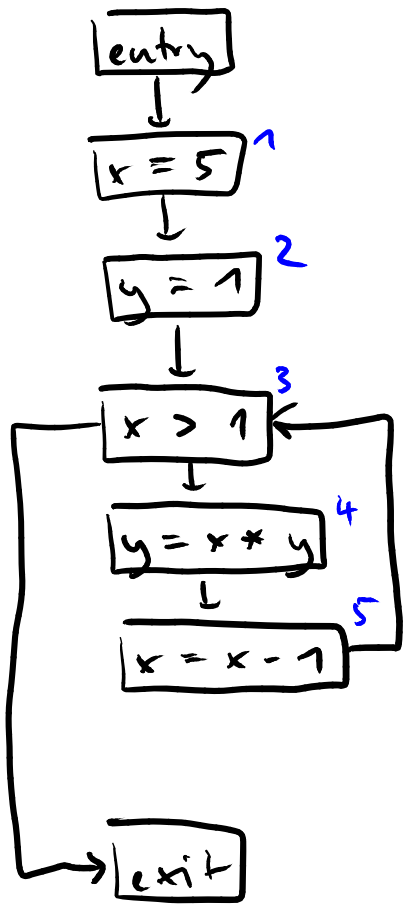
  $$RD_{exit}(s) = (RD_{entry}(s) \setminus kill(s)) \cup gen(S)$$

- **Function** $gen(s)$
  - If s is assignment to v: (v, s)
  - Otherwise: Empty set
- **Function** $kill(s)$
  - If s is assignment to v: (v, s') for all s' that define v
  - Otherwise: Empty set

Defining the Analysis (3)

- **Boundary condition: Entry node starts with all variables undefined**
  - Special "statement" for undefined variables: ?
  - $RD_{entry}(entryNode) = \{(v, ?) | v \in Vars\}$
- **Initially, all nodes have no reaching definitions**

Example: Reaching Definitions



s	gen(s)	kill(s)
1	$\{(x,1)\}$	$\{(x,1),(x,5),(x,?)\}$
2	$\{(y,2)\}$	$\{(y,2),(y,4),(y,?)\}$
3	$\emptyset$	$\emptyset$
4	$\{(y,4)\}$	$\{(y,2),(y,4),(y,?)\}$
5	$\{(x,5)\}$	$\{(x,1),(x,5),(x,?)\}$

Data flow equations

$RD_{entry}(1) = \{(x,?),(y,?)\}$

$RD_{entry}(2) = RD_{exit}(1)$

$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$

$RD_{entry}(4) = RD_{exit}(3)$

$RD_{entry}(5) = RD_{exit}(4)$

$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(x,1),(x,5),(x,?)\} \cup \{(x,1)\}$

$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(y,2),(y,4),(y,?)\} \cup \{(y,2)\}$

$RD_{exit}(3) = RD_{entry}(3)$

$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(y,2),(y,4),(y,?)\} \cup \{(y,4)\}$

$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(x,1),(x,5),(x,?)\} \cup \{(x,5)\}$

Solution

s	$RD_{entry}(s)$	$RD_{exit}(s)$
1	$\{(x,?),(y,?)\}$	$\{(x,1),(y,?)\}$
2	...	...
3	$\{(x,1),(y,2),(y,4),(x,5\}$	...
4	..	..
5	..	$\{(y,4),(x,5)\}$

Outline

- First example: Available expressions
- Basic principles
- More examples ←
- Solving data flow problems
- Inter-procedural analysis
- Sensitivities

Very Busy Expression Analysis

Goal: For each program point, find expressions that must be very busy

- ”Very busy”: On all future paths, expression will be used before any of the variables in it are redefined
- Useful for program optimizations, e.g., hoisting
  - Hoisting an expression: Pre-compute it, e.g., before entering a block, for later use

Example

```
if (a > b) {
  x = b - a;
  y = a - b;
} else {
  y = b - a;
  x = a - b;
}
```

Example

```
if (a > b) {
  x = b - a;
  y = a - b;
} else {
  y = b - a;
  x = a - b;
}
```

`a - b` and `b - a` are very busy here

Defining the Analysis

- **Domain: All non-trivial expressions occurring in the code**
- **Direction: Backward**
- **Meet operator: Intersection**
  - Because we care about very busy expressions that *must* be used

Defining the Analysis (2)

Transfer function:

