

# Static Program Analysis

## Lecture 3: Dataflow Analysis II (Live Variables Analysis)

Thomas Noll

Lehrstuhl für Informatik 2  
(Software Modeling and Verification)

RWTH Aachen University

`noll@cs.rwth-aachen.de`

`http://www-i2.informatik.rwth-aachen.de/i2/spa11/`

Summer Semester 2011

- 1 Repetition: Dataflow Analysis
- 2 Another Example: Live Variables Analysis
- 3 Heading for a Dataflow Analysis Framework
- 4 Order-Theoretic Foundations: the Domain

# Labeled Programs

- Goal: **localization** of analysis information
- Dataflow information will be associated with
  - **skip** statements
  - assignments
  - tests in conditionals (**if**) and loops (**while**)
- Assume set of **labels**  $L$  with meta variable  $l \in L$  (usually  $L = \mathbb{N}$ )

## Definition (Labeled WHILE programs)

The **syntax of labeled WHILE programs** is defined by the following context-free grammar:

$$\begin{aligned} a &::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp \\ b &::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\ c &::= [\text{skip}]^l \mid [x := a]^l \mid c_1 ; c_2 \mid \\ &\quad \text{if } [b]^l \text{ then } c_1 \text{ else } c_2 \mid \text{while } [b]^l \text{ do } c \in Cmd \end{aligned}$$

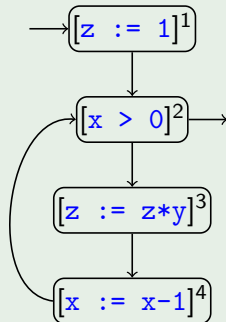
- All labels in  $c \in Cmd$  assumed distinct, denoted by  $L_c$
- Labeled fragments of  $c$  called **blocks**, denoted by  $Blk_c$

## Example

```
c = [z := 1]1;  
  while [x > 0]2 do  
    [z := z*y]3;  
    [x := x-1]4
```

```
init(c) = 1  
final(c) = {2}  
flow(c) = {(1, 2), (2, 3), (3, 4), (4, 2)}
```

Visualization by  
(control) flow graph:



# Goal of Available Expressions Analysis

## Available Expressions Analysis

The goal of **Available Expressions Analysis** is to determine, for each program point, which (complex) expressions *must* have been computed, and not later modified, on all paths to the program point.

## Available Expressions Analysis

The goal of **Available Expressions Analysis** is to determine, for each program point, which (complex) expressions *must* have been computed, and not later modified, on all paths to the program point.

- can be used to **avoid recomputations** of expressions
- only interesting for non-trivial (i.e., complex) arithmetic expressions

# Goal of Available Expressions Analysis

## Available Expressions Analysis

The goal of **Available Expressions Analysis** is to determine, for each program point, which (complex) expressions *must* have been computed, and not later modified, on all paths to the program point.

- can be used to **avoid recomputations** of expressions
- only interesting for non-trivial (i.e., complex) arithmetic expressions

## Example (Available Expressions Analysis)

```
[x := a+b]1;  
[y := a*b]2;  
while [y > a+b]3 do  
  [a := a+1]4;  
  [x := a+b]5
```

# Goal of Available Expressions Analysis

## Available Expressions Analysis

The goal of **Available Expressions Analysis** is to determine, for each program point, which (complex) expressions *must* have been computed, and not later modified, on all paths to the program point.

- can be used to **avoid recomputations** of expressions
- only interesting for non-trivial (i.e., complex) arithmetic expressions

## Example (Available Expressions Analysis)

```
[x := a+b]1;  
[y := a*b]2;  
while [y > a+b]3 do  
  [a := a+1]4;  
  [x := a+b]5
```

- **a+b available at label 3**



# Goal of Available Expressions Analysis

## Available Expressions Analysis

The goal of **Available Expressions Analysis** is to determine, for each program point, which (complex) expressions *must* have been computed, and not later modified, on all paths to the program point.

- can be used to **avoid recomputations** of expressions
- only interesting for non-trivial (i.e., complex) arithmetic expressions

## Example (Available Expressions Analysis)

```
[x := a+b]1;  
[y := a*b]2;  
while [y > a+b]3 do  
  [a := a+1]4;  
  [x := a+b]5
```

- **a+b** available at label 3
- **a+b** not available at label 5

# Goal of Available Expressions Analysis

## Available Expressions Analysis

The goal of **Available Expressions Analysis** is to determine, for each program point, which (complex) expressions *must* have been computed, and not later modified, on all paths to the program point.

- can be used to **avoid recomputations** of expressions
- only interesting for non-trivial (i.e., complex) arithmetic expressions

## Example (Available Expressions Analysis)

```
[x := a+b]1;  
[y := a*b]2;  
while [y > a+b]3 do  
  [a := a+1]4;  
  [x := a+b]5
```

- **a+b** available at label 3
- **a+b** not available at label 5
- possible optimization:  
 while [y > **x**]<sup>3</sup> do

# The Equation System

**Reminder:**  $AE_l = \begin{cases} \emptyset & \text{if } l = \text{init}(c) \\ \bigcap \{ \varphi_{l'}(AE_{l'}) \mid (l', l) \in \text{flow}(c) \} & \text{otherwise} \end{cases}$   
 $\varphi_{l'}(E) = (E \setminus \text{kill}_{AE}(B^{l'})) \cup \text{gen}_{AE}(B^{l'})$

## Example (AE equation system)

```
c = [x := a+b]1;  
    [y := a*b]2;  
    while [y > a+b]3 do  
        [a := a+1]4;  
        [x := a+b]5
```

| $l \in L_c$ | $\text{kill}_{AE}(B^l)$ | $\text{gen}_{AE}(B^l)$ |
|-------------|-------------------------|------------------------|
| 1           | $\emptyset$             | $\{a+b\}$              |
| 2           | $\emptyset$             | $\{a*b\}$              |
| 3           | $\emptyset$             | $\{a+b\}$              |
| 4           | $\{a+b, a*b, a+1\}$     | $\emptyset$            |
| 5           | $\emptyset$             | $\{a+b\}$              |

