



Static Program Analysis

Lecture 2: Dataflow Analysis I

(Introduction & Available Expressions/Live Variables Analysis)

Summer Semester 2018

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-18/spa/>

Exercises

- Assignments made available via [L2P Learning Room](#)
- Requires registration via [Campus web page](#)
- Please work on assignments in [groups of three](#)

Preliminaries on Dataflow Analysis

Dataflow Analysis: the Approach

- Traditional form of **program analysis**
- Idea: describe how analysis information **flows** through program
- Distinctions:
 - dependence on statement order:
 - flow-sensitive** vs. **flow-insensitive** analyses
 - direction of flow:
 - forward** vs. **backward** analyses
 - quantification over paths:
 - may (union)** vs. **must (intersection)** analyses
 - procedures:
 - interprocedural** vs. **intraprocedural** analyses

Preliminaries on Dataflow Analysis

Labelled Programs

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

Definition 2.1 (Labelled WHILE programs)

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

$$\begin{aligned} a &::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp \\ b &::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\ c &::= [\text{skip}]' \mid [x := a]' \mid c_1 ; c_2 \mid \\ &\quad \text{if } [b]' \text{ then } c_1 \text{ else } c_2 \text{ end} \mid \text{while } [b]' \text{ do } c \text{ end} \in Cmd \end{aligned}$$

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

Preliminaries on Dataflow Analysis

A WHILE Program **with Labels**

Example 2.2

```
x := 6;
y := 7;
z := 0;
while x > 0 do
  x := x - 1;
  v := y;
  while v > 0 do
    v := v - 1;
    z := z + 1
  end
end
end
```

Preliminaries on Dataflow Analysis

Representing Control Flow I

Every (labelled) statement has a single entry (given by the initial label) and generally multiple exits (given by the final labels):

Definition 2.3 (Initial and final labels)

The mappings

$$\text{init} : \text{Cmd} \rightarrow \text{Lab} \quad \text{and} \quad \text{final} : \text{Cmd} \rightarrow 2^{\text{Lab}}$$

respectively return the **initial** and **final** label(s) of a statement:

$c \in \text{Cmd}$	$\text{init}(c)$	$\text{final}(c)$
$[\text{skip}]'$	$/$	$\{ / \}$
$[x := a]'$	$/$	$\{ / \}$
$c_1 ; c_2$	$\text{init}(c_1)$	$\text{final}(c_2)$
$\text{if } [b]'$ then c_1 else c_2 end	$/$	$\text{final}(c_1) \cup \text{final}(c_2)$
$\text{while } [b]'$ do c end	$/$	$\{ / \}$

Representing Control Flow II

Definition 2.4 (Flow relation)

Given a statement $c \in \text{Cmd}$, the (control) flow relation

$$\text{flow}(c) \subseteq \text{Lab} \times \text{Lab}$$

is defined by

$$\text{flow}([\text{skip}]') := \emptyset$$

$$\text{flow}([x := a]') := \emptyset$$

$$\text{flow}(c_1 ; c_2) := \text{flow}(c_1) \cup \text{flow}(c_2) \cup \text{final}(c_1) \times \{\text{init}(c_2)\}$$

$$\text{flow}(\text{if } [b]' \text{ then } c_1 \text{ else } c_2 \text{ end}) := \text{flow}(c_1) \cup \text{flow}(c_2) \cup \{(l, \text{init}(c_1)), (l, \text{init}(c_2))\}$$

$$\text{flow}(\text{while } [b]' \text{ do } c \text{ end}) := \text{flow}(c) \cup \{(l, \text{init}(c))\} \cup \text{final}(c) \times \{l\}$$

