Program Analysis

Data Flow Analysis (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2023/2024

Warm-up Quiz

What does this Python code print?

```
one   = all([])
two   = all([[]])
three = all([[[]]])

print(f"{one}, {two}, {three}")
```

Warm-up Quiz

What does this Python code print?

```
one   = all([])
two   = all([[]])
three = all([[[]]])

print(f"{one}, {two}, {three}")
```

Answer: True, False, True

Warm-up Quiz

What does this Python code print?

```
one   = all([])
two   = all([[]])
three = all([[[]]])

print(f"{one}, {two}, {three}")
```

Returns `True` except if any element of the iterable evaluates to `False`

Answer: True, False, True

Warm-up Quiz

What does this Python code print?

```
one   = all([])
two   = all([[]])
three = all([[[]]])

print(f"{one}, {two}, {three}")
```

Returns `True` except if any element of the iterable evaluates to `False`

Empty list $\Rightarrow$ `True`

Answer: True, False, True

Warm-up Quiz

What does this Python code print?

```
one   = all([])
two   = all([[]])
three = all([[[]]])

print(f"{one}, {two}, {three}")
```

Returns `True` except if any element of the iterable evaluates to `False`

Inner, empty list evaluates to `False` ⇒ `False`

Answer: True, False, True

Warm-up Quiz

What does this Python code print?

```
one   = all([])
two   = all([[]])
three = all([[[]]])

print(f"{one}, {two}, {three}")
```

Returns `True` except if any element of the iterable evaluates to `False`

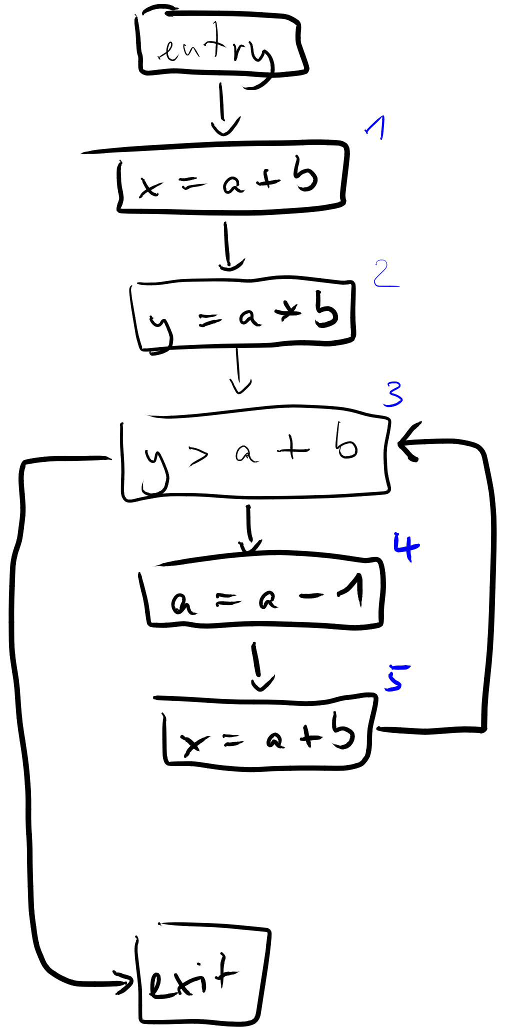
Inner, non-empty list evaluates to `True` ⇒ `True`

Answer: True, False, True

Example

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

Control flow graph



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	$\emptyset$	$\{a - 1, a + b, a * b\}$
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

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

$$AE_{entry}(1) = \emptyset$$
$$AE_{entry}(2) = AE_{exit}(1)$$
$$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$$
$$AE_{entry}(4) = AE_{exit}(3)$$
$$AE_{entry}(5) = AE_{exit}(4)$$
$$AE_{exit}(1) = AE_{entry}(1) \cup \{a+b\}$$
$$AE_{exit}(2) = AE_{entry}(2) \cup \{a*b\}$$
$$AE_{exit}(3) = AE_{entry}(3) \cup \{a+b\}$$
$$AE_{exit}(4) = AE_{entry}(4) \setminus \{a+b, a*b, a-1\}$$
$$AE_{exit}(5) = AE_{entry}(5) \cup \{a+b\}$$

Solution of these equations:

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

Quiz

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

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?

Quiz

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

No, because modifying `x` kills `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:**
  $AE_{exit}(s) = (AE_{entry}(s) \setminus kill(s)) \cup gen(s)$

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

- Non-trivial expressions
- Forward
- $AE_{exit}(s) = (AE_{entry} \setminus kill(s)) \cup gen(s)$
- Intersection ($\cap$)
- $AE_{entry}(entryNode) = \emptyset$
- $\emptyset$

Example: Available expressions

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

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 (assignments) in the code**
  - Set of pairs (v, s) of variables and statements
  - (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**:

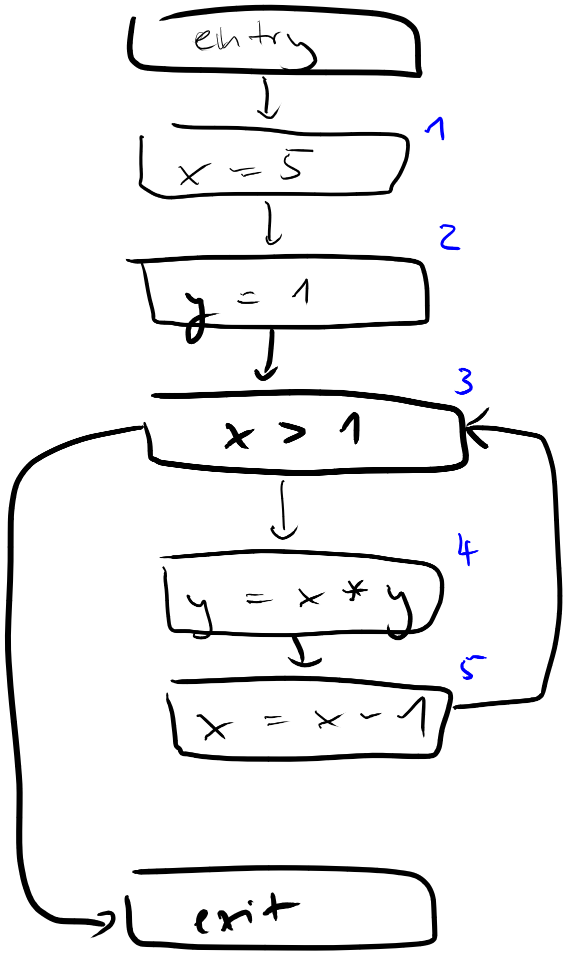
  $RD_{exit}(s) = (RD_{entry}(s) \setminus kill(s)) \cup gen(S)$

- **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: ?
  - $RD_{entry}(entryNode) = \{(v, ?) \mid v \in Vars\}$
- **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	∅	∅
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}	RD_{exit}
1	$\{(x,?), (y,?)\}$	$\{(x,1), (y,?)\}$
2	$\{(x,1), (y,?)\}$	$\{(x,1), (y,2)\}$
3	$\{(x,1), (x,5), (y,2), (y,4)\}$	$\{(x,1), (x,5), (y,2), (y,4)\}$
4	— " —	$\{(x,1), (x,5), (y,4)\}$
5	$\{(x,1), (x,5), (y,4)\}$	$\{(y,4), (x,5)\}$