Compiler

Static Analysis:

Classic Problems

Classic Static Analysis Problems

- We have seen Reaching Definition (RD) Analysis
- We'll look at three more analyses
  - Available Expressions (AE)
  - Very Busy Expressions (VBE)
  - Live Variables (LV)
- We'll see what's similar and different and generalize to data flow frameworks

Reaching Definition (RD) Analysis

- RD analysis determines
  - For each program point, which assignments *may* have been made and not overwritten along *some path* to that program point.

RD Example

```
static int factorial(int n) {
  int result;
  int i;
  [StaticJavaLib.assertTrue(n >= 1);]1
  [result = 1;]2
  [i = 2;]3
  [while (i <= n) {
    [result = result * i;]5
    [i = i + 1;]6
  }]4
  [return result;]7
}
```

Clearly the definition in 2 may reach 5, we describe this more compactly by saying `(result, 2)` reaches the entry 5.

Denoting RD

- We can describe the set of reaching definitions for the entry and exit of each program point as:
  - $RD_{entry} : Lab_* \rightarrow P(Var_* \times D)$
  - $RD_{exit} : Lab_* \rightarrow P(Var_* \times D)$
- Notes
  - $D = \{ \bullet, ? \} \cup Lab_*$
  - Lab_* and Var_* are the subsets of labels and variables occurring in the program under analysis, e.g., { 1, 2, …, 7 } and { `n`, `result`, `i` }
  - dot (•) denotes definition from a formal parameter (or a field)
  - question mark (?) denotes unknown definition

RD Flow Equations

- A definition is *killed* in a block if its variables is defined in the block

$kill_{\mathrm{RD}} : \mathrm{Lab}_* \to P(\mathrm{Var}_* \times D)$

- $kill_{\mathrm{RD}}([x = e\ ;]^l) = \{\ (x, d') \mid d' \in D\ \}$
- $kill_{\mathrm{RD}}([\ldots]^l) = \emptyset$

RD Flow Equations

- A definition is *generated* in a block if the block assigns a value to a variable

  $gen_{RD} : \mathrm{Lab}_* \rightarrow P(\mathrm{Var}_* \times D)$

- $gen_{RD}([x = e\ ;]^l) = \{\ (x, l)\ \}$
- $gen_{RD}([\ldots]^l) = \emptyset$

Gen/Kill RD Example

l	$kill_{RD}(l)$	$gen_{RD}(l)$
1	$\emptyset$	$\emptyset$
2	{ (r, ?), (r, 2), (r, 5) }	{ (r, 2) }
3	{ (i,?), (i, 3), (i, 6) }	{ (i,3) }
4	$\emptyset$	$\emptyset$
5	{ (r, ?), (r, 2), (r, 5) }	{ (r,5) }
6	{ (i,?), (i, 3), (i, 6) }	{ (i,6) }
7	$\emptyset$	$\emptyset$

RD Flow Equations

- The flow equations are now expressed using the two functions

$$
RD_{entry}(l) = \begin{cases} \{ (x,\bullet) \mid x \in (\text{Param}_* \cup \text{Field}_*)\} \cup \{ (x,?) \mid x \in \text{Local}_* \}, & \text{if } l = b_{init} \\ \bigcup \{ RD_{exit}(l') \mid l' \in \mathit{preds}(l) \}, & \text{otherwise} \end{cases}
$$

$$
RD_{exit}(l) = (RD_{entry}(l) \setminus \mathit{kill}_{RD}(l)) \cup \mathit{gen}_{RD}(l)
$$

RD Flow Equations

- Data flow information is propagated in execution order
- Information at entries is calculated in terms of information at preceding exits
- Information at program entry, i.e., b_{init}, represents the fact that local variables are undefined when the program starts.
- In a language like Java where all fields (and parameters) are guaranteed to have initial values one would use dot (•)

