Norwegian University of Science and Technology

Compiler Construction

Lecture 20: Reaching definitions

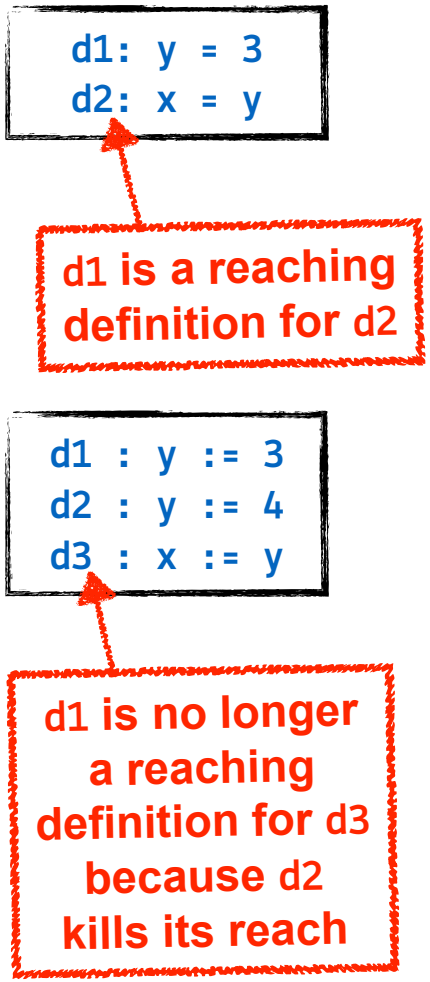
Michael Engel

Overview

- Data-flow analyses
  - Forward analyses: Reaching definitions
  - Uninitialized variables analysis
  - Copy propagation

Reaching definitions analysis

- A **definition** of a variable `x` is a statement which **assigns a value** to `x`
- A unique **label** (representing the def) is associated with each assignment
  - different occurrences of the same assignment become different definitions
- A definition `d` **reaches** a point `p` if there is a path from the point immediately following `d` to `p` such that `d` is not "killed" along that path
- A definition of a variable is **killed** between two points when there is another definition of that variable along the path
  - `r1 = r2 + r3` kills previous definitions of `r1`



Reaching definitions vs. liveness

- Reaching definitions is different from uses of variables or computation of expressions
  - labels are not associated with them and `hence` lexically same computations are not treated as different entities for analysis
- Liveness
  - analyzes **variables** (e.g., virtual registers)
  - doesn't care about specific users
- Reaching defs
  - analyzes **operations**, each def is different
- **Forward dataflow analysis** as propagation occurs from defs downwards
  - liveness was backward analysis

Data flow equations

- A definition $d_i \in Defs$ of a variable $x \in Var$ reaches a program point u if d_i occurs on some path from Start to u and is not followed by any other definition of x on this path
- The data flow equations to define the required analysis are:

$$In_n = \begin{cases} BI & \text{if } n \text{ is Start block} \\ \bigcup_{p \in pred(n)} Out_p & \text{otherwise} \end{cases}$$

