Static Program Analysis

Lecture 2: Dataflow Analysis I (Introduction & Available Expressions 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

Outline

Dataflow Analysis: the Approach

- Traditional form of program analysis

Dataflow Analysis: the Approach

- Traditional form of program analysis
- Idea: describe how analysis information flows through program

Dataflow Analysis: the Approach

- Traditional form of program analysis
- Idea: describe how analysis information flows through program
- Distinctions:

direction of flow:
: forward vs. backward analyses

procedures:
: interprocedural vs. intraprocedural analyses

quantification over paths:
: may (union) vs. must (intersection) analyses

dependence on statement order:
: flow-sensitive vs. flow-insensitive analyses

Labeled Programs

- Goal: localization of analysis information

Labeled Programs

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

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 $L = \mathbb{N}$)

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 2.1 (Labeled WHILE programs)

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

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

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

A WHILE Program

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;
```

A WHILE Program with Labels

Example 2.2

```
[x := 6]^1;
[y := 7]^2;
[z := 0]^3;
while [x > 0]^4 do
  [x := x - 1]^5;
  [v := y]^6;
  while [v > 0]^7 do
    [v := v - 1]^8;
    [z := z + 1]^9
```

Representing Control Flow I

Every (labeled) 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 mapping init : $Cmd \to L$ returns the initial label of a statement:

$$
\begin{aligned}
\text{init}([\texttt{skip}]^l) &:= l \\
\text{init}([x \texttt{ := } a]^l) &:= l \\
\text{init}(c_1 \texttt{;} c_2) &:= \text{init}(c_1) \\
\text{init}(\texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2) &:= l \\
\text{init}(\texttt{while } [b]^l \texttt{ do } c) &:= l
\end{aligned}
$$

Representing Control Flow I

Every (labeled) 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 mapping init : $Cmd \to L$ returns the initial label of a statement:

$$\begin{aligned}
\text{init}([\texttt{skip}]^l) &:= l\\
\text{init}([x \texttt{ := } a]^l) &:= l\\
\text{init}(c_1 \texttt{;} c_2) &:= \text{init}(c_1)\\
\text{init}(\texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2) &:= l\\
\text{init}(\texttt{while } [b]^l \texttt{ do } c) &:= l
\end{aligned}$$

The mapping final : $Cmd \to 2^L$ returns the set of final labels of a statement:

$$\begin{aligned}
\text{final}([\texttt{skip}]^l) &:= \{l\}\\
\text{final}([x \texttt{ := } a]^l) &:= \{l\}\\
\text{final}(c_1 \texttt{;} c_2) &:= \text{final}(c_2)\\
\text{final}(\texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2) &:= \text{final}(c_1) \cup \text{final}(c_2)\\
\text{final}(\texttt{while } [b]^l \texttt{ do } c) &:= \{l\}
\end{aligned}$$

Representing Control Flow II

Definition 2.4 (Flow relation)

Given a statement $c \in Cmd$, the (control) flow relation $\mathrm{flow}(c) \subseteq L \times L$ is defined by

$$
\begin{aligned}
\mathrm{flow}([\texttt{skip}]^l) &:= \emptyset \\
\mathrm{flow}([x \texttt{ := } a]^l) &:= \emptyset \\
\mathrm{flow}(c_1 \texttt{;} c_2) &:= \mathrm{flow}(c_1) \cup \mathrm{flow}(c_2) \cup \\
&\quad \{(l, \mathrm{init}(c_2)) \mid l \in \mathrm{final}(c_1)\} \\
\mathrm{flow}(\texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2) &:= \mathrm{flow}(c_1) \cup \mathrm{flow}(c_2) \cup \\
&\quad \{(l, \mathrm{init}(c_1)), (l, \mathrm{init}(c_2))\} \\
\mathrm{flow}(\texttt{while } [b]^l \texttt{ do } c) &:= \mathrm{flow}(c) \cup \{(l, \mathrm{init}(c))\} \cup \\
&\quad \{(l', l) \mid l' \in \mathrm{final}(c)\}
\end{aligned}
$$

Example 2.5

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

Representing Control Flow III

Example 2.5

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

$$\begin{aligned} \text{init}(c) &= 1 \\ \text{final}(c) &= \{2\} \\ \text{flow}(c) &= \{(1,2),(2,3),(3,4),(4,2)\} \end{aligned}$$

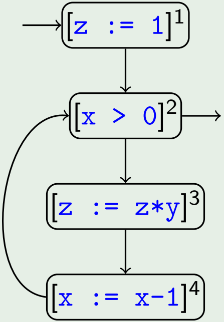
Representing Control Flow III

Example 2.5

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

$$\begin{aligned} \text{init}(c) &= 1 \\ \text{final}(c) &= \{2\} \\ \text{flow}(c) &= \{(1,2),(2,3),(3,4),(4,2)\} \end{aligned}$$

Visualization by (control) flow graph:



Representing Control Flow IV

- To simplify the presentation we will often assume that the program $c \in Cmd$ under consideration has an isolated entry, meaning that
$$\{l \in L \mid (l, \text{init}(c)) \in \text{flow}(c)\} = \emptyset$$
(which is the case when c does not start with a `while` loop)

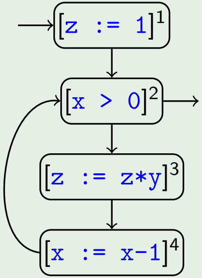
Representing Control Flow IV

- To simplify the presentation we will often assume that the program $c \in \mathit{Cmd}$ under consideration has an isolated entry, meaning that
$$\{l \in L \mid (l, \text{init}(c)) \in \text{flow}(c)\} = \emptyset$$
(which is the case when c does not start with a `while` loop)
- Similarly: $c \in \mathit{Cmd}$ has isolated exits if
$$\{l' \in L \mid (l, l') \in \text{flow}(c) \text{ for some } l \in \text{final}(c)\} = \emptyset$$
(which is the case when no final label identifies a loop header)

Representing Control Flow IV

- To simplify the presentation we will often assume that the program $c \in Cmd$ under consideration has an isolated entry, meaning that
$$\{l \in L \mid (l, \mathrm{init}(c)) \in \mathrm{flow}(c)\} = \emptyset$$
(which is the case when c does not start with a `while` loop)
- Similarly: $c \in Cmd$ has isolated exits if
$$\{l' \in L \mid (l, l') \in \mathrm{flow}(c) \text{ for some } l \in \mathrm{final}(c)\} = \emptyset$$
(which is the case when no final label identifies a loop header)

Example 2.6



has an isolated entry but not isolated exits

Outline

1. Preliminaries on Dataflow Analysis
2. 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.

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

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
```

