NTNU | Norwegian University of Science and Technology

Compiler Construction

Lecture 19 part 2: Live variable analysis

Michael Engel

Overview

- Data-flow analyses
  - Backward analyses: Live variable analysis

Live variable analysis

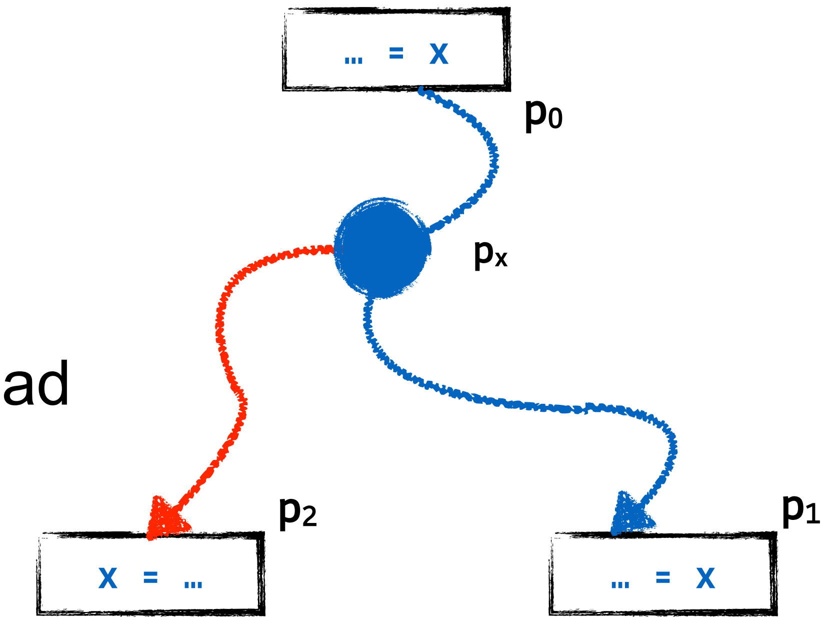
What is Live Variable Analysis?

- For each variable **x** we determine:
  Where is the last program point **p** at which a specific value of **x** is used?
- In other words:
  For **x** and a program point **p** determine if the value of **x** at **p** can still be used along some path starting at **p**
  - If so, **x** is **live** at **p**
  - If not, **x** is **dead** at **p**
- Live variable analysis must take control flow into account
  ⇒ we need to solve a data flow problem

Live variable analysis: example

At point p_0 variable **x** is live:

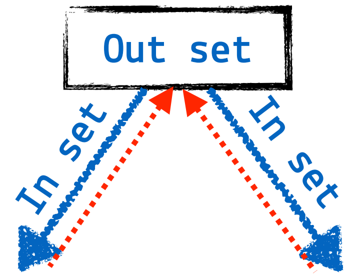
- There is a path to p_1 where the value at p_0 is used
- Beyond p_x towards p_2 the value of **x** is no longer needed and is dead



For each variable and for each program point, we have to observe:

- Where is the last program point beyond which the value is not used?
- **Trace back** from uses to definitions and observe the first definition (backwards) that reaches that use
- That definition **kills** all uses backwards of it

Gen and kill, in and out sets



- A variable is **live** at a point p if its value is used along at least one path
  - A use of x prior to any definition in a basic block means x must be alive
  - A definition of x in a block B prior to any subsequent use means previous uses must be dead
- Accordingly, we obtain:
  - ***Gen**:* set of variables **used** in B
    - the upward exposed reads of variables in block B
  - ***Kill**:* set of variables **defined** in B

$$Out_b = \bigcup_{s \in succ(b)} In_s \qquad In_b = Use_b \cup (Out_b - Def_b)$$

Implementing live variables analysis

- Initialize In_b to the empty set
- Compute **Gen/Use** and **Kill/Def** for each basic block
  - Tracing backwards from the end of the block to the beginning of the block
  - Initialize **last instruction's** Out_i to the empty set
  - Apply $In_i = Use_i \cup (Out_i - def_i)$
- Iteratively apply relations to basic block until convergence
  - $$Out_b = \bigcup_{s \in succ(b)} In_s$$
  - $In_b = Use_b \cup (Out_b - def_b)$
- With Out_b, use relations at instruction level to determine the live variables after each instruction