RD Example

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{ (n,•),(r,?),(i,?) }	{ (n,•),(r,?),(i,?) }
2	{ (n,•),(r,?),(i,?) }	{ (n,•),(r,2),(i,?) }
3	{ (n,•),(r,2),(i,?) }	{ (n,•),(r,2),(i,3) }
4	{ (n,•),(r,2),(r,5),(i,3),(i,6) }	{ (n,•),(r,2),(r,5),(i,3),(i,6) }
5	{ (n,•),(r,2),(r,5),(i,3),(i,6) }	{ (n,•),(r,5),(i,3),(i,6) }
6	{ (n,•),(r,5),(i,3),(i,6) }	{ (n,•),(r,5),(i,6) }
7	{ (n,•),(r,2),(r,5),(i,3),(i,6) }	{ (n,•),(r,2),(r,5),(i,3),(i,6) }

Observation

	RD
Direction	forward
Solution	least
***b_{init}* (f) / *b_{last}* (b) Value**	$\{ (x,\bullet) \mid x \in (\text{Param}_* \cup \text{Field}_*)\} \cup \{ (x,?) \mid x \in \text{Local}_* \}$
Combining Operator	$\cup$
Description	may
Paths	some

Available Expressions (AE) Analysis

- AE analysis determines
  - For each program point, which expressions *must* have already been computed, and not later modified, on *all* paths to the program point.

AE Example

```
int x;
int y;
[x = a + b;]¹
[y = a * b;]²
[while (y > a + b) {
   [a = a + 1;]⁴
   [x = a + b;]⁵
 }]³
```

a + b has already been computed whenever statement 3 is reached so we don’t have to recompute it

Denoting AE

- We can describe the set of available expressions for the entry and exit of each program point as a set of expression drawn from the non-trivial syntactic expression in the program $\mathbf{AExp}_*$:
  - $AE_{entry} : \mathrm{Lab}_* \rightarrow P(\mathrm{AExp}_*)$
  - $AE_{exit} : \mathrm{Lab}_* \rightarrow P(\mathrm{AExp}_*)$
- The non-trivial expressions for our program are: `a + b`, `a * b`, and `a + 1`

AE Flow Equations

- A expression is *killed* in a block if any of its variables is defined in the block

  $kill_{\mathrm{AE}} : \mathrm{Lab}_* \rightarrow P(\mathrm{AExp}_*)$

- $kill_{\mathrm{AE}}([x = e\ ;]^l) = \{\ a' \in \mathrm{AExp}_* \mid \mathrm{x} \in vars(a')\}$
- $kill_{\mathrm{AE}}([\ldots]^l) = \emptyset$

AE Flow Equations

- An expression is *generated* in a block if it is evaluated in the block and none of the variables in the expression are subsequently defined in the block

  gen_{AE} : $\text{Lab}_* \rightarrow P(\text{AExp}_*)$

- $gen_{AE}([x = e\ ;]^l) = \{a' \in \text{AExp}(e) \mid x \text{ not in } vars(a')\}$
- $gen_{AE}([\text{while } (b) \ldots]^l) = \text{AExp}(b)$
- $gen_{AE}([\text{if } (b) \ldots]^l) = \text{AExp}(b)$
- $gen_{AE}([\ldots]^l) = \emptyset$

Gen/Kill AE Example

l	$kill_{AE}(l)$	$gen_{AE}(l)$
1	∅	`{ a + b }`
2	∅	`{ a * b }`
3	∅	`{ a + b }`
4	`{a + b, a * b, a + 1}`	∅
5	∅	`{ a + b }`

AE Flow Equations

- The flow equations are now expressed using the two functions

$$
AE_{entry}(l\ ) = \begin{cases} \emptyset, \text{ if } l = b_{init} \\ \bigcap \{\ AE_{exit}(l'\ ) \mid l' \in \mathit{preds}\ (l\ )\ \}, \text{ otherwise} \end{cases}
$$

$$
AE_{exit}(l\ ) = (AE_{entry}(l\ ) \setminus \mathit{kill}_{\mathrm{AE}}(l\ )) \cup \mathit{gen}_{\mathrm{AE}}(l\ )
$$

AE Flow Equations

- Data flow information is propagated in execution order
- Information at entries is calculated in terms of information at preceding exits
- Information at program entry, i.e., b_{init}, indicates that no expressions have been computed at that point