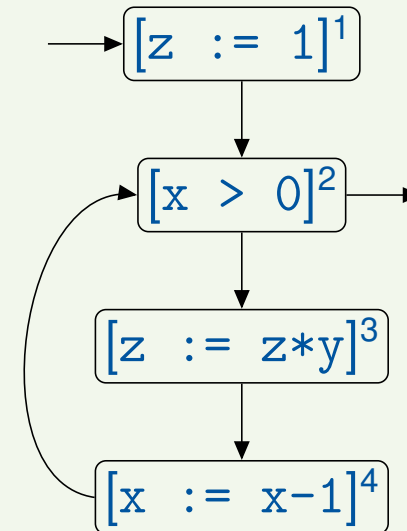
Representing Control Flow III

Example 2.5

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

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

Visualisation by (control) flow graph:



Representing Control Flow IV

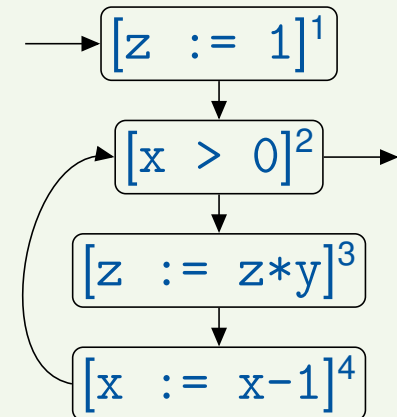
- To simplify the presentation we will often assume that the program $c \in \text{Cmd}$ under consideration has an **isolated entry**, meaning that

$$\{l \in \text{Lab} \mid (l, \text{init}(c)) \in \text{flow}(c)\} = \emptyset$$

(which is the case when c does not start with a **while** loop)

- Similarly: $c \in \text{Cmd}$ has **isolated exits** if $\{l' \in \text{Lab} \mid (l, l') \in \text{flow}(c) \text{ for some } l \in \text{final}(c)\} = \emptyset$ (which is the case when no final label identifies a loop header)

Example 2.6 (cf. Ex. 2.5)



has an isolated entry
but not isolated exits

An Example: Available Expressions Analysis

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 for **Common Subexpression Elimination**:
replace subexpression by variable that contains up-to-date value
- Only interesting for non-trivial (i.e., complex) arithmetic expressions

Example 2.7 (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  
end
```

- a+b available at label 3
- a+b not available at label 5
- possible optimisation:
while [y > x]³ do

An Example: Available Expressions Analysis

Formalising Available Expressions Analysis I

- Given $a \in AExp$, $b \in BExp$, $c \in Cmd$,
 - $Var_a/Var_b/Var_c$ denotes the set of all **variables** occurring in $a/b/c$
 - $CExp_b/CExp_c$ denote the sets of all **complex arithmetic expressions** occurring in b/c
- An expression a is **killed** in a block B if any of the variables in a is modified in B
- Formally: $kill_{AE} : Blk_c \rightarrow 2^{CExp_c}$ is defined by

$$\begin{aligned}kill_{AE}([skip]') &:= \emptyset \\kill_{AE}([x := a]') &:= \{a' \in CExp_c \mid x \in Var_{a'}\} \\kill_{AE}([b]') &:= \emptyset\end{aligned}$$

- An expression a is **generated** in a block B if it is evaluated in and none of its variables are modified by B
- Formally: $gen_{AE} : Blk_c \rightarrow 2^{CExp_c}$ is defined by

$$\begin{aligned}gen_{AE}([skip]') &:= \emptyset \\gen_{AE}([x := a]') &:= \{a \mid x \notin Var_a\} \\gen_{AE}([b]') &:= CExp_b\end{aligned}$$

An Example: Available Expressions Analysis

Formalising Available Expressions Analysis II

Example 2.8 (kill_{AE} / gen_{AE} functions)

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

- $CExp_c = \{a+b, a*b, a+1\}$

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

An Example: Available Expressions Analysis

The Equation System I

- Analysis itself defined by setting up an **equation system**
- For each $l \in Lab_c$, $AE_l \subseteq CExp_c$ represents the **set of available expressions at the entry of block B^l**
- Formally, for $c \in Cmd$ with isolated entry:

$$AE_l = \begin{cases} \emptyset & \text{if } l = \text{init}(c) \\ \bigcap \{ \varphi_{l'}(AE_{l'}) \mid (l', l) \in \text{flow}(c) \} & \text{otherwise} \end{cases}$$

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

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

- Characterisation of analysis:
 - flow-sensitive**: results depending on order of assignments
 - forward**: starts in $\text{init}(c)$ and proceeds downwards
 - must**: \bigcap in equations for AE_l
- Later: solution **not necessarily unique**
 - \implies choose **greatest one**

An Example: Available Expressions Analysis

The Equation System II

Reminder:

$$AE_I = \begin{cases} \emptyset & \text{if } I = \text{init}(c) \\ \bigcap \{ \varphi_{I'}(AE_{I'}) \mid (I', I) \in \text{flow}(c) \} & \text{otherwise} \end{cases}$$

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

Example 2.9 (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
    end
  