Compute use and def for a basic block

```
t = a + 1
```

Use = {a}
Def = {t}

```
b = t
```

Use = {t}
Def = {b}

```
if (a=b) goto L2
```

Use = {a,b}
Def = {}

$In = Use \cup (Out - def)$

$Out = \{\}$

$In_i = Use_i \cup (Out_i - def_i)$

Compute use and def for a basic block

```
t = a + 1
```
Use = {a}
Def = {t}

```
b = t
```
Use = {t}
Def = {b}

```
if (a=b) goto L2
```
Use = {a,b}
Def = {}

In **=** {a,b} ∪ ({} - {}) = {a,b}

Out **=** {}

$$In_i = Use_i \cup (Out_i - def_i)$$

Compute use and def for a basic block

`t = a + 1`	Use = {`a`} Def = {`t`}
`b = t`	Use = {`t`} Def = {`b`}
`if (a=b) goto L`$_2$	Use = {`a,b`} Def = {}

$\boldsymbol{In = Use \cup (Out - def)}$

***Out* = {a,b}**

***Out* = {}**

$\boldsymbol{In_i = Use_i \cup (Out_i - def_i)}$

Compute use and def for a basic block

```
t = a + 1
```
Use = {a}
Def = {t}

In = {t} ∪ ({a,b} - {b}) = {a,t}

```
b = t
```
Use = {t}
Def = {b}

Out = {a,b}

```
if (a=b) goto L2
```
Use = {a,b}
Def = {}

Out = {}

$$In_i = Use_i \cup (Out_i - def_i)$$

Compute use and def for a basic block

`t = a + 1`	Use = {`a`} Def = {`t`}
`b = t`	Use = {`t`} Def = {`b`}
`if (a=b) goto L₂`	Use = {`a,b`} Def = {}

$In = Use \cup (Out - def)$

$Out = \{a,t\}$

$Out = \{a,b\}$

$Out = \{\}$

$In_i = Use_i \cup (Out_i - def_i)$

Compute use and def for a basic block

`t = a + 1`	Use = {a} Def = {t}	***In* =** {a} ∪ ({a,t} - {t}) = {a}
		***Out* =** {a,t}
`b = t`	Use = {t} Def = {b}	
		***Out* =** {a,b}
if (a=b) goto L_2	Use = {a,b} Def = { }	
		***Out* =** {}

$$In_i = Use_i \cup (Out_i - def_i)$$

Compute use and def for a basic block

```
t = a + 1
```

Use = {a}
Def = {t}

```
b = t
```

Use = {t}
Def = {b}

```
if (a=b) goto L2
```

Use = {a,b}
Def = {}

In = {a} ∪ ({a,t} - {t}) = **{a}**

Out = {a,t}

Out = {a,b}

Out = {}

$$\boldsymbol{InUse_i = Use_i \cup (OutUse_i - defUse_i)}$$

$\boxed{Use_b = \{a\}}$

Compute use and def for a basic block

```
t = a + 1
```

Use = {a}
Def = {t}

```
b = t
```

Use = {t}
Def = {b}

```
if (a=b) goto L2
```

Use = {a,b}
Def = {}

***OutDef* = {}**

$$\boldsymbol{InDef_i = Def_i \cup (OutDef_i)}$$

Compute use and def for a basic block

Statement	Use / Def
`t = a + 1`	Use = {a} Def = {t}
`b = t`	Use = {t} Def = {b}
`if (a=b) goto L2`	Use = {a,b} Def = { }

$\boldsymbol{InDef} = \boldsymbol{Def} \cup (\boldsymbol{OutDef}) = \{\}$

$\boldsymbol{OutDef} = \{\}$

$$InDef_i = Def_i \cup (OutDef_i)$$

Compute use and def for a basic block

`t = a + 1`	Use = {a} Def = {t}
`b = t`	Use = {t} Def = {b}
`if (a=b) goto L`$_2$	Use = {a,b} Def = { }

$\textbf{\textit{InDef}} = \textbf{\textit{Def}} \cup \textbf{\textit{(OutDef)}} = \{b\}$

