Program Analysis – Lecture 5

Data Flow Analysis (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2019/2020

Warm-up Quiz

What does the following code print?

```
var a, b;
var x = {};
x[a] = 23;
console.log(x[b]);
```

Nothing **23** **undefined** **false**

https://ilias3.uni-stuttgart.de/vote/KN2I

Warm-up Quiz

What does the following code print?

```
var a, b;
var x = {};
x[a] = 23;
console.log(x[b]);
```

Nothing | **23** | **undefined** | **false**

https://ilias3.uni-stuttgart.de/vote/KN2I

Warm-up Quiz

What does the following code print?

```
var a, b;
var x = {};
x[a] = 23;
console.log(x[b]);
```

Have value `undefined`

Write and then read "undefined" property of `x`

Nothing | **23** | **undefined** | **false**

https://ilias3.uni-stuttgart.de/vote/KN2I

Data Flow Analysis

Basic idea

- Propagate analysis information along the edges of a control flow graph
- Goal: Compute analysis state at each program point
- For each statement, define how it affects the analysis state
- For loops: Iterate until fix-point reached

Available Expressions: Example

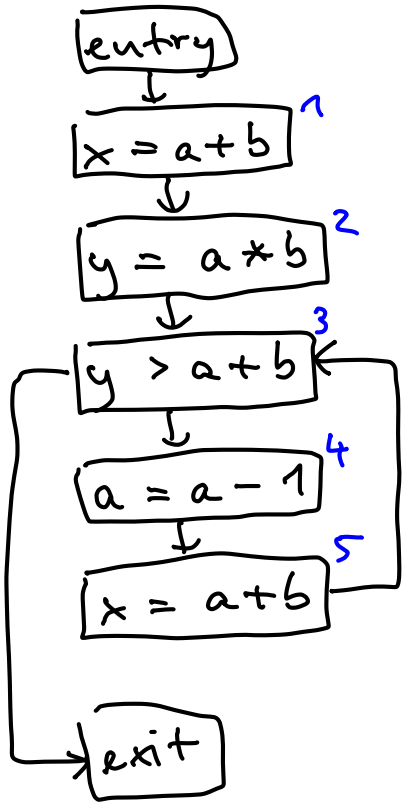
```
var x = a + b;
var y = a * b;
while (y > a + b) {
  a = a + 1;
  x = a + b;
}
```

Available Expressions: Example

```
var x = a + b;
var y = a * b;
while (y > a + b) {
  a = a + 1;
  x = a + b;
}
```

Available every time execution reaches this point

Control flow graph



7 nodes, 7 edges

Non-trivial expressions:

$a+b$

$a*b$

$a-1$

Transfer function for each statement

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

Propagating Available Expressions

- Initially, no available expressions
- Forward analysis: Propagate available expressions in the direction of control flow
- For each statement s, outgoing available expressions are:
  incoming avail. exprs. minus $kill(s)$ plus $gen(s)$
- When control flow splits, propagate available expressions both ways
- When control flows merge, intersect the incoming available expressions

Data flow equations

$AE_{entry}(s)$ .. avail. expr. at entry of s

$AE_{exit}(s)$ .. —"— exit —"—

$AE_{entry}(1) = \emptyset$

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

$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$

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

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

$AE_{exit}(1) = AE_{entry}(1) \cup \{a+b\}$

$AE_{exit}(2) = AE_{entry}(2) \cup \{a*b\}$

$AE_{exit}(3) = AE_{entry}(3) \cup \{a+b\}$

$AE_{exit}(4) = AE_{entry}(4) \setminus \{a+b, a*b, a+1\}$

$AE_{exit}(5) = AE_{entry}(5) \cup \{a+b\}$

Solution of these equations

s	$AE_{entry}(s)$	$AE_{exit}(s)$
1	$\emptyset$	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	$\boxed{\{a+b\}}$	$\{a+b\}$
4	$\{a+b\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

see example

Quiz

```
var m = x - y;
if (random()) {
  while (m > 0) {
    x = y + 1;
  }
} else {
  n = x - y;
}
z = x - y;
```

Is `x-y` an available expression when entering this statement?