$$Out_n = (In_n - Kill_n) \cup Gen_n$$

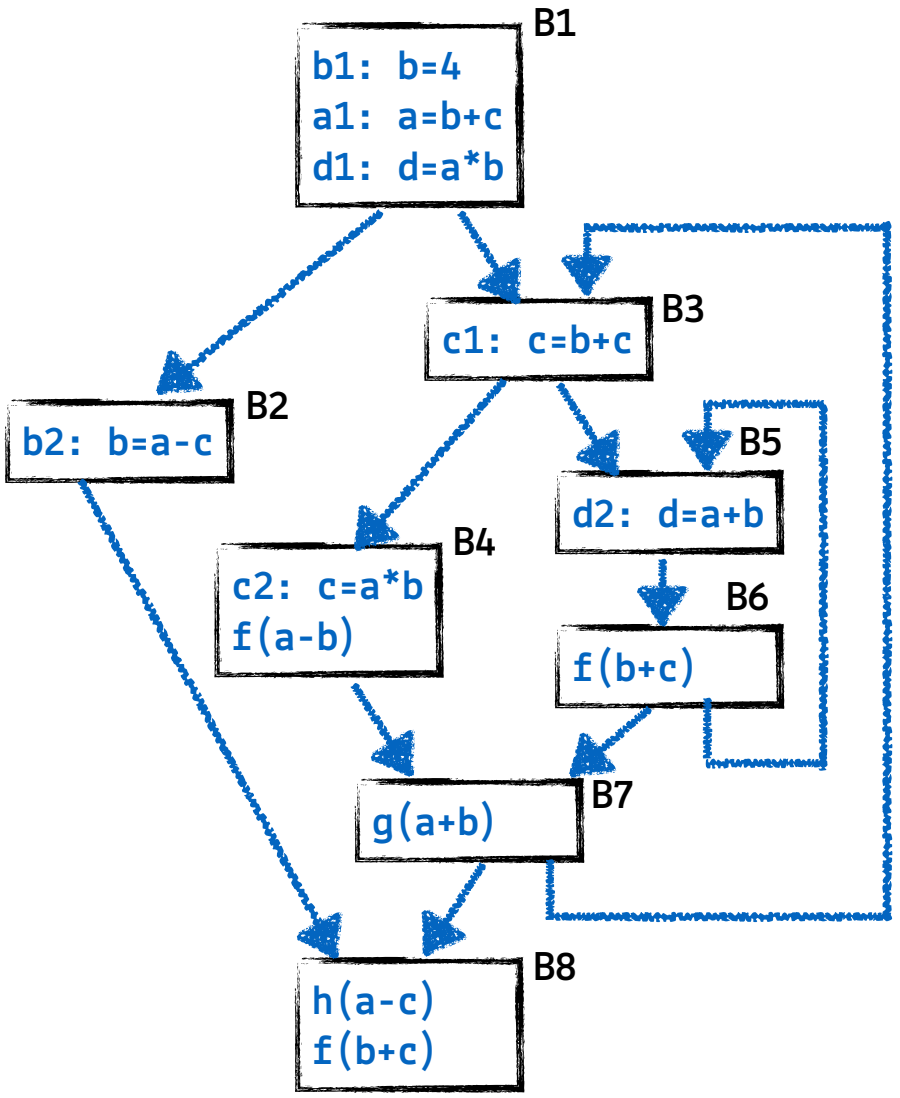
  where In_n, Out_n, Gen_n, $Kill_n$, and BI are sets of definitions
- Note the use of $\cup$ to capture the *"any path"* nature of data flow
  - This is similar to liveness analysis except that now the data flow is forward rather than backward

Assumptions for reaching def. analys.

- For every local variable `x`, it is assumed that a fictitious definition `x = undef` reaches `Entry(Start)`
  - This is required for the optimization of copy propagation (→ discussed later)
- If definition `x = undef` reaches a use of `x`, it suggests a potential use before definition
- Whether this happens at run time depends on the actual results of conditions along the path taken to reach the program point.
- Gen_n contains downwards exposed definitions in `n` whereas $Kill_n$ contains all definitions of all variables modified in `n`
  - Thus $Gen_n \subseteq Kill_n$ for reaching definitions analysis