AE Example

l	$AE_{entry}(l)$	$AE_{exit}(l)$
1	∅	{ a + b }
2	{ a + b }	{ a + b, a * b }
3	{ a + b }	{ a + b }
4	{ a + b }	∅
5	∅	{ a + b }

Observation

	RD	AE
Direction	forward	forward
Solution	least	greatest
b_{init} **(f) /** b_{last} **(b) Value**	$\{ (x,\bullet) \mid x \in (\text{Param}_* \cup \text{Field}_*)\} \cup \{ (x,?) \mid x \in \text{Local}_* \}$	$\emptyset$
Combining Operator	$\cup$	$\cap$
Description	may	must
Paths	some	all

Very Busy Expressions (VBE) Analysis

- VBE analysis determines
  - For each program point, which expressions *must* be used before any variable occurring in them is redefined along *all* paths leading from the exit of the program point.

VBE Example

```
int x;
int y;
[if (a > b) {
   [x = b - a;]²
   [y = a - b;]³
 } else {
   [y = b - a;]⁴
   [x = a - b;]⁵
 }]¹
```

a - b and b - a have subsequent uses on all paths leading from the conditional and are therefore *very busy*. We can hoist their calculation out of the conditional bodies.

Denoting VBE

- We can describe the set of very busy expressions for the entry and exit of each program point as a set of expression drawn from the non-trivial syntactic expression in the program $AExp_*$:
  - $VBE_{entry} : Lab_* \rightarrow P(AExp_*)$
  - $VBE_{exit} : Lab_* \rightarrow P(AExp_*)$
- The non-trivial expressions for our program are: a - b and b - a

VBE Flow Equations

- A expression is *killed* in a block if any of its variables is defined in the block

$kill_{\text{VBE}}$: $\text{Lab}_* \to P(\text{AExp}_*)$

- $kill_{\text{VBE}}([x = e\ ;]^l) = \{\ a' \in \text{AExp}_* \mid x \in vars(a')\}$
- $kill_{\text{VBE}}([\ldots]^l) = \emptyset$

VBE Flow Equations

- An expression is *generated* in a block if it is evaluated in the block

  $gen_{\text{VBE}} : \text{Lab}_* \rightarrow P(\text{AExp}_*)$

- $gen_{\text{VBE}}([x = e\ ;]^l) = \text{AExp}(e)$
- $gen_{\text{VBE}}([\text{while } (b) \ldots]^l) = \text{AExp}(b)$
- $gen_{\text{VBE}}([\text{if } (b) \ldots]^l) = \text{AExp}(b)$
- $gen_{\text{VBE}}([\ldots]^l) = \emptyset$

Gen/Kill VBE Example

l	$kill_{VBE}(l)$	$gen_{VBE}(l)$
1	∅	∅
2	∅	{ b – a }
3	∅	{ a - b }
4	∅	{ b – a }
5	∅	{ a - b }

VBE Flow Equations

- The flow equations are now expressed using the two functions

$$\mathrm{VBE}_{exit}(l\,) = \begin{cases} \emptyset, \text{ if } l = b_{last} \\ \bigcap \{ \mathrm{VBE}_{entry}(l'\,) \mid l' \in succs\,(l\,) \}, \text{ otherwise} \end{cases}$$

$$\mathrm{VBE}_{entry}(l\,) = (\mathrm{VBE}_{exit}(l\,) \setminus kill_{\mathrm{VBE}}(l\,)) \cup gen_{\mathrm{VBE}}(l\,)$$

VBE Flow Equations

- Data flow information is propagated opposite to execution order
- Information at exits is calculated in terms of information at succeeding entries
- Statement entry information is calculated from exit information
- Information at program entry, i.e., b_{last}, represents the fact that there are no uses of expressions after termination

VBE Example

l	$VBE_{entry}(l)$	$VBE_{exit}(l)$
1	{ a – b, b - a }	{ a – b, b - a }
2	{ a – b, b - a }	{ a – b }
3	{ a - b }	∅
4	{ a – b, b - a }	{ a – b }
5	{ a – b }	∅

Observation

	RD	AE	VBE
Direction	forward	forward	backward
Solution	least	greatest	greatest
b_{init} **(f) /** b_{last} **(b) Value**	$\{ (x,\bullet) \mid x \in (Param_* \cup Field_*)\} \cup \{ (x,?) \mid x \in Local_* \}$	$\emptyset$	$\emptyset$
Combining Operator	$\cup$	$\cap$	$\cap$
Description	may	must	must
Paths	some	all	all

Live Variable (LV) Analysis

- LV analysis determines
  - For each program point, which variables *must* have a subsequent use on *some path* from the program point prior to the next definition of that variable.