Equations:

$$AE_1 = \emptyset$$

$$AE_2 = \varphi_1(AE_1) = AE_1 \cup \{a+b\}$$

$$AE_3 = \varphi_2(AE_2) \cap \varphi_5(AE_5) \\ = (AE_2 \cup \{a*b\}) \cap (AE_5 \cup \{a+b\})$$

$$AE_4 = \varphi_3(AE_3) = AE_3 \cup \{a+b\}$$

$$AE_5 = \varphi_4(AE_4) = AE_4 \setminus \{a+b, a*b, a+1\}$$

Solution:  $AE_1 = \emptyset$

$$AE_2 = \{a+b\}$$

$$AE_3 = \{a+b\}$$

$$AE_4 = \{a+b\}$$

$$AE_5 = \emptyset$$

- 1 Repetition: Dataflow Analysis
- 2 Another Example: Live Variables Analysis
- 3 Heading for a Dataflow Analysis Framework
- 4 Order-Theoretic Foundations: the Domain

# Goal of Live Variables Analysis

## Live Variables Analysis

The goal of **Live Variables Analysis** is to determine, for each program point, which variables *may* be live at the exit from the point.

## Live Variables Analysis

The goal of **Live Variables Analysis** is to determine, for each program point, which variables *may* be live at the exit from the point.

- A variable is called **live** at the exit from a block if there exists a path from the block to a use of the variable that does not re-define the variable

## Live Variables Analysis

The goal of **Live Variables Analysis** is to determine, for each program point, which variables *may* be live at the exit from the point.

- A variable is called **live** at the exit from a block if there exists a path from the block to a use of the variable that does not re-define the variable
- All variables considered to be live at the **end** of the program (alternative: restriction to output variables)

## Live Variables Analysis

The goal of **Live Variables Analysis** is to determine, for each program point, which variables *may* be live at the exit from the point.

- A variable is called **live** at the exit from a block if there exists a path from the block to a use of the variable that does not re-define the variable
- All variables considered to be live at the **end** of the program (alternative: restriction to output variables)
- Can be used for **Dead Code Elimination**:  
remove assignments to non-live variables



## Example 3.1 (Live Variables Analysis)

```
[x := 2]1;  
[y := 4]2;  
[x := 1]3;  
if [y > 0]4 then  
    [z := x]5  
else  
    [z := y*y]6;  
[x := z]7
```

## Example 3.1 (Live Variables Analysis)

```
[x := 2]1;  
[y := 4]2;  
[x := 1]3;  
if [y > 0]4 then  
  [z := x]5  
else  
  [z := y*y]6;  
[x := z]7
```

- x not live at exit from label 1

## Example 3.1 (Live Variables Analysis)

```
[x := 2]1;  
[y := 4]2;  
[x := 1]3;  
if [y > 0]4 then  
    [z := x]5  
else  
    [z := y*y]6;  
[x := z]7
```

- x not live at exit from label 1
- y live at exit from 2

## Example 3.1 (Live Variables Analysis)

```
[x := 2]1;  
[y := 4]2;  
[x := 1]3;  
if [y > 0]4 then  
  [z := x]5  
else  
  [z := y*y]6;  
[x := z]7
```

- x not live at exit from label 1
- y live at exit from 2
- x live at exit from 3

## Example 3.1 (Live Variables Analysis)

```
[x := 2]1;  
[y := 4]2;  
[x := 1]3;  
if [y > 0]4 then  
  [z := x]5  
else  
  [z := y*y]6;  
[x := z]7
```

- **x** not live at exit from label 1
- **y** live at exit from 2
- **x** live at exit from 3
- **z** live at exits from 5 and 6

## Example 3.1 (Live Variables Analysis)

```
[x := 2]1;  
[y := 4]2;  
[x := 1]3;  
if [y > 0]4 then  
  [z := x]5  
else  
  [z := y*y]6;  
[x := z]7
```

- x not live at exit from label 1
- y live at exit from 2
- x live at exit from 3
- z live at exits from 5 and 6
- possible optimization: remove [x := 2]<sup>1</sup>

- A variable on the left-hand side of an assignment is **killed** by the assignment; tests and **skip** do not kill

- A variable on the left-hand side of an assignment is **killed** by the assignment; tests and **skip** do not kill
- Formally:  $\text{kill}_{\text{LV}} : \text{Blk}_c \rightarrow 2^{\text{Var}_c}$  is defined by

$$\begin{aligned}\text{kill}_{\text{LV}}([\text{skip}]') &:= \emptyset \\ \text{kill}_{\text{LV}}([x := a]') &:= \{x\} \\ \text{kill}_{\text{LV}}([b]') &:= \emptyset\end{aligned}$$



- A variable on the left-hand side of an assignment is **killed** by the assignment; tests and **skip** do not kill
- Formally:  $\text{kill}_{\text{LV}} : \text{Blk}_c \rightarrow 2^{\text{Var}_c}$  is defined by

$$\begin{aligned}\text{kill}_{\text{LV}}([\text{skip}]') &:= \emptyset \\ \text{kill}_{\text{LV}}([x := a]') &:= \{x\} \\ \text{kill}_{\text{LV}}([b]') &:= \emptyset\end{aligned}$$

- Every reading access **generates** a live variable

- A variable on the left-hand side of an assignment is **killed** by the assignment; tests and **skip** do not kill
- Formally:  $\text{kill}_{\text{LV}} : \text{Blk}_c \rightarrow 2^{\text{Var}_c}$  is defined by

$$\begin{aligned}\text{kill}_{\text{LV}}([\text{skip}]') &:= \emptyset \\ \text{kill}_{\text{LV}}([x := a]') &:= \{x\} \\ \text{kill}_{\text{LV}}([b]') &:= \emptyset\end{aligned}$$

- Every reading access **generates** a live variable
- Formally:  $\text{gen}_{\text{LV}} : \text{Blk}_c \rightarrow 2^{\text{Var}_c}$  is defined by

$$\begin{aligned}\text{gen}_{\text{LV}}([\text{skip}]') &:= \emptyset \\ \text{gen}_{\text{LV}}([x := a]') &:= \text{Var}_a \\ \text{gen}_{\text{LV}}([b]') &:= \text{Var}_b\end{aligned}$$

## Example 3.2 ( $\text{kill}_{\text{LV}}$ / $\text{gen}_{\text{LV}}$ functions)

```
c = [x := 2]1;  
    [y := 4]2;  
    [x := 1]3;  
    if [y > 0]4 then  
        [z := x]5  
    else  
        [z := y*y]6;  
    [x := z]7
```