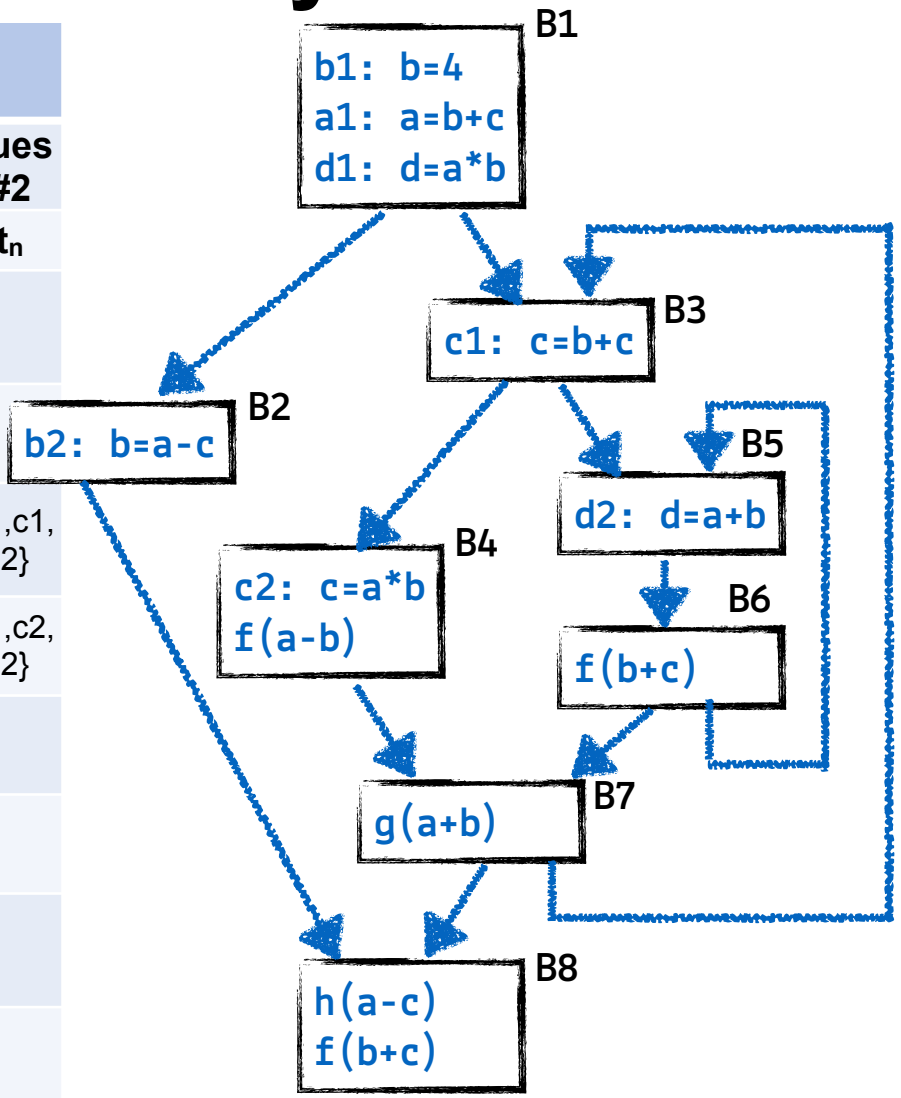
Example

- Labels of assignments consist of variable names and an instance number
  - used to represent the definitions in the programs
- Definitions `a0`, `b0`, `c0`, and `d0` represent the special definitions `a=undef`, `b=undef`, `c=undef`, and `d=undef` respectively
- Since the confluence operation is ∪, the initial value at each program point is ∅



Reaching definitions analysis results

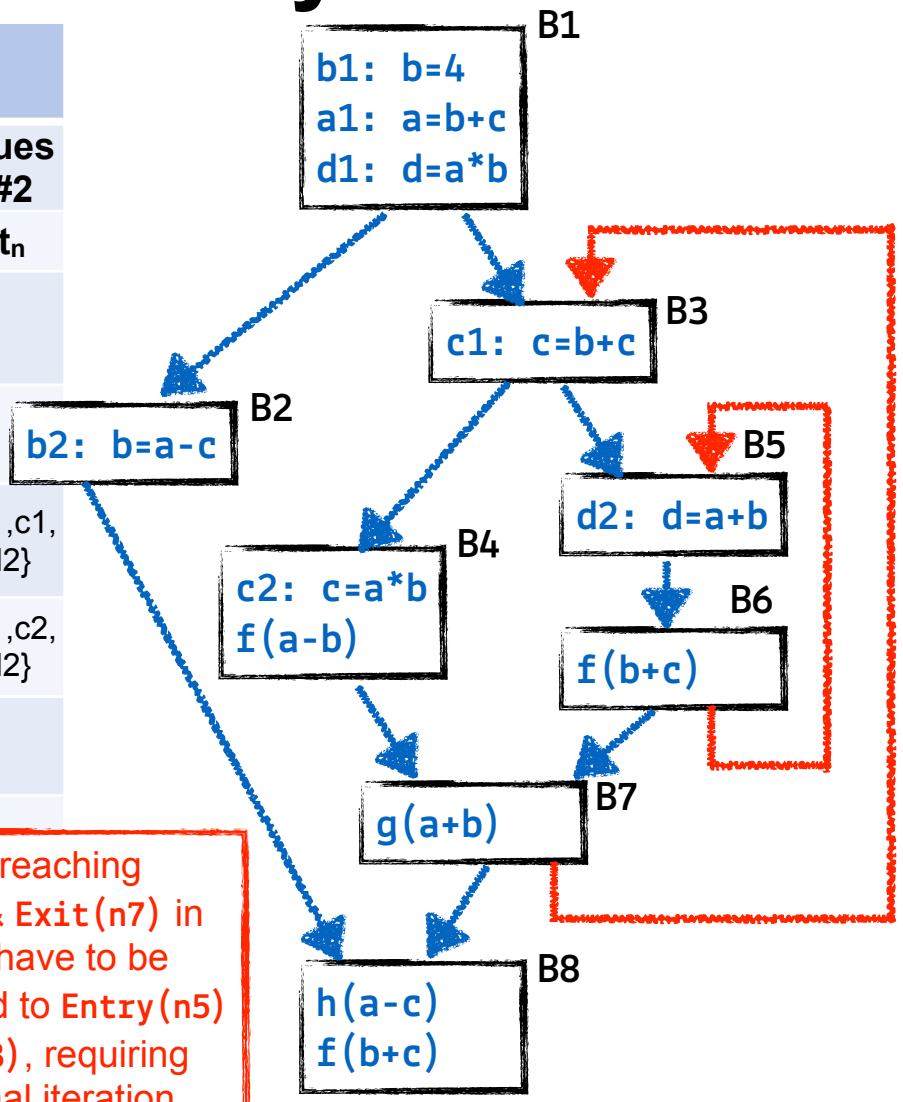
Block	Local information		Global information			
			Iteration #1		Changed values in iteration #2	
	Gen_n	$Kill_n$	In_n	Out_n	In_n	Out_n
B1	{a1,b1, d1}	{a0,a1, b0, b1,b2, d0,d1,d2}	{a0,b0, c0,d0}	{a1,b1, c0,d1}		
B2	{b2}	{b0,b1,b2}	{a1,b1, c0,d1}	{a1,b2, c0,d1}		
B3	{c1}	{c0,c1,c2}	{a1,b1, c0,d1}	{a1,b1, c1,d1}	{a1,b1, c0,c1,c2, d1,d2}	{a1,b1,c1, d1,d2}
B4	{c2}	{c0,c1,c2}	{a1,b1, c1,d1}	{a1,b1, c2,d2}	{a1,b1, c1,d1,d2}	{a1,b1,c2, d1,d2}
B5	{d2}	{d0,d1,d2}	{a1,b1, c1,d1}	{a1,b1, c1,d2}	{a1,b1, c1,d1,d2}	
B6	ø	ø	{a1,b1, c1,d1}	{a1,b1, c1,d2}		
B7	ø	ø	{a1,b1, c1,c2, d1,c2}	{a1,b1, c1,c2, d1,c2}		
B8	ø	ø	{a1,b1, b2,c0, c1,c2, d1,d2}	{a1,b1, b2,c0, c1,c2, d1,d2}		