$\textbf{\textit{OutDef}} = \{\}$

$\textbf{\textit{OutDef}} = \{\}$

$$InDef_i = Def_i \cup (OutDef_i)$$

Compute use and def for a basic block

```
t = a + 1
```

Use = {a}
Def = {t}

```
b = t
```

Use = {t}
Def = {b}

```
if (a=b) goto L₂
```

Use = {a,b}
Def = {}

InDef = Def ∪ ***(OutDef)***

= {t} ∪ {b}

OutDef = {b}

OutDef = {}

OutDef = {}

$$InDef_i = Def_i \cup (OutDef_i)$$

Compute use and def for a basic block

```
t = a + 1
```
Use = {a}
Def = {t}

```
b = t
```
Use = {t}
Def = {b}

```
if (a=b) goto L2
```
Use = {a,b}
Def = { }

***InDef* =** {t, b}

***OutDef* =** {b}

***OutDef* =** {}

***OutDef* =** {}

$$InDef_i = Def_i \cup (OutDef_i)$$

$\boxed{Def_b = \{t,b\}}$

Liveness semantics

Assuming that variable x is live at the exit of basic block n, there are four possibilities with four distinct semantics:

Case	Local information		Effect on liveness
1	$x\boxed{?} \notin \boldsymbol{Gen_n}$	$x\boxed{?} \notin \boldsymbol{Kill_n}$	Liveness of **x** is unaffected in block **n**
2	$x\boxed{?} \in \boldsymbol{Gen_n}$	$x\boxed{?} \notin \boldsymbol{Kill_n}$	Liveness of **x** is generated in block **n**
3	$x\boxed{?} \notin \boldsymbol{Gen_n}$	$x\boxed{?} \in \boldsymbol{Kill_n}$	Liveness of **x** is killed in block **n**
4	$x\boxed{?} \in \boldsymbol{Gen_n}$	$x\boxed{?} \in \boldsymbol{Kill_n}$	Liveness of **x** is unaffected in block **n** in spite of **x** being modified in **n**

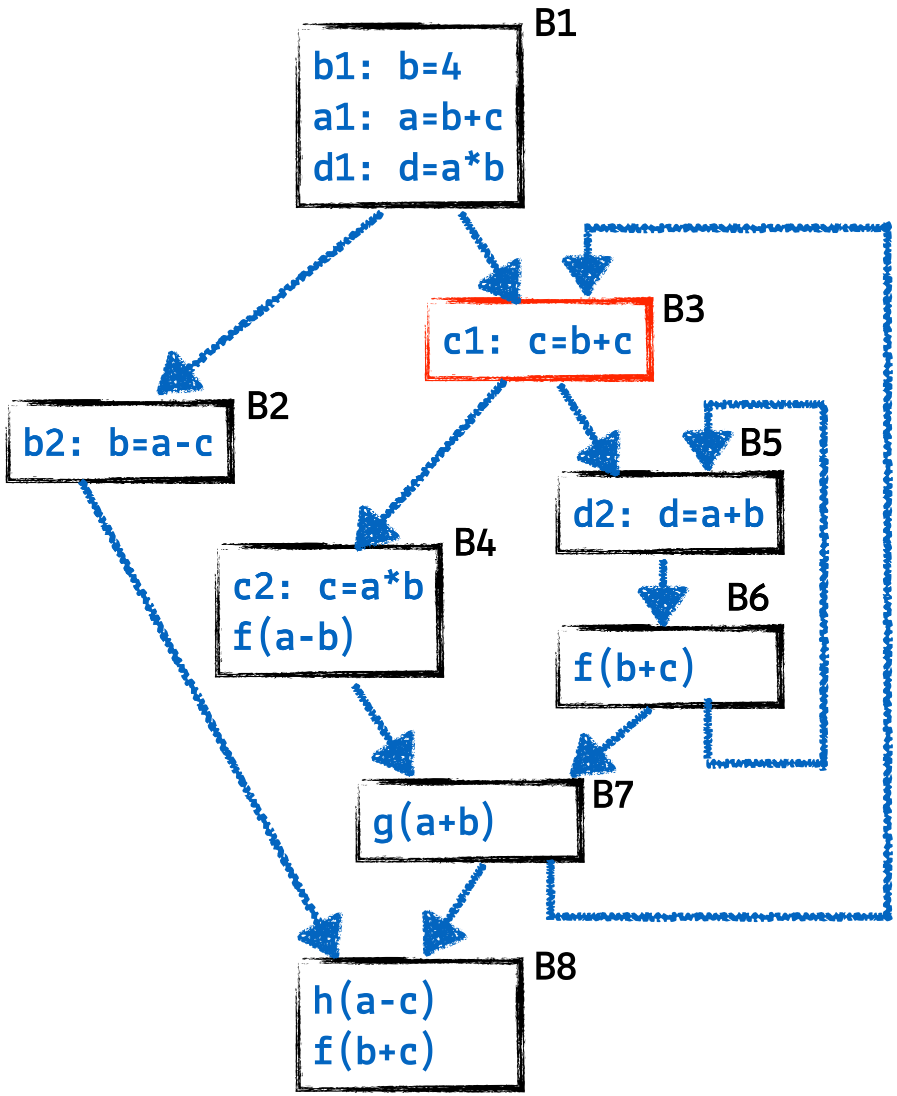
- Variable **x** is live at `Entry(n)` in cases 1, 2, and 4 but the reason for its liveness is different in each case
- Case 4 captures the fact that the liveness of x is killed in n but is regenerated within n

```
a = b + c
c = 42
```