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
```

- 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 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
```

- 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 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
```

- 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 $a \in AExp$, $b \in BExp$, $c \in Cmd$
  - $Var_a/Var_b/Var_c$ denotes the set of all variables occurring in $a/b/c$
  - $AExp_b/AExp_c$ denote the sets of all complex arithmetic expressions occurring in b/c

Formalizing 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 $a/b/c$
  - $AExp_b/AExp_c$ 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

Formalizing 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 $a/b/c$
  - $AExp_b / AExp_c$ 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: $\mathsf{kill}_{\mathsf{AE}} : Blk_c \to 2^{AExp_c}$ is defined by

$$
\begin{aligned}
\mathsf{kill}_{\mathsf{AE}}([\mathtt{skip}]^l) &:= \emptyset \\
\mathsf{kill}_{\mathsf{AE}}([x \mathrel{\mathtt{:=}} a]^l) &:= \{a' \in AExp_c \mid x \in Var_{a'}\} \\
\mathsf{kill}_{\mathsf{AE}}([b]^l) &:= \emptyset
\end{aligned}
$$

Formalizing 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 $a/b/c$
  - $AExp_b/AExp_c$ 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: $\text{kill}_{\text{AE}} : Blk_c \to 2^{AExp_c}$ is defined by

$$\begin{aligned} \text{kill}_{\text{AE}}([\texttt{skip}]^l) &:= \emptyset \\ \text{kill}_{\text{AE}}([x \texttt{ := } a]^l) &:= \{a' \in AExp_c \mid x \in Var_{a'}\} \\ \text{kill}_{\text{AE}}([b]^l) &:= \emptyset \end{aligned}$$

- An expression a is generated in a block B if it is evaluated in and none of its variables are modified by B

Formalizing 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 $a/b/c$
  - $AExp_b/AExp_c$ 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: $\mathrm{kill}_{\mathrm{AE}} : Blk_c \to 2^{AExp_c}$ is defined by

$$\begin{aligned} \mathrm{kill}_{\mathrm{AE}}([\texttt{skip}]^l) &:= \emptyset \\ \mathrm{kill}_{\mathrm{AE}}([x \;\texttt{:=}\; a]^l) &:= \{a' \in AExp_c \mid x \in Var_{a'}\} \\ \mathrm{kill}_{\mathrm{AE}}([b]^l) &:= \emptyset \end{aligned}$$

- An expression a is generated in a block B if it is evaluated in and none of its variables are modified by B
- Formally: $\mathrm{gen}_{\mathrm{AE}} : Blk_c \to 2^{AExp_c}$ is defined by

$$\begin{aligned} \mathrm{gen}_{\mathrm{AE}}([\texttt{skip}]^l) &:= \emptyset \\ \mathrm{gen}_{\mathrm{AE}}([x \;\texttt{:=}\; a]^l) &:= \{a \mid x \notin Var_a\} \\ \mathrm{gen}_{\mathrm{AE}}([b]^l) &:= AExp_b \end{aligned}$$

Formalizing Available Expressions Analysis II

Example 2.8 ($\text{kill}_{\text{AE}}/\text{gen}_{\text{AE}}$ functions)

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

Formalizing Available Expressions Analysis II

Example 2.8 ($\mathrm{kill}_{\mathrm{AE}}/\mathrm{gen}_{\mathrm{AE}}$ 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` $\}$

Formalizing Available Expressions Analysis II

Example 2.8 ($\text{kill}_{AE}/\text{gen}_{AE}$ 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_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}

The Equation System I

- Analysis itself defined by setting up an equation system

The Equation System I

- Analysis itself defined by setting up an equation system
- For each $l \in L_c$, $\mathrm{AE}_l \subseteq AExp_c$ represents the set of available expressions at the entry of block B^l

The Equation System I

- Analysis itself defined by setting up an equation system
- For each $l \in L_c$, $\mathrm{AE}_l \subseteq AExp_c$ represents the set of available expressions at the entry of block B^l
- Formally, for $c \in Cmd$ with isolated entry:

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

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

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

The Equation System I

- Analysis itself defined by setting up an equation system
- For each $l \in L_c$, $\mathrm{AE}_l \subseteq AExp_c$ represents the set of available expressions at the entry of block B^l
- Formally, for $c \in Cmd$ with isolated entry:

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

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

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

- Characterization of analysis:

forward: starts in $\mathrm{init}(c)$ and proceeds downwards

must: $\bigcap$ in equation for AE_l

flow-sensitive: results depending on order of assignments

The Equation System I

- Analysis itself defined by setting up an equation system
- For each $l \in L_c$, $\mathrm{AE}_l \subseteq AExp_c$ represents the set of available expressions at the entry of block B^l
- Formally, for $c \in Cmd$ with isolated entry:

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

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

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

- Characterization of analysis:

  forward: starts in $\mathrm{init}(c)$ and proceeds downwards

  must: $\bigcap$ in equation for AE_l

  flow-sensitive: results depending on order of assignments

- Later: solution not necessarily unique

  $\Longrightarrow$ choose greatest one

The Equation System II

Reminder:

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

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

The Equation System II

Reminder:

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

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

Example 2.9 (AE equation system)

$c = [\texttt{x := a+b}]^1;$
$\quad [\texttt{y := a*b}]^2;$
$\quad \texttt{while } [\texttt{y > a+b}]^3 \texttt{ do}$
$\qquad [\texttt{a := a+1}]^4;$
$\qquad [\texttt{x := a+b}]^5$

The Equation System II

Reminder:

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

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

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
```