Reaching definitions analysis results

Block	Local information		Global information			
			Iteration #1		Changed values in iteration #2	
	Gen_n	$Kill_n$	In_n	Out_n	In_n	Out_n
B1	{a1,b1, d1}	{a0,a1, b0, b1,b2, d0,d1,d2}	{a0,b0, c0,d0}	{a1,b1, c0,d1}		
B2	{b2}	{b0,b1,b2}	{a1,b1, c0,d1}	{a1,b2, c0,d1}		
B3	{c1}	{c0,c1,c2}	{a1,b1, c0,d1}	{a1,b1, c1,d1}	{a1,b1, c0,c1,c2, d1,d2}	{a1,b1,c1, d1,d2}
B4	{c2}	{c0,c1,c2}	{a1,b1, c1,d1}	{a1,b1, c2,d2}	{a1,b1, c1,d1,d2}	{a1,b1,c2, d1,d2}
B5	{d2}	{d0,d1,d2}	{a1,b1, c1,d1}	{a1,b1, c1,d2}	{a1,b1, c1,d1,d2}	
B6	ø	ø	{a1,b1, c1,d1}	{a1,b1, c1,d2}		
B7	ø	ø	{a1,b1, c1,c2, d1,c2}	{a1,b1, c1,c2, d1,c2}		
B8	ø	ø	{a1,b1, b2,c0, c1,c2, d1,d2}	{a1,b1, b2,c0, c1,c2, d1,d2}		

definitions reaching `Exit(n6)` & `Exit(n7)` in iteration 1 have to be propagated to `Entry(n5)` & `Entry(n3)`, requiring an additional iteration