LV Example

```
int x;
int y;
int z;
[x = 2;]1
[y = 4;]2
[x = 1;]3
[if (y > x) {
   [z = y;]5
 } else {
   [z = y * y;]6
 }]4
 [x = z;]7
```

Since the value assigned at 1 is never used, the assignment can be removed

Denoting LV

- We can describe the set of live variables for the entry and exit of each program point as a subset variables in the program Var_*:
  - $\mathrm{LV}_{entry} : \mathrm{Lab}_* \rightarrow P(\mathrm{Var}_*)$
  - $\mathrm{LV}_{exit} : \mathrm{Lab}_* \rightarrow P(\mathrm{Var}_*)$

LV Flow Equations

- A variable is *killed* in a block if it is defined in the block

$$kill_{\mathrm{LV}} : \mathrm{Lab}_* \rightarrow P(\mathrm{Var}_*)$$

- $kill_{\mathrm{LV}}([x = e\ ;]^l) = \{\ x\ \}$
- $kill_{\mathrm{LV}}([\ldots]^l) = \emptyset$

LV Flow Equations

- A variable (use) is *generated* in a block if it is evaluated in the block

  $gen_{\mathrm{LV}} : \mathrm{Lab}_* \rightarrow P(\mathrm{Var}_*)$

- $gen_{\mathrm{LV}}([x = e\ ;]^l) = vars(e)$
- $gen_{\mathrm{LV}}([\mathrm{while}\ (b)\ \ldots\ ]^l) = vars(b)$
- $gen_{\mathrm{LV}}([\mathrm{if}\ (b)\ \ldots\ ]^l) = vars(b)$
- $gen_{\mathrm{LV}}([\ldots\ ]^l) = \emptyset$

Gen/Kill LV Example

l	$kill_{LV}(l)$	$gen_{LV}(l)$
1	{ x }	∅
2	{ y }	∅
3	{ x }	∅
4	∅	{ x, y }
5	{ z }	{ y }
6	{ z }	{ y }
7	{ x }	{ z }

LV Flow Equations

- The flow equations are now expressed using the two functions

$$\mathrm{LV}_{exit}(l\,) = \begin{cases} \emptyset, \text{ if } l = b_{last} \\ \bigcup \{\, \mathrm{LV}_{entry}(l'\,) \mid l' \in succs\,(l\,) \,\}, \text{ otherwise} \end{cases}$$

$$\mathrm{LV}_{entry}(l\,) = (\mathrm{LV}_{exit}(l\,) \setminus kill_{\mathrm{LV}}(l\,)) \cup gen_{\mathrm{LV}}(l\,)$$

LV Flow Equations

- Data flow information is propagated opposite to execution order
- Information at flows in at the final node since we are interested in future uses of variables
- For whole programs, there are no uses subsequent to a final node so we use $\emptyset$ as the exit value of final statements
- If we are analyzing a part of a program we would take Var_* as the initial value since there might be subsequent uses.

LV Example

l	$LV_{entry}(l)$	$LV_{exit}(l)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	{ y }
3	{ y }	{ x, y }
4	{ x, y }	{ y }
5	{ y }	{ z }
6	{ y }	{ z }
7	{ z }	$\emptyset$

Observation

	RD	AE	VBE	LV
Direction	forward	forward	backward	backward
Solution	least	greatest	greatest	least
***b*$_{init}$ (f) / *b*$_{last}$ (b) Value**	$\{ (x,\bullet) \mid x \in (Param_{*} \cup Field_{*})\} \cup \{ (x,?) \mid x \in Local_{*} \}$	$\emptyset$	$\emptyset$	$\emptyset$
Combining Operator	$\cup$	$\cap$	$\cap$	$\cup$
Description	may	must	must	may
Paths	some	all	all	some