## Example 3.2 ( $\text{kill}_{\text{LV}}/\text{gen}_{\text{LV}}$ functions)

```
c = [x := 2]1;  
    [y := 4]2;  
    [x := 1]3;  
    if [y > 0]4 then  
        [z := x]5  
    else  
        [z := y*y]6;  
    [x := z]7
```

- $\text{Var}_c = \{x, y, z\}$

## Example 3.2 ( $\text{kill}_{\text{LV}}/\text{gen}_{\text{LV}}$ functions)

```
c = [x := 2]1;  
    [y := 4]2;  
    [x := 1]3;  
    if [y > 0]4 then  
        [z := x]5  
    else  
        [z := y*y]6;  
    [x := z]7
```

- $\text{Var}_c = \{x, y, z\}$

- $l \in L_c \quad \text{kill}_{\text{LV}}(B^l) \quad \text{gen}_{\text{LV}}(B^l)$

|   |     |     |
|---|-----|-----|
| 1 | {x} | ∅   |
| 2 | {y} | ∅   |
| 3 | {x} | ∅   |
| 4 | ∅   | {y} |
| 5 | {z} | {x} |
| 6 | {z} | {y} |
| 7 | {x} | {z} |

- For each  $l \in L_c$ ,  $LV_l \subseteq Var_c$  represents the set of **live variables at the exit of block  $B^l$**

- For each  $l \in L_c$ ,  $LV_l \subseteq Var_c$  represents the set of **live variables at the exit of block  $B^l$**
- Formally, for a program  $c \in Cmd$  with isolated exits:

$$LV_l = \begin{cases} Var_c & \text{if } l \in \text{final}(c) \\ \bigcup \{ \varphi_{l'}(LV_{l'}) \mid (l, l') \in \text{flow}(c) \} & \text{otherwise} \end{cases}$$

where  $\varphi_{l'} : 2^{Var_c} \rightarrow 2^{Var_c}$  denotes the **transfer function** of block  $B^{l'}$ , given by

$$\varphi_{l'}(V) := (V \setminus \text{kill}_{LV}(B^{l'})) \cup \text{gen}_{LV}(B^{l'})$$

- For each  $I \in L_c$ ,  $LV_I \subseteq Var_c$  represents the set of **live variables at the exit of block  $B^I$**
- Formally, for a program  $c \in Cmd$  with isolated exits:

$$LV_I = \begin{cases} Var_c & \text{if } I \in \text{final}(c) \\ \bigcup \{ \varphi_{I'}(LV_{I'}) \mid (I, I') \in \text{flow}(c) \} & \text{otherwise} \end{cases}$$

where  $\varphi_{I'} : 2^{Var_c} \rightarrow 2^{Var_c}$  denotes the **transfer function** of block  $B^{I'}$ , given by

$$\varphi_{I'}(V) := (V \setminus \text{kill}_{LV}(B^{I'})) \cup \text{gen}_{LV}(B^{I'})$$

- Characterization of analysis:
  - backward**: starts in  $\text{final}(c)$  and proceeds upwards
  - may**:  $\bigcup$  in equation for  $LV_I$
  - flow-sensitive**: results depending on order of assignments



- For each  $I \in L_c$ ,  $LV_I \subseteq Var_c$  represents the set of **live variables at the exit of block  $B^I$**
- Formally, for a program  $c \in Cmd$  with isolated exits:

$$LV_I = \begin{cases} Var_c & \text{if } I \in \text{final}(c) \\ \bigcup \{ \varphi_{I'}(LV_{I'}) \mid (I, I') \in \text{flow}(c) \} & \text{otherwise} \end{cases}$$

where  $\varphi_{I'} : 2^{Var_c} \rightarrow 2^{Var_c}$  denotes the **transfer function** of block  $B^{I'}$ , given by

$$\varphi_{I'}(V) := (V \setminus \text{kill}_{LV}(B^{I'})) \cup \text{gen}_{LV}(B^{I'})$$

- Characterization of analysis:
  - backward**: starts in  $\text{final}(c)$  and proceeds upwards
  - may**:  $\bigcup$  in equation for  $LV_I$
  - flow-sensitive**: results depending on order of assignments
- Later: solution **not necessarily unique**  
 $\implies$  choose **least one**

# The Equation System II

**Reminder:**

$$LV_I = \begin{cases} Var_c & \text{if } I \in \text{final}(c) \\ \bigcup \{ \varphi_{I'}(LV_{I'}) \mid (I, I') \in \text{flow}(c) \} & \text{otherwise} \end{cases}$$
$$\varphi_{I'}(V) = (V \setminus \text{kill}_{LV}(B_{I'})) \cup \text{gen}_{LV}(B_{I'})$$

# The Equation System II

**Reminder:**  $LV_I = \begin{cases} Var_c & \text{if } I \in \text{final}(c) \\ \bigcup \{ \varphi_{I'}(LV_{I'}) \mid (I, I') \in \text{flow}(c) \} & \text{otherwise} \end{cases}$   
 $\varphi_{I'}(V) = (V \setminus \text{kill}_{LV}(B_{I'})) \cup \text{gen}_{LV}(B_{I'})$

## Example 3.3 (LV equation system)

```
c = [x := 2]1; [y := 4]2;  
    [x := 1]3;  
    if [y > 0]4 then  
        [z := x]5  
    else  
        [z := y*y]6;  
    [x := z]7
```

# The Equation System II

**Reminder:**  $LV_I = \begin{cases} Var_c & \text{if } I \in \text{final}(c) \\ \bigcup \{ \varphi_{I'}(LV_{I'}) \mid (I, I') \in \text{flow}(c) \} & \text{otherwise} \end{cases}$

$\varphi_{I'}(V) = (V \setminus \text{kill}_{LV}(B^{I'})) \cup \text{gen}_{LV}(B^{I'})$

## Example 3.3 (LV equation system)

```

c = [x := 2]1; [y := 4]2;
    [x := 1]3;
    if [y > 0]4 then
      [z := x]5
    else
      [z := y*y]6;
      [x := z]7