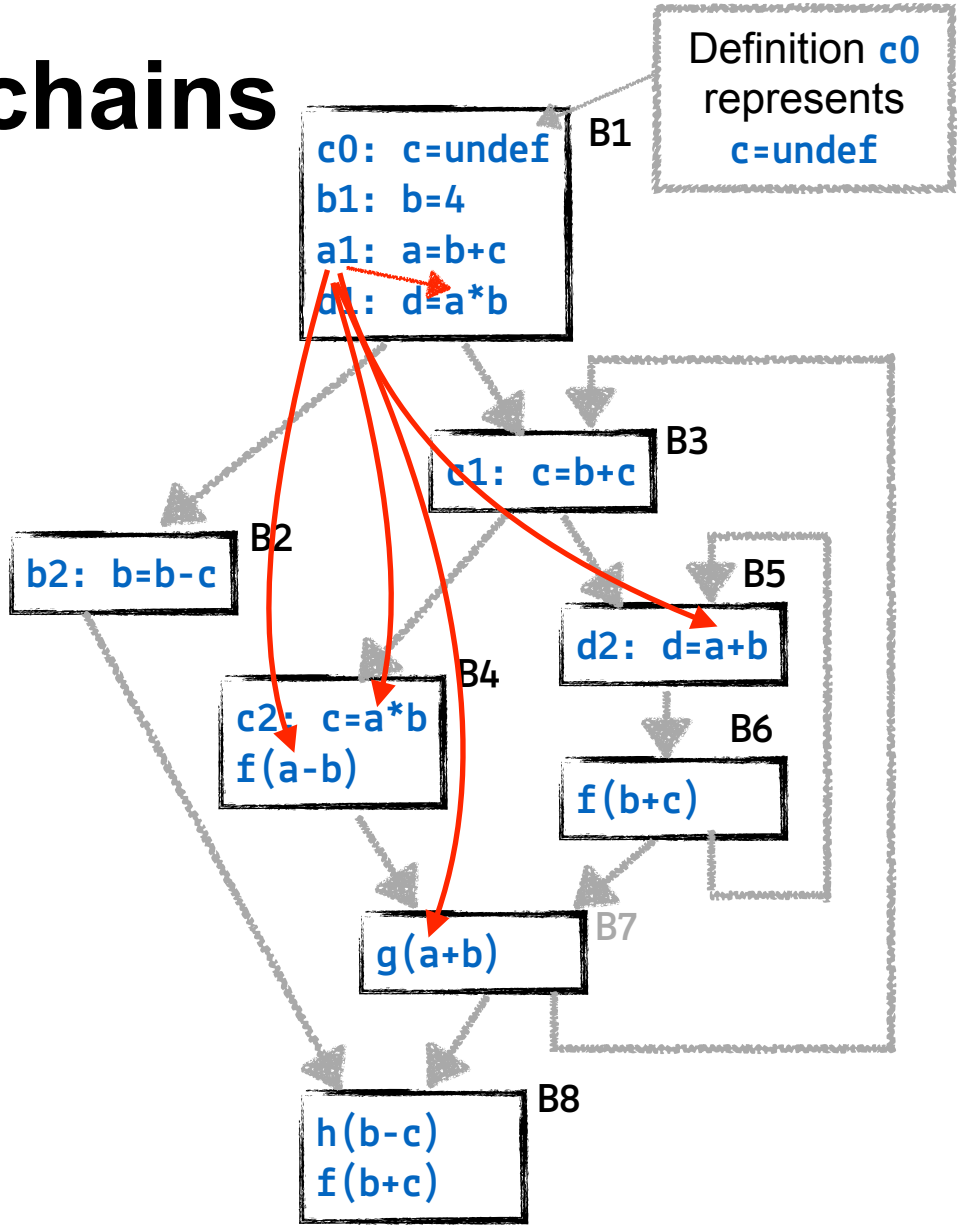


def-use & use-def chains

- Reaching definitions analysis is used for constructing use-def and def-use chains which connect definitions to their uses
  - These chains facilitate several optimizing transformations