Example

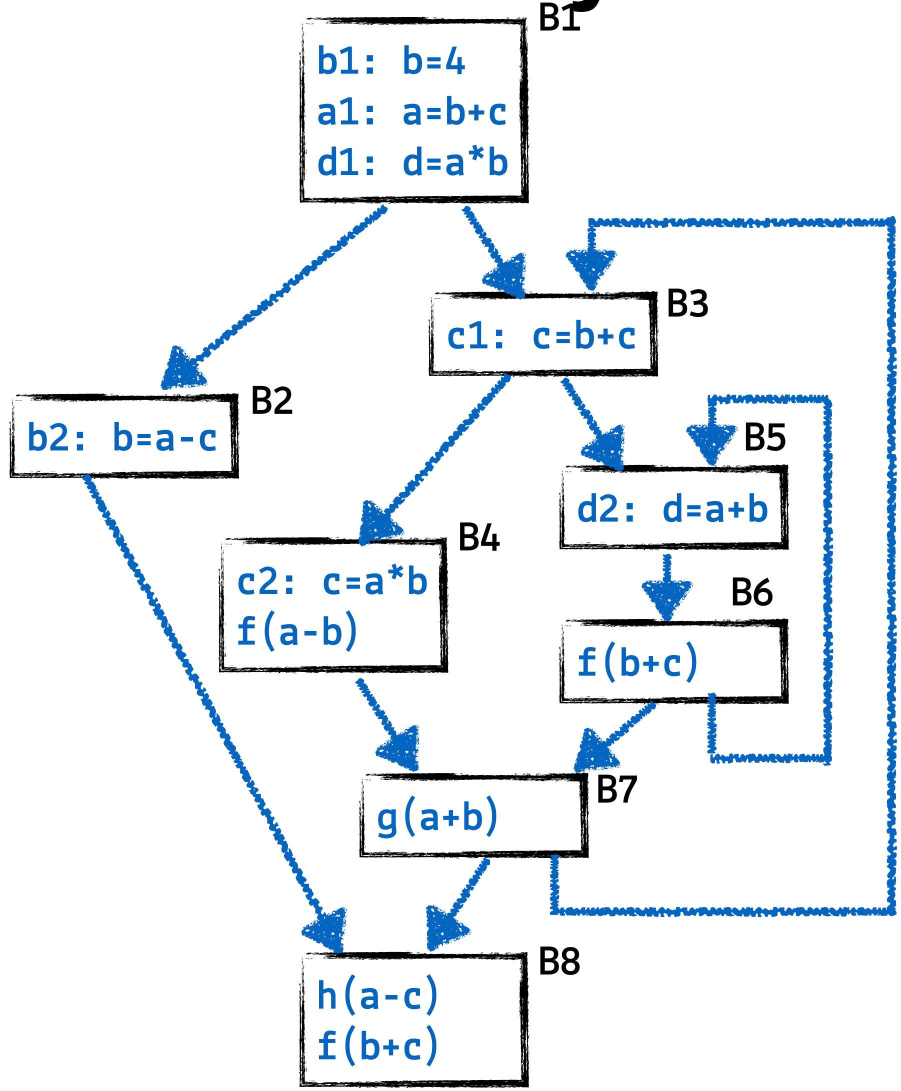
Var = {`a,b,c,d`}
Defs = {`a1,b1,b2,c1,c2,d1,d2`}
Expr = {`a∗b,a+b,a-b,a-c,b+c`}