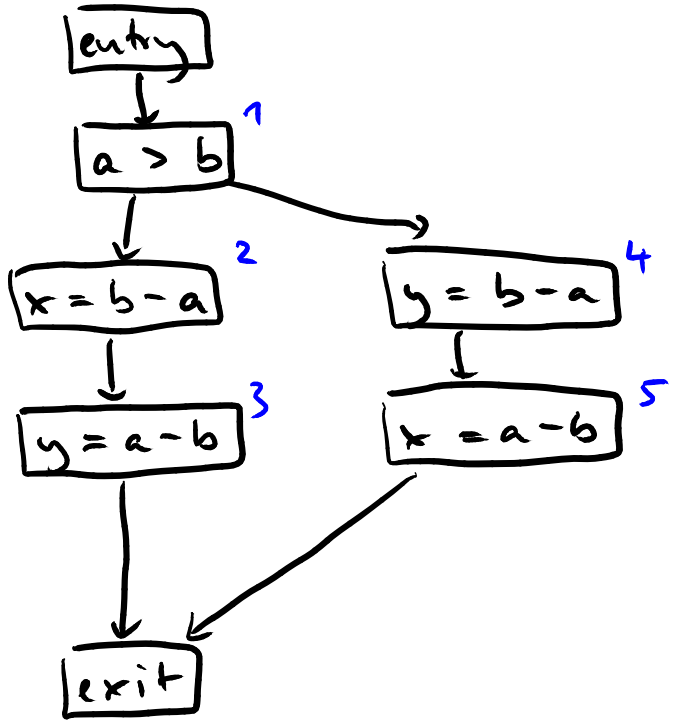
$$VB_{entry}(s) = (VB_{exit}(s) \setminus kill(s)) \cup gen(S)$$

- Backward analysis: Returns expressions that are very busy expressions at entry of statement
- Function $gen(s)$
  - All expressions e that appear in s
- Function $kill(s)$
  - If s assigns to x, all expressions in which x occurs
  - Otherwise: Empty set

Defining the Analysis (3)

- **Boundary condition: Final node starts with no very busy expressions**
  - $VB_{exit}(finalNode) = \emptyset$
- **Initially, all nodes have no very busy expressions**

Example: Very busy expr. analysis



s	gen(s)	kill(s)
1	$\emptyset$	$\emptyset$
2	{b-a}	$\emptyset$
3	{a-b}	$\emptyset$
4	{b-a}	$\emptyset$
5	{a-b}	$\emptyset$

s	VBentry(s)	VBexit(s)
1	{a-b, b-a}	{a-b, b-a}
2	{a-b, b-a}	{a-b}
3	{a-b}	$\emptyset$
4	{a-b, b-a}	{a-b}
5	{a-b}	$\emptyset$

Live Variables Analysis

Goal: For each statement, find variables that are may be "live" at the exit from the statement

- "Live": The variable is used before being redefined
- Useful, e.g., for identifying dead code
  - Bug detection: Dead assignments are typically unexpected
  - Optimization: Remove dead code

Example

```
x = 2;
y = 4;
x = 1;
if (y > x) {
  z = y;
} else {
  z = y * y;
  x = z;
}
```

Example

```
x = 2;
y = 4;
x = 1;
if (y > x) {
  z = y;
} else {
  z = y * y;
  x = z;
}
```

`x` is not live after this statement

Example

```
x = 2;
y = 4;
x = 1;
if (y > x) {
  z = y;
} else {
  z = y * y;
  x = z;
}
```

Both `x` and `y` are live after this statement

Defining the Analysis

- **Domain: All variables occurring in the code**
- **Direction: Backward**
- **Meet operator: Union**
  - Because we care about whether a variable *may* be used

Defining the Analysis (2)

Transfer function:

$$LV_{entry}(s) = (LV_{exit}(s) \setminus kill(s)) \cup gen(S)$$

- Backward analysis: Returns set of variables that are live at entry of statement
- Function $gen(s)$
  - All variables v that are used in s
- Function $kill(s)$
  - If s assigns to x, then it kills x
  - Otherwise: Empty set

Defining the Analysis (3)

- **Boundary condition: Final node starts with no live variables**
  - $LV_{exit}(finalNode) = \emptyset$
- **Initially, all nodes have no live variables**

Quiz: Live Variables

```
x = 2;
y = 4;
x = 1;
if (y > x) {
  z = y;
} else {
  z = y * y;
  x = z;
}
```

Compute the live variables before and after every statement.

s	$LV_{entry}(s)$	$LV_{exit}(s)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{y\}$
3	$\{y\}$	$\{x, y\}$
4	$\{x, y\}$	$\{y\}$
5	$\{y\}$	$\emptyset$
6	$\{y\}$	$\{z\}$
7	$\{z\}$	$\emptyset$