Example:
def-use chain for variable `a`

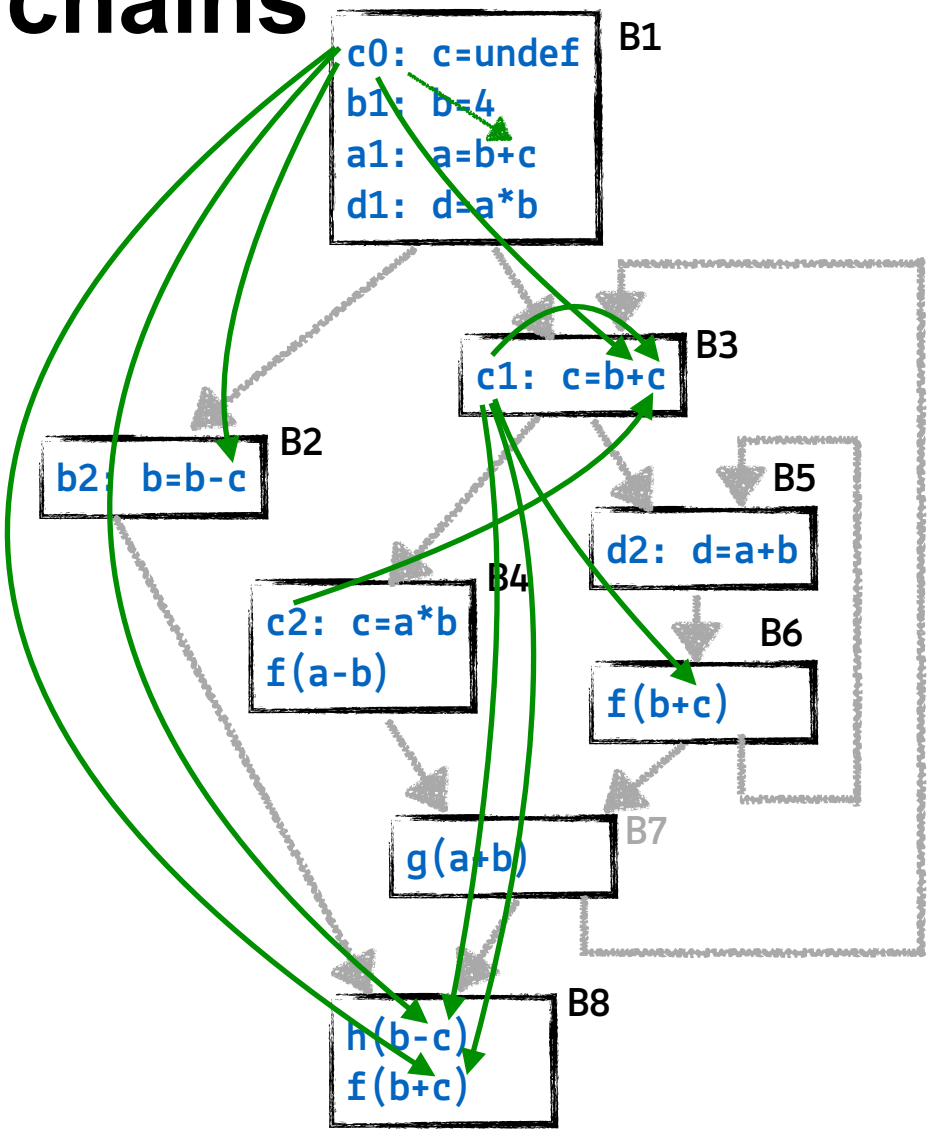
- Chains always start at a *label*



def-use & use-def chains

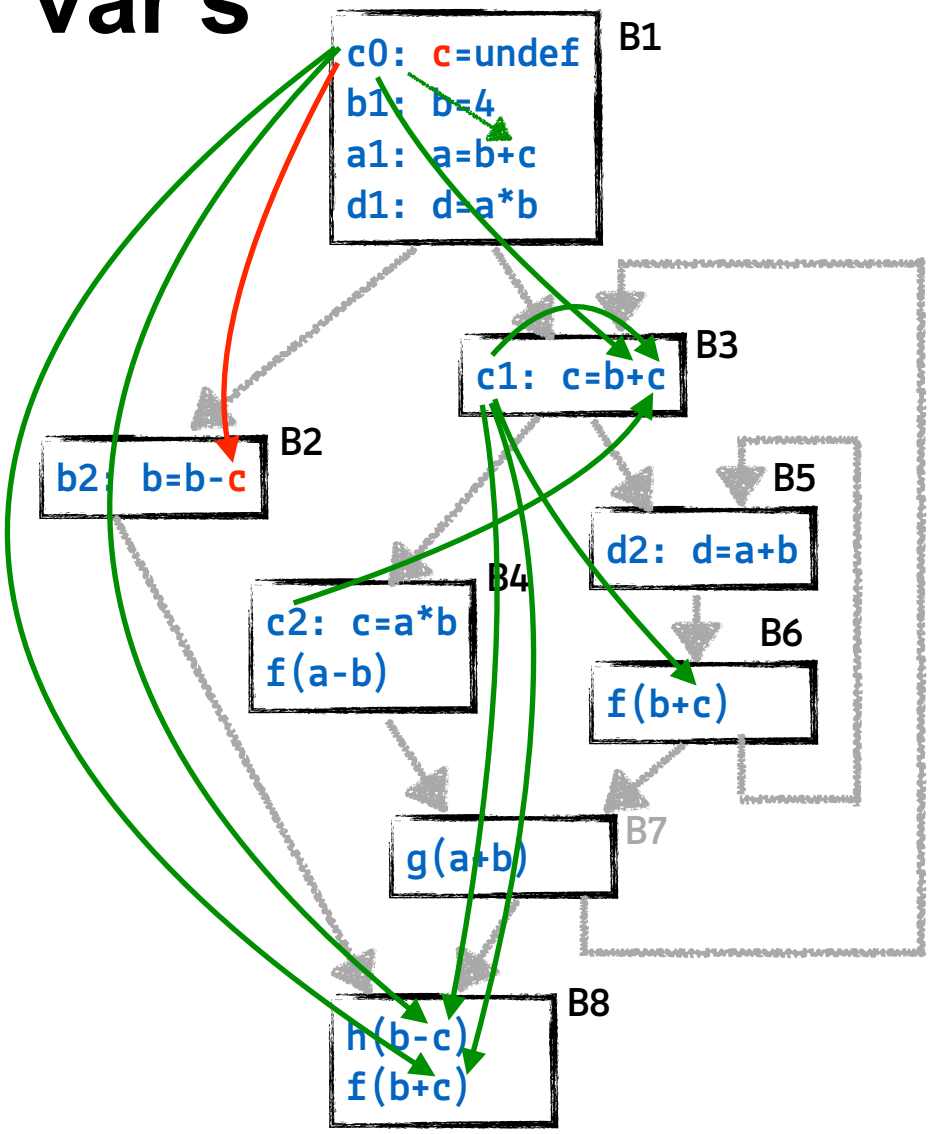
Example:
def-use chain for variable `c`

- Definition `c0` reaches some uses of `c`
- This suggests a potential use before any assigning meaningful value
- This, in turn, makes variable `b` ***potentially undefined***