```

$I \in L_c \quad \text{kill}_{LV}(B^I) \quad \text{gen}_{LV}(B^I)$

|   |     |     |
|---|-----|-----|
| 1 | {x} | ∅   |
| 2 | {y} | ∅   |
| 3 | {x} | ∅   |
| 4 | ∅   | {y} |
| 5 | {z} | {x} |
| 6 | {z} | {y} |
| 7 | {x} | {z} |

# The Equation System II

**Reminder:**  $LV_I = \begin{cases} Var_c & \text{if } I \in \text{final}(c) \\ \bigcup \{ \varphi_{I'}(LV_{I'}) \mid (I, I') \in \text{flow}(c) \} & \text{otherwise} \end{cases}$

$\varphi_{I'}(V) = (V \setminus \text{kill}_{LV}(B^{I'})) \cup \text{gen}_{LV}(B^{I'})$

## Example 3.3 (LV equation system)

```

c = [x := 2]1; [y := 4]2;
    [x := 1]3;
    if [y > 0]4 then
        [z := x]5
    else
        [z := y*y]6;
        [x := z]7

```

$I \in L_c$     $\text{kill}_{LV}(B^I)$     $\text{gen}_{LV}(B^I)$

|   |     |     |
|---|-----|-----|
| 1 | {x} | ∅   |
| 2 | {y} | ∅   |
| 3 | {x} | ∅   |
| 4 | ∅   | {y} |
| 5 | {z} | {x} |
| 6 | {z} | {y} |
| 7 | {x} | {z} |

$LV_1 = \varphi_2(LV_2) = LV_2 \setminus \{y\}$   
 $LV_2 = \varphi_3(LV_3) = LV_3 \setminus \{x\}$   
 $LV_3 = \varphi_4(LV_4) = LV_4 \cup \{y\}$   
 $LV_4 = \varphi_5(LV_5) \cup \varphi_6(LV_6)$   
 $\quad = ((LV_5 \setminus \{z\}) \cup \{x\}) \cup ((LV_6 \setminus \{z\}) \cup \{y\})$   
 $LV_5 = \varphi_7(LV_7) = (LV_7 \setminus \{x\}) \cup \{z\}$   
 $LV_6 = \varphi_7(LV_7) = (LV_7 \setminus \{x\}) \cup \{z\}$   
 $LV_7 = \{x, y, z\}$

# The Equation System II

**Reminder:**  $LV_I = \begin{cases} Var_c & \text{if } I \in \text{final}(c) \\ \bigcup \{ \varphi_{I'}(LV_{I'}) \mid (I, I') \in \text{flow}(c) \} & \text{otherwise} \end{cases}$   
 $\varphi_{I'}(V) = (V \setminus \text{kill}_{LV}(B_{I'})) \cup \text{gen}_{LV}(B_{I'})$

## Example 3.3 (LV equation system)

```
c = [x := 2]1; [y := 4]2;
    [x := 1]3;
    if [y > 0]4 then
      [z := x]5
    else
      [z := y*y]6;
      [x := z]7
```

$I \in L_c$     $\text{kill}_{LV}(B^I)$     $\text{gen}_{LV}(B^I)$

|   |     |     |
|---|-----|-----|
| 1 | {x} | ∅   |
| 2 | {y} | ∅   |
| 3 | {x} | ∅   |
| 4 | ∅   | {y} |
| 5 | {z} | {x} |
| 6 | {z} | {y} |
| 7 | {x} | {z} |

$$\begin{aligned} LV_1 &= \varphi_2(LV_2) = LV_2 \setminus \{y\} \\ LV_2 &= \varphi_3(LV_3) = LV_3 \setminus \{x\} \\ LV_3 &= \varphi_4(LV_4) = LV_4 \cup \{y\} \\ LV_4 &= \varphi_5(LV_5) \cup \varphi_6(LV_6) \\ &= ((LV_5 \setminus \{z\}) \cup \{x\}) \cup ((LV_6 \setminus \{z\}) \cup \{y\}) \\ LV_5 &= \varphi_7(LV_7) = (LV_7 \setminus \{x\}) \cup \{z\} \\ LV_6 &= \varphi_7(LV_7) = (LV_7 \setminus \{x\}) \cup \{z\} \\ LV_7 &= \{x, y, z\} \end{aligned}$$

**Solution:**

$$\begin{aligned} LV_1 &= \emptyset \\ LV_2 &= \{y\} \\ LV_3 &= \{x, y\} \\ LV_4 &= \{x, y\} \\ LV_5 &= \{y, z\} \\ LV_6 &= \{y, z\} \\ LV_7 &= \{x, y, z\} \end{aligned}$$

- 1 Repetition: Dataflow Analysis
- 2 Another Example: Live Variables Analysis
- 3 Heading for a Dataflow Analysis Framework**
- 4 Order-Theoretic Foundations: the Domain

# Similarities Between Analysis Problems

- **Observation:** the analyses presented so far have some similarities



# Similarities Between Analysis Problems

- **Observation:** the analyses presented so far have some **similarities**

⇒ Look for underlying **framework**

# Similarities Between Analysis Problems

- **Observation:** the analyses presented so far have some **similarities**

⇒ Look for underlying **framework**

- **Advantage:** possibility for designing (efficient) **generic algorithms for solving dataflow equations**

# Similarities Between Analysis Problems

- **Observation:** the analyses presented so far have some **similarities**

⇒ Look for underlying **framework**

- **Advantage:** possibility for designing (efficient) **generic algorithms for solving dataflow equations**
- **Overall pattern:** for  $c \in Cmd$  and  $l \in L_c$ , the **analysis information** (AI) is described by **equations** of the form

$$AI_l = \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup \{ \varphi_{l'}(AI_{l'}) \mid (l', l) \in F \} & \text{otherwise} \end{cases}$$

where

- the set of **extremal labels**,  $E$ , is  $\{\text{init}(c)\}$  or  $\{\text{final}(c)\}$
- $\iota$  specifies the **extremal analysis information**
- the **combination operator**,  $\bigsqcup$ , is  $\cap$  or  $\cup$
- $\varphi_{l'}$  denotes the **transfer function** of block  $B_{l'}$
- the **flow relation**  $F$  is  $\text{flow}(c)$  or  $\text{flow}^R(c) (:= \{(l', l) \mid (l, l') \in \text{flow}(c)\})$

