

Semantics and Verification of Software

Lecture 15: Dataflow Analysis

Thomas Noll

Lehrstuhl für Informatik 2
RWTH Aachen University
noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/svsw/>

Summer semester 2007

- 1 Repetition: Dataflow Analysis
- 2 Available Expressions Analysis (continued)
- 3 Live Variables Analysis
- 4 A Dataflow Analysis Framework

Labelled Programs

- Goal: **localization** of analysis information
- Dataflow information will be associated with
 - assignments
 - tests in conditionals (**if**) and loops (**while**)
 - **skip** statements

These constructs will be called **blocks**.

- Assume set of **labels** Lab with meta variable $l \in Lab$
(usually $Lab = \mathbb{N}$)

Definition (Labelled WHILE programs)

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

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

Here all labels in a statement $c \in Cmd$ are assumed to be distinct.

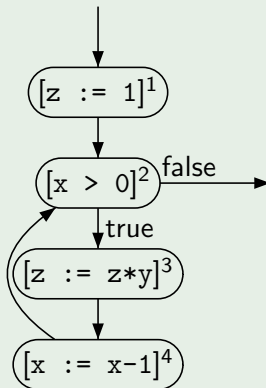
Representing Control Flow

Example

Visualization by **flow graph**:

```
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)}
```



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
- for simplicity: only non-trivial 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**]³ do

Formalizing Available Expressions Analysis I

- Given $c \in Cmd$, $Lab_c/Block_c/AExp_c$ denote the sets of all labels/blocks/complex arithmetic expressions occurring in c , respectively
- An expression a is **killed** in a block B if any of the variables in a is modified in B
- Formally: $kill_{AE} : Block_c \rightarrow 2^{AExp_c}$ is defined by
$$\begin{aligned}kill_{AE}([skip]^l) &:= \emptyset \\kill_{AE}([x := a]^l) &:= \{a' \in AExp_c \mid x \in FV(a')\} \\kill_{AE}([b]^l) &:= \emptyset\end{aligned}$$
- An expression a is **generated** in a block B if it is evaluated in and none of its variables are modified by B
- Formally: $gen_{AE} : Block_c \rightarrow 2^{AExp_c}$ is defined by
$$\begin{aligned}gen_{AE}([skip]^l) &:= \emptyset \\gen_{AE}([x := a]^l) &:= \{a \mid x \notin FV(a)\} \\gen_{AE}([b]^l) &:= AExp_b\end{aligned}$$

Example (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
```

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

$l \in Lab_c$	kill _{AE} (B^l)	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	$\emptyset$	$\{a+b\}$

- 1 Repetition: Dataflow Analysis
- 2 Available Expressions Analysis (continued)
- 3 Live Variables Analysis
- 4 A Dataflow Analysis Framework

The Equation System I

- Analysis itself defined by setting up an **equation system**
- For each $l \in Lab_c$, $AE_l \subseteq AExp_c$ 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 $\varphi_{l'} : 2^{AExp_c} \rightarrow 2^{AExp_c}$ denotes the **transfer function** of block $B^{l'}$, given by

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

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

The Equation System II

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 15.1 (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
```

Equations:

$$AE_1 = \emptyset$$

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

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

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

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

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

Solution: $AE_1 = \emptyset$
 $AE_2 = \{a+b\}$
 $AE_3 = \{a+b\}$
 $AE_4 = \{a+b\}$
 $AE_5 = \emptyset$

- 1 Repetition: Dataflow Analysis
- 2 Available Expressions Analysis (continued)
- 3 Live Variables Analysis
- 4 A Dataflow Analysis Framework

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
- No variable considered to be live at the **end** of the program (choice depends on application, e.g., output variables live)
- Can be used for **Dead Code Elimination**:
remove assignments to non-live variables

Example 15.2 (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]¹

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

$$\text{kill}_{\text{LV}}([\text{skip}]^l) := \emptyset$$

$$\text{kill}_{\text{LV}}([x := a]^l) := \{x\}$$

$$\text{kill}_{\text{LV}}([b]^l) := \emptyset$$

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

$$\text{gen}_{\text{AE}}([\text{skip}]^l) := \emptyset$$

$$\text{gen}_{\text{AE}}([x := a]^l) := \text{FV}(a)$$

$$\text{gen}_{\text{AE}}([b]^l) := \text{FV}(b)$$

Example 15.3 (kill_{LV} / 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;  
    [x := z]7
```

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

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

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

The Equation System I

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

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

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

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

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

The Equation System II

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

Example 15.4 (LV equation system)

$c = [x := 2]^1; [y := 4]^2;$

$[x := 1]^3;$

if $[y > 0]^4$ then

$[z := x]^5$

else

$[z := y*y]^6;$

$[x := z]^7$

$$LV_1 = \varphi_2(LV_2) = LV_2 \setminus \{y\}$$

$$LV_2 = \varphi_3(LV_3) = LV_3 \setminus \{x\}$$

$$LV_3 = \varphi_4(LV_4) = LV_4 \cup \{y\}$$

$$LV_4 = \varphi_5(LV_5) \cup \varphi_6(LV_6)$$

$$= ((LV_5 \setminus \{z\}) \cup \{x\}) \cup ((LV_6 \setminus \{z\}) \cup \{y\})$$

$$LV_5 = \varphi_7(LV_7) = (LV_7 \setminus \{x\}) \cup \{z\}$$

$$LV_6 = \varphi_7(LV_7) = (LV_7 \setminus \{x\}) \cup \{z\}$$

$$LV_7 = \emptyset$$

$l \in Lab_c$ $\text{kill}_{LV}(B^l)$ $\text{gen}_{LV}(B^l)$

1	$\{x\}$	$\emptyset$
2	$\{y\}$	$\emptyset$
3	$\{x\}$	$\emptyset$
4	$\emptyset$	$\{y\}$
5	$\{z\}$	$\{x\}$
6	$\{z\}$	$\{y\}$
7	$\{x\}$	$\{z\}$

Solution: $LV_1 = \emptyset$

$$LV_2 = \{y\}$$

$$LV_3 = \{x, y\}$$

$$LV_4 = \{x, y\}$$

$$LV_5 = \{z\}$$

$$LV_6 = \{z\}$$

$$LV_7 = \emptyset$$

- 1 Repetition: Dataflow Analysis
- 2 Available Expressions Analysis (continued)
- 3 Live Variables Analysis
- 4 A Dataflow Analysis 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**
- **Overall pattern:** for $c \in Cmd$ and $l \in Lab_c$, 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

- ι specifies the initial analysis information
- E is $\{\text{init}(c)\}$ or $\{\text{final}(c)\}$
- $\bigsqcup$ is $\bigcap$ or $\bigcup$
- $\varphi_{l'}$ denotes the transfer function of block $B^{l'}$
- F is $\text{flow}(c)$ or $\text{flow}^R(c)$ ($:= \{(l', l) \mid (l, l') \in \text{flow}(c)\}$)

- **Direction of information flow:**

- **forward:**

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

- **backward:**

- $F = \text{flow}^R(c)$
 - Al_l concerns exit of B^l
 - 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)