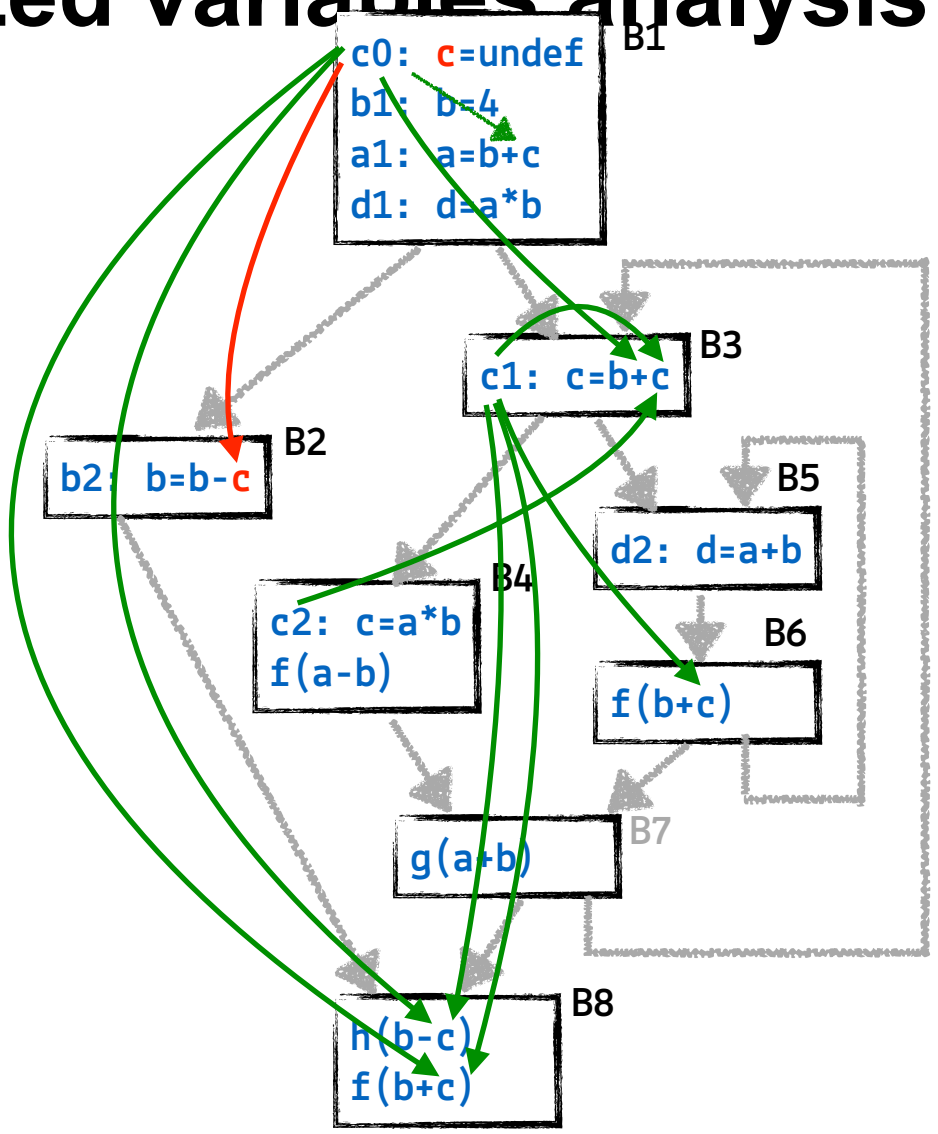
Finding undefined var's

- Definition `c0` reaches some uses of `c`
- This, in turn, makes variable `b` ***potentially undefined***
- ***Transitive effects*** of undefined variables are captured by ***possibly uninitialized variables analysis***
- Possibly uninitialized variables analysis is non-separable – whether a variable is possibly undefined may depend on whether other variables are possibly undefined.



Possibly uninitialized variables analysis

- For definition `xi` of variable `x`, reaching definitions analysis discovers a set of **definition reaching paths:**
- a sequence of blocks `(b1,b2,...,bk)` which is a prefix of some potential execution path starting at `b1` such that:
  - `b1` contains the definition `xi`
  - `bk` is either `End` or contains a definition of `x`
  - no other block in the path contains a definition of `x`
- **Example:** some definition reaching paths for variable `c` are: `(B4,B7,B3)`, `(B3,B5,B6,B7,B3)` and `(B3,B5,B6,B5,B6,B7,B8)`