- **Direction of information flow:**

- **forward:**

- $F = \text{flow}(c)$
    - $AI_I$  concerns entry of  $B^I$
    - $c$  has isolated entry

- **backward:**

- $F = \text{flow}^R(c)$
    - $AI_I$  concerns exit of  $B^I$
    - $c$  has isolated exits

- **Direction of information flow:**

- **forward:**

- $F = \text{flow}(c)$
    - $AI_I$  concerns entry of  $B^I$
    - $c$  has isolated entry

- **backward:**

- $F = \text{flow}^R(c)$
    - $AI_I$  concerns exit of  $B^I$
    - $c$  has isolated exits

- **Quantification over paths:**

- **may:**

- $\sqcup = \bigcup$
    - property satisfied by some path
    - interested in least solution (later)

- **must:**

- $\sqcap = \bigcap$
    - property satisfied by all paths
    - interested in greatest solution (later)

**Goal:** solve dataflow equation system by **fixpoint iteration**

- ① Characterize solution of equation system as **fixpoint** of a transformation

**Goal:** solve dataflow equation system by **fixpoint iteration**

- ① Characterize solution of equation system as **fixpoint** of a transformation
- ② Introduce **partial order** for comparing analysis results

**Goal:** solve dataflow equation system by **fixpoint iteration**

- ① Characterize solution of equation system as **fixpoint** of a transformation
- ② Introduce **partial order** for comparing analysis results
- ③ Establish **least upper bound** as combination operator



**Goal:** solve dataflow equation system by **fixpoint iteration**

- ① Characterize solution of equation system as **fixpoint** of a transformation
- ② Introduce **partial order** for comparing analysis results
- ③ Establish **least upper bound** as combination operator
- ④ Ensure **monotonicity** of transfer functions

**Goal:** solve dataflow equation system by **fixpoint iteration**

- ① Characterize solution of equation system as **fixpoint** of a transformation
- ② Introduce **partial order** for comparing analysis results
- ③ Establish **least upper bound** as combination operator
- ④ Ensure **monotonicity** of transfer functions
- ⑤ Guarantee termination of fixpoint iteration by **ascending chain condition**

**Goal:** solve dataflow equation system by **fixpoint iteration**

- ① Characterize solution of equation system as **fixpoint** of a transformation
- ② Introduce **partial order** for comparing analysis results
- ③ Establish **least upper bound** as combination operator
- ④ Ensure **monotonicity** of transfer functions
- ⑤ Guarantee termination of fixpoint iteration by **ascending chain condition**
- ⑥ Optimize fixpoint iteration by **worklist algorithm**

- 1 Repetition: Dataflow Analysis
- 2 Another Example: Live Variables Analysis
- 3 Heading for a Dataflow Analysis Framework
- 4 Order-Theoretic Foundations: the Domain

- **Wanted:** solution of (dataflow) equation system

- **Wanted:** solution of (dataflow) equation system
- **Problem:** recursive dependencies between dataflow variables

- **Wanted:** **solution** of (dataflow) equation system
- **Problem:** **recursive dependencies** between dataflow variables
- **Idea:** characterize solution as **fixpoint** of transformation:

$$(AI_I = \tau_I)_{I \in L_c} \iff \Phi((AI_I)_{I \in L_c}) = (AI_I)_{I \in L_c}$$

where  $\Phi((AI_I)_{I \in L_c}) := (\tau_I)_{I \in L_c}$

- **Wanted:** solution of (dataflow) equation system
- **Problem:** recursive dependencies between dataflow variables
- **Idea:** characterize solution as fixpoint of transformation:

$$(AI_I = \tau_I)_{I \in L_c} \iff \Phi((AI_I)_{I \in L_c}) = (AI_I)_{I \in L_c}$$

where  $\Phi((AI_I)_{I \in L_c}) := (\tau_I)_{I \in L_c}$

- **Approach:** approximate fixpoint by iteration



The domain of analysis information usually forms a partial order where the ordering relation compares the “precision” of information.

## Definition 3.4 (Partial order)

A **partial order (PO)**  $(D, \sqsubseteq)$  consists of a set  $D$ , called **domain**, and of a relation  $\sqsubseteq \subseteq D \times D$  such that, for every  $d_1, d_2, d_3 \in D$ ,

**reflexivity:**  $d_1 \sqsubseteq d_1$

**transitivity:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

**antisymmetry:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always  $d_1 \sqsubseteq d_2$  or  $d_2 \sqsubseteq d_1$ .

# Partial Orders

The domain of analysis information usually forms a partial order where the ordering relation compares the “precision” of information.

## Definition 3.4 (Partial order)

A **partial order (PO)**  $(D, \sqsubseteq)$  consists of a set  $D$ , called **domain**, and of a relation  $\sqsubseteq \subseteq D \times D$  such that, for every  $d_1, d_2, d_3 \in D$ ,

**reflexivity:**  $d_1 \sqsubseteq d_1$

**transitivity:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

**antisymmetry:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always  $d_1 \sqsubseteq d_2$  or  $d_2 \sqsubseteq d_1$ .

## Example 3.5

①  $(\mathbb{N}, \leq)$  is a total partial order

# Partial Orders

The domain of analysis information usually forms a partial order where the ordering relation compares the “precision” of information.

## Definition 3.4 (Partial order)

A **partial order (PO)**  $(D, \sqsubseteq)$  consists of a set  $D$ , called **domain**, and of a relation  $\sqsubseteq \subseteq D \times D$  such that, for every  $d_1, d_2, d_3 \in D$ ,

**reflexivity:**  $d_1 \sqsubseteq d_1$

**transitivity:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

**antisymmetry:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always  $d_1 \sqsubseteq d_2$  or  $d_2 \sqsubseteq d_1$ .

## Example 3.5

- ①  $(\mathbb{N}, \leq)$  is a total partial order
- ②  $(\mathbb{N}, <)$  is not a partial order (since not reflexive)

# Partial Orders

The domain of analysis information usually forms a partial order where the ordering relation compares the “precision” of information.

## Definition 3.4 (Partial order)

A **partial order (PO)**  $(D, \sqsubseteq)$  consists of a set  $D$ , called **domain**, and of a relation  $\sqsubseteq \subseteq D \times D$  such that, for every  $d_1, d_2, d_3 \in D$ ,

**reflexivity:**  $d_1 \sqsubseteq d_1$

**transitivity:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

**antisymmetry:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always  $d_1 \sqsubseteq d_2$  or  $d_2 \sqsubseteq d_1$ .