Outline

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

Defining a Data Flow Analysis

Any data flow analysis:

Defined by six properties

- Domain
- Direction
- Transfer function
- Meet operator
- Boundary condition
- Initial values

Domain

- Analysis associates some information with every program point
- "Information" means elements of a set
- Domain of the analysis: All possible elements the set may have
- E.g., for available expressions analysis:
  Domain is set of non-trivial expressions

Direction

- **Analysis propagates information along the control flow graph**
- **Forward analysis: Normal flow of control**
- **Backward analysis: Invert all edges**
  - Reasons about executions in reverse
- **E.g., available expression analysis: Forward**

Transfer Function

- **Defines how a statement affects the propagated information**
- $DF_{exit}(s) =$ **some function of** $DF_{entry}(s)$
- **E.g., for available expression analysis:**
  $AE_{exit}(s) = (AE_{entry} \setminus kill(s)) \cup gen(s)$

Meet Operator

- **What if two statements s_1, s_2 flow to a statement s?**
  - Forward analysis: Execution branches merge
  - Backward analysis: Branching point
- **Meet operator defines how to combine the incoming information**
  - Union: $DF_{entry}(s) = DF_{exit}(s_1) \cup DF_{exit}(s_2)$
  - Intersection: $DF_{entry}(s) = DF_{exit}(s_1) \cap DF_{exit}(s_2)$

Meet Operator

- **What if two statements s_1, s_2 flow to a statement s?**
  - Forward analysis: Execution branches merge
  - Backward analysis: Branching point
- **Meet operator defines how to combine the incoming information**
  - Union: $DF_{entry}(s) = DF_{exit}(s_1) \cup DF_{exit}(s_2)$
  - Intersection: $DF_{entry}(s) = DF_{exit}(s_1) \cap DF_{exit}(s_2)$

E.g., available expressions analysis

Boundary Condition

- **What information to start with at the first CFG node?**
  - Forward analysis: First node is enty node
  - Backward analysis: First node is exit node
- **Common choices**
  - Empty set
  - Entire domain

Boundary Condition

- **What information to start with at the first CFG node?**
  - Forward analysis: First node is enty node
  - Backward analysis: First node is exit node
- **Common choices**
  - Empty set
  - Entire domain

E.g., available expressions analysis

Initial Values

- **What is the information to start with at intermediate nodes?**
- **Common choices**
  - Empty set
  - Entire domain

Initial Values

- **What is the information to start with at intermediate nodes?**
- **Common choices**
  - Empty set
  - Entire domain

E.g., available expressions analysis

Defining a Data Flow Analysis

Any data flow analysis:

Defined by six properties

- Domain
- Direction
- Transfer function

- Meet operator
- Boundary condition
- Initial values

Defining a Data Flow Analysis

Any data flow analysis:

Defined by six properties

Domain	Non-trivial expressions
Direction	Forward
Transfer function	$AE_{exit}(s) = (AE_{entry} \setminus kill(s)) \cup gen(s)$
Meet operator	Intersection ($\cap$)
Boundary condition	$AE_{entry}(entryNode) = \emptyset$
Initial values	$\emptyset$

Example: Available expressions

Outline

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

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**
- **Direction: Forward**
- **Meet operator: Union**
  - Because we care about definitions that *may* reach a program point

Transfer Function

Defined via gen and $kill$ functions (again)

- Returns set of pairs (v, s) of variables and stmts
- (v, s) means a definition of v at 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