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

The Equation System II

Reminder:

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

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

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

$l \in L_c$	$\mathsf{kill}_{\mathsf{AE}}(B^l)$	$\mathsf{gen}_{\mathsf{AE}}(B^l)$
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:

$$\begin{aligned}
\mathsf{AE}_1 &= \emptyset \\
\mathsf{AE}_2 &= \varphi_1(\mathsf{AE}_1) = \mathsf{AE}_1 \cup \{\texttt{a+b}\} \\
\mathsf{AE}_3 &= \varphi_2(\mathsf{AE}_2) \cap \varphi_5(\mathsf{AE}_5) \\
&= (\mathsf{AE}_2 \cup \{\texttt{a*b}\}) \cap (\mathsf{AE}_5 \cup \{\texttt{a+b}\}) \\
\mathsf{AE}_4 &= \varphi_3(\mathsf{AE}_3) = \mathsf{AE}_3 \cup \{\texttt{a+b}\} \\
\mathsf{AE}_5 &= \varphi_4(\mathsf{AE}_4) = \mathsf{AE}_4 \setminus \{\texttt{a+b}, \texttt{a*b}, \texttt{a+1}\}
\end{aligned}$$

The Equation System II

Reminder:

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

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

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
```

$l \in L_c$	$\mathrm{kill}_{\mathrm{AE}}(B^l)$	$\mathrm{gen}_{\mathrm{AE}}(B^l)$
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:

$$\begin{aligned} \mathrm{AE}_1 &= \emptyset \\ \mathrm{AE}_2 &= \varphi_1(\mathrm{AE}_1) = \mathrm{AE}_1 \cup \{\texttt{a+b}\} \\ \mathrm{AE}_3 &= \varphi_2(\mathrm{AE}_2) \cap \varphi_5(\mathrm{AE}_5) \\ &= (\mathrm{AE}_2 \cup \{\texttt{a*b}\}) \cap (\mathrm{AE}_5 \cup \{\texttt{a+b}\}) \\ \mathrm{AE}_4 &= \varphi_3(\mathrm{AE}_3) = \mathrm{AE}_3 \cup \{\texttt{a+b}\} \\ \mathrm{AE}_5 &= \varphi_4(\mathrm{AE}_4) = \mathrm{AE}_4 \setminus \{\texttt{a+b}, \texttt{a*b}, \texttt{a+1}\} \end{aligned}$$

Solution:

$$\begin{aligned} \mathrm{AE}_1 &= \emptyset \\ \mathrm{AE}_2 &= \{\texttt{a+b}\} \\ \mathrm{AE}_3 &= \{\texttt{a+b}\} \\ \mathrm{AE}_4 &= \{\texttt{a+b}\} \\ \mathrm{AE}_5 &= \emptyset \end{aligned}$$