## Example 3.5

- ①  $(\mathbb{N}, \leq)$  is a total partial order
- ②  $(\mathbb{N}, <)$  is not a partial order (since not reflexive)
- ③ (Live Variables)  $(2^{Var_c}, \sqsubseteq)$  is a (non-total) partial order

# Partial Orders

The domain of analysis information usually forms a partial order where the ordering relation compares the “precision” of information.

## Definition 3.4 (Partial order)

A **partial order (PO)**  $(D, \sqsubseteq)$  consists of a set  $D$ , called **domain**, and of a relation  $\sqsubseteq \subseteq D \times D$  such that, for every  $d_1, d_2, d_3 \in D$ ,

**reflexivity:**  $d_1 \sqsubseteq d_1$

**transitivity:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$

**antisymmetry:**  $d_1 \sqsubseteq d_2$  and  $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always  $d_1 \sqsubseteq d_2$  or  $d_2 \sqsubseteq d_1$ .

## Example 3.5

- ①  $(\mathbb{N}, \leq)$  is a total partial order
- ②  $(\mathbb{N}, <)$  is not a partial order (since not reflexive)
- ③ (Live Variables)  $(2^{Var_c}, \sqsubseteq)$  is a (non-total) partial order
- ④ (Available Expressions)  $(2^{AExp_c}, \supseteq)$  is a (non-total) partial order

# Upper Bounds

In the dataflow equation system, analysis information from several predecessors is combined by taking the least upper bound.

## Definition 3.6 ((Least) upper bound)

Let  $(D, \sqsubseteq)$  be a partial order and  $S \subseteq D$ .

- 1 An element  $d \in D$  is called an **upper bound** of  $S$  if  $s \sqsubseteq d$  for every  $s \in S$  (notation:  $S \sqsubseteq d$ ).

In the dataflow equation system, analysis information from several predecessors is combined by taking the least upper bound.

## Definition 3.6 ((Least) upper bound)

Let  $(D, \sqsubseteq)$  be a partial order and  $S \subseteq D$ .

- 1 An element  $d \in D$  is called an **upper bound** of  $S$  if  $s \sqsubseteq d$  for every  $s \in S$  (notation:  $S \sqsubseteq d$ ).
- 2 An upper bound  $d$  of  $S$  is called **least upper bound (LUB)** or **supremum** of  $S$  if  $d \sqsubseteq d'$  for every upper bound  $d'$  of  $S$  (notation:  $d = \bigsqcup S$ ).

# Upper Bounds

In the dataflow equation system, analysis information from several predecessors is combined by taking the least upper bound.

## Definition 3.6 ((Least) upper bound)

Let  $(D, \sqsubseteq)$  be a partial order and  $S \subseteq D$ .

- 1 An element  $d \in D$  is called an **upper bound** of  $S$  if  $s \sqsubseteq d$  for every  $s \in S$  (notation:  $S \sqsubseteq d$ ).
- 2 An upper bound  $d$  of  $S$  is called **least upper bound (LUB)** or **supremum** of  $S$  if  $d \sqsubseteq d'$  for every upper bound  $d'$  of  $S$  (notation:  $d = \bigsqcup S$ ).

## Example 3.7

- 1  $S \subseteq \mathbb{N}$  has a LUB in  $(\mathbb{N}, \leq)$  iff it is finite



# Upper Bounds

In the dataflow equation system, analysis information from several predecessors is combined by taking the least upper bound.

## Definition 3.6 ((Least) upper bound)

Let  $(D, \sqsubseteq)$  be a partial order and  $S \subseteq D$ .

- 1 An element  $d \in D$  is called an **upper bound** of  $S$  if  $s \sqsubseteq d$  for every  $s \in S$  (notation:  $S \sqsubseteq d$ ).
- 2 An upper bound  $d$  of  $S$  is called **least upper bound (LUB)** or **supremum** of  $S$  if  $d \sqsubseteq d'$  for every upper bound  $d'$  of  $S$  (notation:  $d = \bigsqcup S$ ).

## Example 3.7

- 1  $S \subseteq \mathbb{N}$  has a LUB in  $(\mathbb{N}, \leq)$  iff it is finite
- 2 (Live Variables)  $(D, \sqsubseteq) = (2^{\text{Var}_c}, \subseteq)$ . Given  $V_1, \dots, V_n \subseteq \text{Var}_c$ ,  
$$\bigsqcup \{V_1, \dots, V_n\} = \bigcup \{V_1, \dots, V_n\}$$

# Upper Bounds

In the dataflow equation system, analysis information from several predecessors is combined by taking the least upper bound.

## Definition 3.6 ((Least) upper bound)

Let  $(D, \sqsubseteq)$  be a partial order and  $S \subseteq D$ .

- 1 An element  $d \in D$  is called an **upper bound** of  $S$  if  $s \sqsubseteq d$  for every  $s \in S$  (notation:  $S \sqsubseteq d$ ).
- 2 An upper bound  $d$  of  $S$  is called **least upper bound (LUB)** or **supremum** of  $S$  if  $d \sqsubseteq d'$  for every upper bound  $d'$  of  $S$  (notation:  $d = \bigsqcup S$ ).

## Example 3.7

- 1  $S \subseteq \mathbb{N}$  has a LUB in  $(\mathbb{N}, \leq)$  iff it is finite
- 2 (Live Variables)  $(D, \sqsubseteq) = (2^{Var_c}, \subseteq)$ . Given  $V_1, \dots, V_n \subseteq Var_c$ ,  
$$\bigsqcup \{V_1, \dots, V_n\} = \bigcup \{V_1, \dots, V_n\}$$
- 3 (Avail. Expr.)  $(D, \sqsubseteq) = (2^{AExp_c}, \supseteq)$ . Given  $A_1, \dots, A_n \subseteq AExp_c$ ,  
$$\bigsqcup \{A_1, \dots, A_n\} = \bigcap \{A_1, \dots, A_n\}$$

# Complete Lattices

Since  $\{\varphi_{I'}(A|_{I'}) \mid (I', I) \in F\}$  can contain arbitrary elements, the existence of least upper bounds must be ensured for arbitrary subsets.