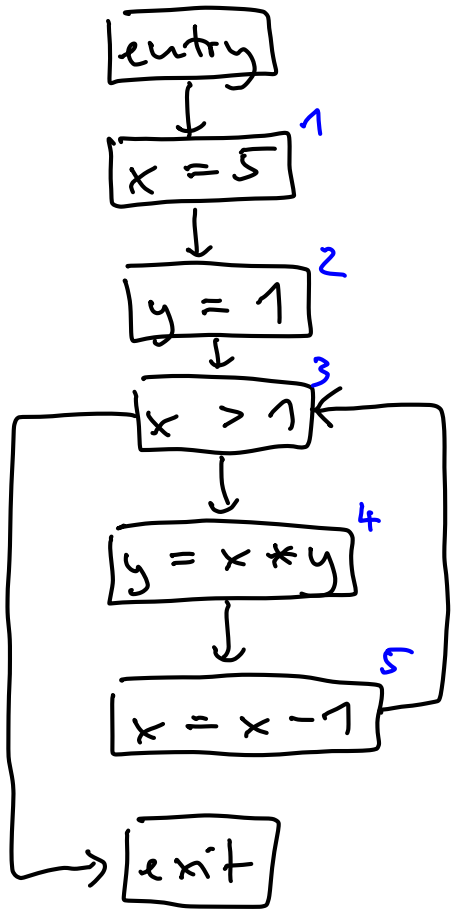
Defining the Analysis (2)

- **Special "statement" for undefined variables: ?**
- **Transfer function:**

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

- **Boundary condition: Entry node starts with all variables undefined**
  - $RD_{entry}(entryNode) = \{(v, ?) | v \in Vars\}$
- **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	$\{(x,5)\}$	$\{(x,1),(x,5),(x,?)\}$

$\Sigma = 16$

Data flow equation

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

" $(2) = RDexit(1)$

" $(3) = RDexit(2) \cup RDexit(5)$

" $(4) = RDexit(3)$

" $(5) = $ " (4)

$RDexit(1) = (RDentry(1) \setminus \{(x,1), (x,5), (x,?)\}) \cup \{x,1\}$

$RDexit(2) = \ldots$ (similar)

$RDexit(3) = RDentry(3)$

" $(4) =$

" $(5) = \ldots$ (similar)

Solution

s	RDentry(s)	RDexit(s)
1	$\{(x,?),(y,?)\}$	$\{(y,?),(x,1)\}$
2	...	..
3	$\{(x,1),(y,2),(y,5),(x,5)\}$	...
4	..	..
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 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

Transfer Function

Defined via gen and $kill$ functions (again)

- Returns set of expressions
- Backward analysis, i.e., returned expressions 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

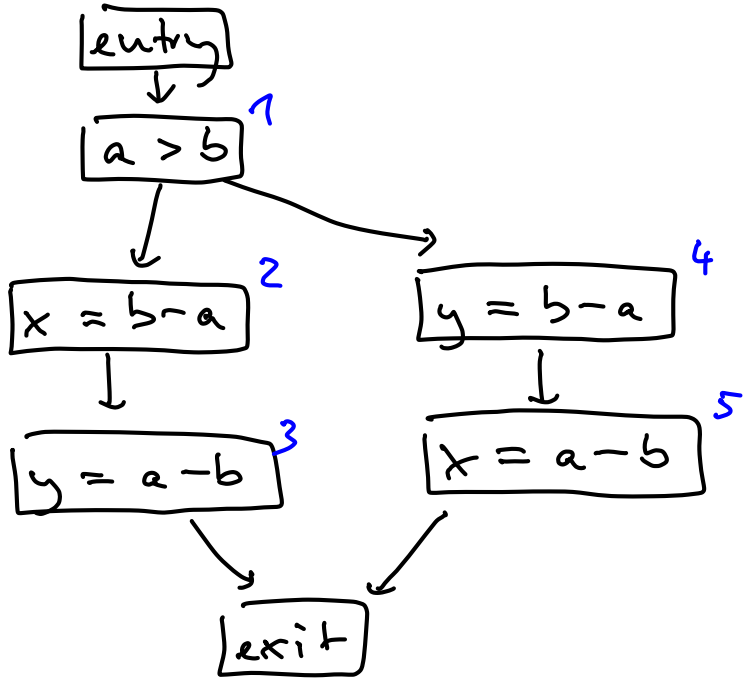
Defining the Analysis (2)

- **Transfer function:**

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

- **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 expression 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	$VB_{entry}(s)$	$VB_{exit}(s)$
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}	$\emptyset$