```

Equations:

$$AE_1 = \emptyset$$

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

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

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

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

(Unique) solution:

$$AE_1 = \emptyset$$

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

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

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

$$AE_5 = \emptyset$$

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

Another Example: Live Variables Analysis

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.

- 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

Another Example: Live Variables Analysis

An Example

Example 2.10 (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  
end;  
[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 optimisation: remove [x := 2]¹

Another Example: Live Variables Analysis

Formalising Live Variables Analysis I

- 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

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

Another Example: Live Variables Analysis

Formalising Live Variables Analysis II

Example 2.11 ($\text{kill}_{LV}/\text{gen}_{LV}$ functions)

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

- $\text{Var}_c = \{x, y, z\}$
- $l \in \text{Lab}_c$ $\text{kill}_{LV}(B^l)$ $\text{gen}_{LV}(B^l)$

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

Another Example: Live Variables Analysis

The Equation System I

- For each $I \in Lab_c$, $LV_I \subseteq Var_c$ 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

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

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

Another Example: Live Variables Analysis

The Equation System II

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

Example 2.12 (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
    end;
    [x := z]7

```

$l \in Lab_c$	$\text{kill}_{LV}(B^l)$	$\text{gen}_{LV}(B^l)$
1	{x}	∅
2	{y}	∅
3	{x}	∅
4	∅	{y}
5	{z}	{x}
6	{z}	{y}
7	{x}	{z}

Equations:

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

(Unique) solution:

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

Heading for a Dataflow Analysis Framework

Similarities Between Analysis Problems

- **Observation:** the analyses presented so far have some **similarities**
- ⇒ Look for underlying **framework**
- **Advantages:**
 - possibility for designing (efficient) **generic algorithms** for solving dataflow equations
 - enables generic **correctness proofs** of analyses and algorithms
- **Overall pattern:** for $c \in \text{Cmd}$ and $l \in \text{Lab}_c$, the **analysis information (AI)** is described by **equations** of the form

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

where

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

Heading for a Dataflow Analysis Framework

Characterisation of Analyses

Direction of information flow

- **Forward:**
 - $F = \text{flow}(c)$
 - AI_l refers to entry of B^l
 - c has isolated entry
- **Backward:**
 - $F = \text{flow}^R(c)$
 - AI_l refers to exit of B^l
 - c has isolated exits

Quantification over paths

- **May:**
 - $\sqcup = \cup$
 - property satisfied by some path
 - interested in least solution (later)
- **Must:**
 - $\sqcap = \cap$
 - property satisfied by all paths
 - interested in greatest solution (later)

Heading for a Dataflow Analysis Framework

Solutions as Fixpoints

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

$$(A_l = \tau_l)_{l \in Lab_c} \iff \Phi((A_l)_{l \in Lab_c}) = (A_l)_{l \in Lab_c}$$

where $\Phi((A_l)_{l \in Lab_c}) := (\tau_l)_{l \in Lab_c}$

- **Approach:** approximate fixpoint by **iteration**

Heading for a Dataflow Analysis Framework

Roadmap

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

1. Characterise 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. Optimise fixpoint iteration by **worklist algorithm**