## Definition 3.8 (Complete lattice)

A **complete lattice** is a partial order  $(D, \sqsubseteq)$  such that all subsets of  $D$  have least upper bounds. In this case,

$$\perp := \bigsqcup \emptyset$$

denotes the **least element** of  $D$ .

# Complete Lattices

Since  $\{\varphi_{I'}(A|_{I'}) \mid (I', I) \in F\}$  can contain arbitrary elements, the existence of least upper bounds must be ensured for arbitrary subsets.

## Definition 3.8 (Complete lattice)

A **complete lattice** is a partial order  $(D, \sqsubseteq)$  such that all subsets of  $D$  have least upper bounds. In this case,

$$\perp := \bigsqcup \emptyset$$

denotes the **least element** of  $D$ .

## Example 3.9

- 1  $(\mathbb{N}, \leq)$  is not a complete lattice as, e.g.,  $\mathbb{N}$  does not have a LUB

# Complete Lattices

Since  $\{\varphi_{I'}(AI_{I'}) \mid (I', I) \in F\}$  can contain arbitrary elements, the existence of least upper bounds must be ensured for arbitrary subsets.

## Definition 3.8 (Complete lattice)

A **complete lattice** is a partial order  $(D, \sqsubseteq)$  such that all subsets of  $D$  have least upper bounds. In this case,

$$\perp := \bigsqcup \emptyset$$

denotes the **least element** of  $D$ .

## Example 3.9

- ①  $(\mathbb{N}, \leq)$  is not a complete lattice as, e.g.,  $\mathbb{N}$  does not have a LUB
- ② (Live Variables)  
 $(D, \sqsubseteq) = (2^{Var_c}, \sqsubseteq)$  is a complete lattice with  $\perp = \emptyset$

# Complete Lattices

Since  $\{\varphi_{I'}(AI_{I'}) \mid (I', I) \in F\}$  can contain arbitrary elements, the existence of least upper bounds must be ensured for arbitrary subsets.

## Definition 3.8 (Complete lattice)

A **complete lattice** is a partial order  $(D, \sqsubseteq)$  such that all subsets of  $D$  have least upper bounds. In this case,

$$\perp := \bigsqcup \emptyset$$

denotes the **least element** of  $D$ .

## Example 3.9

- ①  $(\mathbb{N}, \leq)$  is not a complete lattice as, e.g.,  $\mathbb{N}$  does not have a LUB
- ② (Live Variables)  
 $(D, \sqsubseteq) = (2^{Var_c}, \sqsubseteq)$  is a complete lattice with  $\perp = \emptyset$
- ③ (Available Expressions)  
 $(D, \sqsubseteq) = (2^{AExp_c}, \supseteq)$  is a complete lattice with  $\perp = AExp_c$

# Duality in Complete Lattices

- **Dual** concept of least upper bound: greatest lower bound
- **Definitions:**
  - An element  $d \in D$  is called a **lower bound** of  $S \subseteq D$  if  $d \sqsubseteq s$  for every  $s \in S$  (notation:  $d \sqsubseteq S$ ).
  - A lower bound  $d$  is called **greatest lower bound (GLB)** or **infimum** of  $S$  if  $d' \sqsubseteq d$  for every lower bound  $d'$  of  $S$  (notation:  $d = \bigsqcap S$ ).

# Duality in Complete Lattices

- **Dual** concept of least upper bound: greatest lower bound
- **Definitions:**
  - An element  $d \in D$  is called a **lower bound** of  $S \subseteq D$  if  $d \sqsubseteq s$  for every  $s \in S$  (notation:  $d \sqsubseteq S$ ).
  - A lower bound  $d$  is called **greatest lower bound (GLB)** or **infimum** of  $S$  if  $d' \sqsubseteq d$  for every lower bound  $d'$  of  $S$  (notation:  $d = \sqcap S$ ).
- **Examples:**
  - (Live Variables)  $(D, \sqsubseteq) = (2^{Var_c}, \subseteq)$ ,  $\sqcap \{V_1, \dots, V_n\} = \bigcap \{V_1, \dots, V_n\}$
  - (Available Expressions)  $(D, \sqsubseteq) = (2^{AExp_c}, \supseteq)$ ,  
 $\sqcap \{A_1, \dots, A_n\} = \bigcup \{A_1, \dots, A_n\}$



# Duality in Complete Lattices

- **Dual** concept of least upper bound: greatest lower bound
- **Definitions:**
  - An element  $d \in D$  is called a **lower bound** of  $S \subseteq D$  if  $d \sqsubseteq s$  for every  $s \in S$  (notation:  $d \sqsubseteq S$ ).
  - A lower bound  $d$  is called **greatest lower bound (GLB)** or **infimum** of  $S$  if  $d' \sqsubseteq d$  for every lower bound  $d'$  of  $S$  (notation:  $d = \bigwedge S$ ).
- **Examples:**
  - (Live Variables)  $(D, \sqsubseteq) = (2^{Var_c}, \subseteq)$ ,  $\bigwedge \{V_1, \dots, V_n\} = \bigcap \{V_1, \dots, V_n\}$
  - (Available Expressions)  $(D, \sqsubseteq) = (2^{AExp_c}, \supseteq)$ ,  
 $\bigwedge \{A_1, \dots, A_n\} = \bigcup \{A_1, \dots, A_n\}$
- **Lemma:** the following are equivalent:
  - $(D, \sqsubseteq)$  is a complete lattice  
(i.e., every subset of  $D$  has a least upper bound)
  - Every subset of  $D$  has a greatest lower bound

# Duality in Complete Lattices

- **Dual** concept of least upper bound: greatest lower bound
- **Definitions:**
  - An element  $d \in D$  is called a **lower bound** of  $S \subseteq D$  if  $d \sqsubseteq s$  for every  $s \in S$  (notation:  $d \sqsubseteq S$ ).
  - A lower bound  $d$  is called **greatest lower bound (GLB)** or **infimum** of  $S$  if  $d' \sqsubseteq d$  for every lower bound  $d'$  of  $S$  (notation:  $d = \bigwedge S$ ).
- **Examples:**
  - (Live Variables)  $(D, \sqsubseteq) = (2^{Var_c}, \subseteq)$ ,  $\bigwedge \{V_1, \dots, V_n\} = \bigcap \{V_1, \dots, V_n\}$
  - (Available Expressions)  $(D, \sqsubseteq) = (2^{AExp_c}, \supseteq)$ ,  
 $\bigwedge \{A_1, \dots, A_n\} = \bigcup \{A_1, \dots, A_n\}$
- **Lemma:** the following are equivalent:
  - $(D, \sqsubseteq)$  is a complete lattice  
(i.e., every subset of  $D$  has a least upper bound)
  - Every subset of  $D$  has a greatest lower bound
- **Corollary:** every complete lattice has a greatest element  $\top := \bigwedge \emptyset$

Chains are generated by the approximation of the analysis information in the fixpoint iteration.