- Variable `c` is contained in both ***Gen*₃** and ***Kill*₃**



Example: trace of liveness analysis

Block	Local information		Global information			
			Iteration #1		Iteration #2	
	Gen_n	$Kill_n$	Out_n	In_n	Out_n	In_n
B8	{a,b,c}	ø	ø	{a,b,c}	ø	{a,b,c}
B7	{a,b}	ø	{a,b,c}	{a,b,c}	{a,b,c}	{a,b,c}
B6	{b,c}	ø	{a,b,c}	{a,b,c}	{a,b,c}	{a,b,c}
B5	{a,b}	{d}	{a,b,c}	{a,b,c}	{a,b,c}	{a,b,c}
B4	{a,b}	{c}	{a,b,c}	{a,b}	{a,b,c}	{a,b}
B3	{b,c}	{c}	{a,b,c}	{a,b,c}	{a,b,c}	{a,b,c}
B2	{a,c}	{b}	{a,b,c}	{a,c}	{a,b,c}	{a,c}
B1	{c}	{a,b,d}	{a,b,c}	{c}	{a,b,c}	{c}



Example: trace of liveness analysis

Block	Local information		Global information			
	Gen_n	$Kill_n$	Iteration #1		Iteration #2	
			Out_n	In_n	Out_n	In_n
B8	{a,b,c}	ø	ø	{a,b,c}	ø	{a,b,c}
B7	{a,b}	ø	{a,b,c}	{a,b,c}	{a,b,c}	{a,b,c}
B6	{b,c}	ø	{a,b,c}	{a,b,c}	{a,b,c}	{a,b,c}
B5	{a,b}	{d}	{a,b,c}	{a,b,c}	{a,b,c}	{a,b,c}
B4	{a,b}	{c}	{a,b,c}	{a,b}	{a,b,c}	{a,b}
B3	{b,c}	{c}	{a,b,c}	{a,b,c}	{a,b,c}	{a,b,c}
B2	{a,c}	{b}	{a,b,c}	{a,c}	{a,b,c}	{a,c}
B1	{c}	{a,b,d}	{a,b,c}	{c}	{a,b,c}	{c}

The data flow values computed in iteration #2 are identical to the values computed in iteration #1 ⇒ ***convergence***

The result would be ***different*** if we had used the ***universal set*** (here: `{a,b,c,d}`) as initialization. Then, `d` would have been live at `Exit(B7)` whereas `d` is not used anywhere in the program