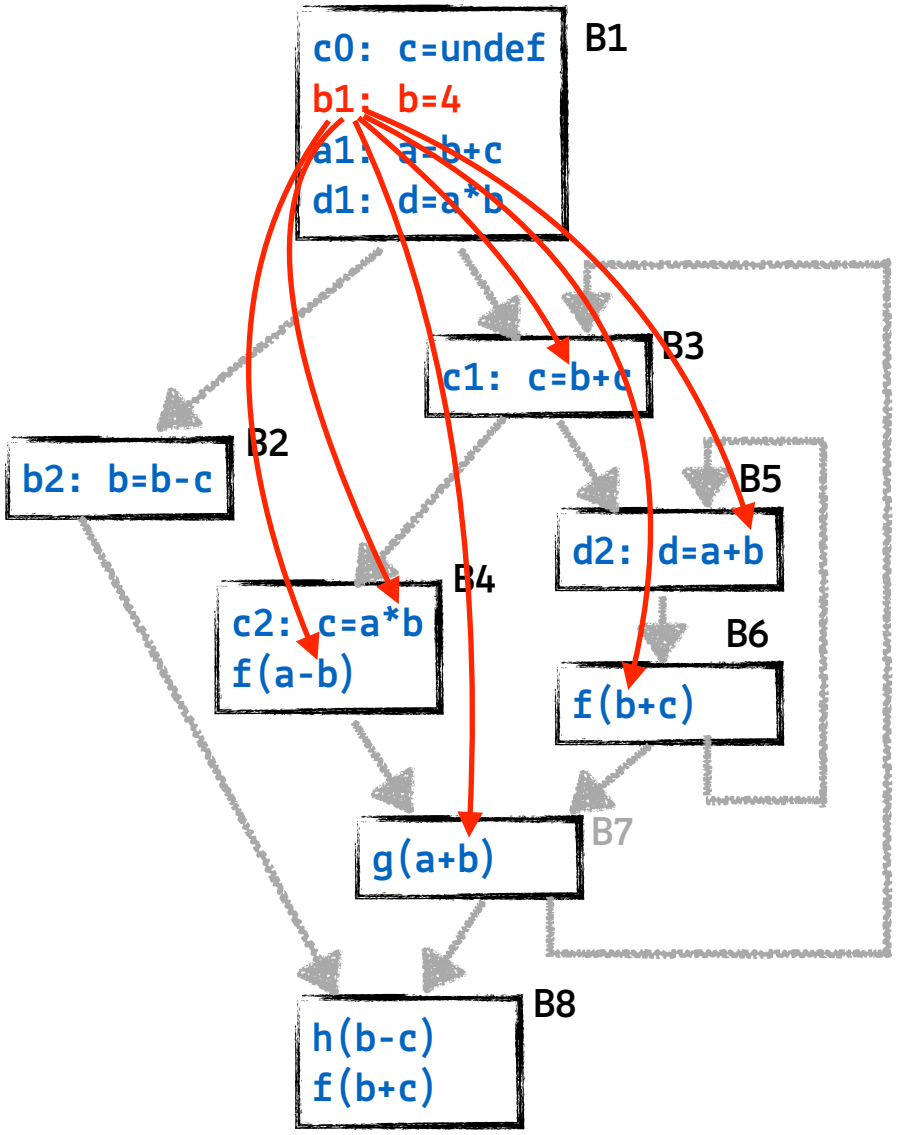
Reaching def. for copy propagation

- Another application of reaching definitions analysis is in performing **copy propagation**
- A definition of the form `x=y` is called a **copy** because it merely copies the value of `y` to `x`
- When such a definition reaches a use of `x`, and no other definition of `x` reaches that use then the use of `x` can be replaced by `y`

Copy propagation

Example:

- Copy `b=4` in block `B1` is the only definition which reaches the uses of `b` in blocks `B3`, `B4`, `B5`, `B6` and `B7`
- Thus all these uses can be replaced by the constant `4`



Copy propagation

```
c0: c=undef
b1: b=4
a1: a=b+c
d1: d=a*b
```

- In the above example, the right hand side (RHS) value is constant
- With variables on the RHS, e.g. `x=y`, replacing the uses of `x` by `y` requires an additional check to ensure that the value of `y` has not been modified along the path from copy to use

A variant of our reaching definitions analysis can accomplish this:

- We restrict the defs to **copies**, a def `x=y` is contained in:
  - $\texttt{Gen}_n$ if it is downwards exposed in `n`, i.e. not being followed by a definition of `x` or `y`, and in
  - $\texttt{Kill}_b$ if `n` contains a definition of `x` or `y`
- We can now perform reaching definitions analysis
- If one def reaches a use, we can perform **copy propagation**

Use of copy propagation

- This copy propagation optimization does not improve the program on its own
- But it has the potential of creating **dead code**:
  - When copy propagation is performed using `x = y`, it is possible that all uses of `x` are replaced by `y` thus making `x` **dead** after the assignment
  - Thus this assignment can be safely deleted

References

[1] Allen, Frances E. and Cocke, John. A catalogue of optimizing transformations. RC 3548, IBM T. J. Watson Research Center, Yorktown Heights, N.Y., September 1971