## Definition 3.10 (Chain)

Let  $(D, \sqsubseteq)$  be a partial order. A subset  $S \subseteq D$  is called an **(ascending) chain** in  $D$  if, for every  $s_1, s_2 \in S$ ,

$$s_1 \sqsubseteq s_2 \text{ or } s_2 \sqsubseteq s_1$$

(that is,  $S$  is a totally ordered subset of  $D$ ).

Chains are generated by the approximation of the analysis information in the fixpoint iteration.

## Definition 3.10 (Chain)

Let  $(D, \sqsubseteq)$  be a partial order. A subset  $S \subseteq D$  is called an **(ascending) chain** in  $D$  if, for every  $s_1, s_2 \in S$ ,

$$s_1 \sqsubseteq s_2 \text{ or } s_2 \sqsubseteq s_1$$

(that is,  $S$  is a totally ordered subset of  $D$ ).

## Example 3.11

- 1 Every  $S \subseteq \mathbb{N}$  is a chain in  $(\mathbb{N}, \leq)$

Chains are generated by the approximation of the analysis information in the fixpoint iteration.

## Definition 3.10 (Chain)

Let  $(D, \sqsubseteq)$  be a partial order. A subset  $S \subseteq D$  is called an **(ascending) chain** in  $D$  if, for every  $s_1, s_2 \in S$ ,

$$s_1 \sqsubseteq s_2 \text{ or } s_2 \sqsubseteq s_1$$

(that is,  $S$  is a totally ordered subset of  $D$ ).

## Example 3.11

- 1 Every  $S \subseteq \mathbb{N}$  is a chain in  $(\mathbb{N}, \leq)$
- 2  $\{\emptyset, \{0\}, \{0, 1\}, \{0, 1, 2\}, \dots\}$  is a chain in  $(2^{\mathbb{N}}, \subseteq)$

Chains are generated by the approximation of the analysis information in the fixpoint iteration.

## Definition 3.10 (Chain)

Let  $(D, \sqsubseteq)$  be a partial order. A subset  $S \subseteq D$  is called an **(ascending) chain** in  $D$  if, for every  $s_1, s_2 \in S$ ,

$$s_1 \sqsubseteq s_2 \text{ or } s_2 \sqsubseteq s_1$$

(that is,  $S$  is a totally ordered subset of  $D$ ).

## Example 3.11

- 1 Every  $S \subseteq \mathbb{N}$  is a chain in  $(\mathbb{N}, \leq)$
- 2  $\{\emptyset, \{0\}, \{0, 1\}, \{0, 1, 2\}, \dots\}$  is a chain in  $(2^{\mathbb{N}}, \subseteq)$
- 3  $\{\emptyset, \{0\}, \{1\}\}$  is not a chain in  $(2^{\mathbb{N}}, \subseteq)$

# The Ascending Chain Condition

Termination of fixpoint iteration is guaranteed by the Ascending Chain Condition.

## Definition 3.12 (Ascending Chain Condition)

A partial order  $(D, \sqsubseteq)$  satisfies the **Ascending Chain Condition (ACC)** if each ascending chain  $d_1 \sqsubseteq d_2 \sqsubseteq \dots$  eventually stabilizes, i.e., there exists  $n \in \mathbb{N}$  such that  $d_n = d_{n+1} = \dots$

# The Ascending Chain Condition

Termination of fixpoint iteration is guaranteed by the Ascending Chain Condition.

## Definition 3.12 (Ascending Chain Condition)

A partial order  $(D, \sqsubseteq)$  satisfies the **Ascending Chain Condition (ACC)** if each ascending chain  $d_1 \sqsubseteq d_2 \sqsubseteq \dots$  eventually stabilizes, i.e., there exists  $n \in \mathbb{N}$  such that  $d_n = d_{n+1} = \dots$

## Example 3.13

①  $(\mathbb{N}, \leq)$  does not satisfy ACC



# The Ascending Chain Condition

Termination of fixpoint iteration is guaranteed by the Ascending Chain Condition.

## Definition 3.12 (Ascending Chain Condition)

A partial order  $(D, \sqsubseteq)$  satisfies the **Ascending Chain Condition (ACC)** if each ascending chain  $d_1 \sqsubseteq d_2 \sqsubseteq \dots$  eventually stabilizes, i.e., there exists  $n \in \mathbb{N}$  such that  $d_n = d_{n+1} = \dots$

## Example 3.13

- ①  $(\mathbb{N}, \leq)$  does not satisfy ACC
- ② (Live Variables)  $(D, \sqsubseteq) = (2^{Var_c}, \sqsubseteq)$  satisfies ACC since  $Var_c$  (unlike  $Var$ ) is finite

# The Ascending Chain Condition

Termination of fixpoint iteration is guaranteed by the Ascending Chain Condition.

## Definition 3.12 (Ascending Chain Condition)

A partial order  $(D, \sqsubseteq)$  satisfies the **Ascending Chain Condition (ACC)** if each ascending chain  $d_1 \sqsubseteq d_2 \sqsubseteq \dots$  eventually stabilizes, i.e., there exists  $n \in \mathbb{N}$  such that  $d_n = d_{n+1} = \dots$

## Example 3.13

- ①  $(\mathbb{N}, \leq)$  does not satisfy ACC
- ② (Live Variables)  $(D, \sqsubseteq) = (2^{Var_c}, \subseteq)$  satisfies ACC since  $Var_c$  (unlike  $Var$ ) is finite
- ③ (Available Expressions)  $(D, \sqsubseteq) = (2^{AExp_c}, \supseteq)$  satisfies ACC since  $AExp_c$  (unlike  $AExp$ ) is finite