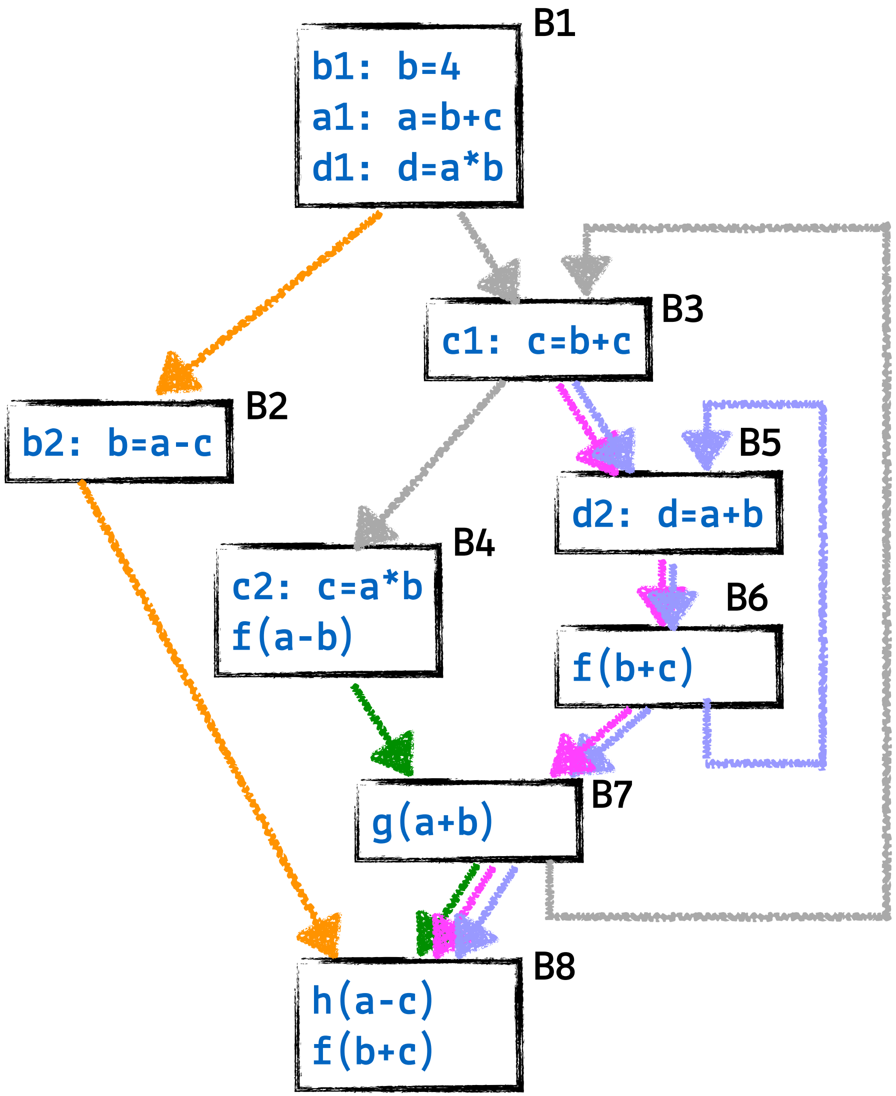
Liveness paths

- For a given variable **x**, liveness analysis discovers a set of **liveness paths**
- Each liveness path is a sequence of blocks (B_1, B_2,…, B_k) which is a prefix of some potential execution path starting at B_1 such that:
  - B_k contains an upwards exposed use of **x**, and
  - **x** is either **Start** or contains an assignment to **x**, and
  - no other block on the path contains an assignment to **x**

Liveness paths

- Some liveness paths for variable `c` in our example program are:

  (B4,B7,B8), (B3,B5,B6,B7,B8), (B3,B5,B6,B5,B6,B7,B8), and (B1,B2,B8)



Applications of liveness analysis

- **Finding uninitialized variables**:
  - Languages like C typically do not define the behavior of programs with uninitialized variables
  - This definition reaches...
    this use…
    but the def might not get executed!
- Common source of security problems [2]

```
…
if (…)
    x = 1
…
a = x
```

Applications of liveness

- **Dead code elimination**:
  - If **x** is not live at a exit of an assignment of **x**, then this assignment can be safely deleted
- Discover useless store operations
  - At an operation that stores **v** to memory, if **v** is not live then the store is useless
- In the example, the assignments `global=1` and `global=3` assign to dead variables
- `i` is not live at the end of `f`, so the assignment can be eliminated

```
int global;
void f ()
{
  int i;
  // dead store:
  i = 1;
  // dead store:
  global = 1;
  global = 2;
  return;
  // unreachable:
  global = 3;
}
```

```
int global;
void f ()
{
  global = 2;
  return;
}
```

Applications of liveness analysis

- **Register allocation**:
  - If a variable **x** is live at a program point, the current value of **x** is likely to be used along some execution path and hence **x** is a potential candidate for being allocated a register
  - On the other hand, if **x** is not live, the register allocated to **x** can be allocated to some other variable without the need of storing the value of **x** in memory
  - More details on register allocation later

Dead variables analysis

- A variable is ***dead*** (i.e., not live) if it is dead ***along all paths***
- We can perform dead variables analysis instead of live variables analysis
- The interpretation of ***Inn*** and $\boldsymbol{Out_n}$ changes
  - If a variable is contained in $\boldsymbol{In_n}$ or $\boldsymbol{Out_n}$, it is dead instead of being live

References

[1] J. C. Beatty (1975).
An algorithm for tracing live variables based on a straightened program graph,
International Journal of Computer Mathematics, 5:1-4, 97-108,
DOI: 10.1080/00207167508803104

[2] http://cwe.mitre.org/data/definitions/457.html