Dataflow Analysis

Compilers course

Masters in Informatics and Computing Engineering (MIEIC), 3rd Year

João M. P. Cardoso

Dep. de Engenharia Informática
Faculdade de Engenharia (FEUP), Universidade do Porto,
Porto, Portugal
Email:jmpc@acm.org

Outline

- Dataflow Analysis
- Liveness Analysis
- Exercise
- Dataflow Analysis Issues
- Applications of Dataflow Analysis

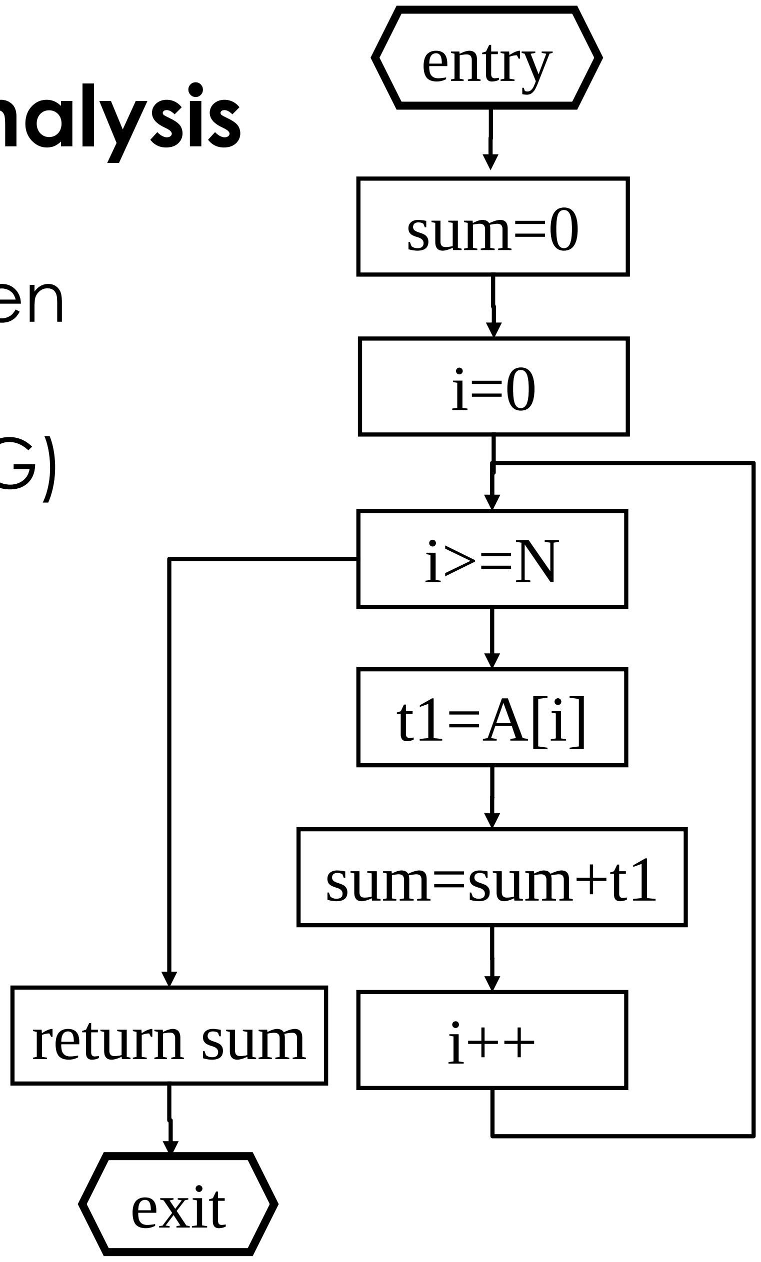
Dataflow Analysis

- Example of a dataflow problem
  - Two variables “a” and “b” can be stored in the same register if their lifetimes do not overlap, but how to determine their lifetimes?
  - We need to analyse the program flow in order to determine the lifetime of each variable
    - A problem known as *liveness analysis*
- *Liveness analysis* is performed using dataflow analysis
- There are many compiler optimizations using dataflow analysis

Dataflow Analysis

- Dataflow analysis is often performed using the control flow graph (CFG)

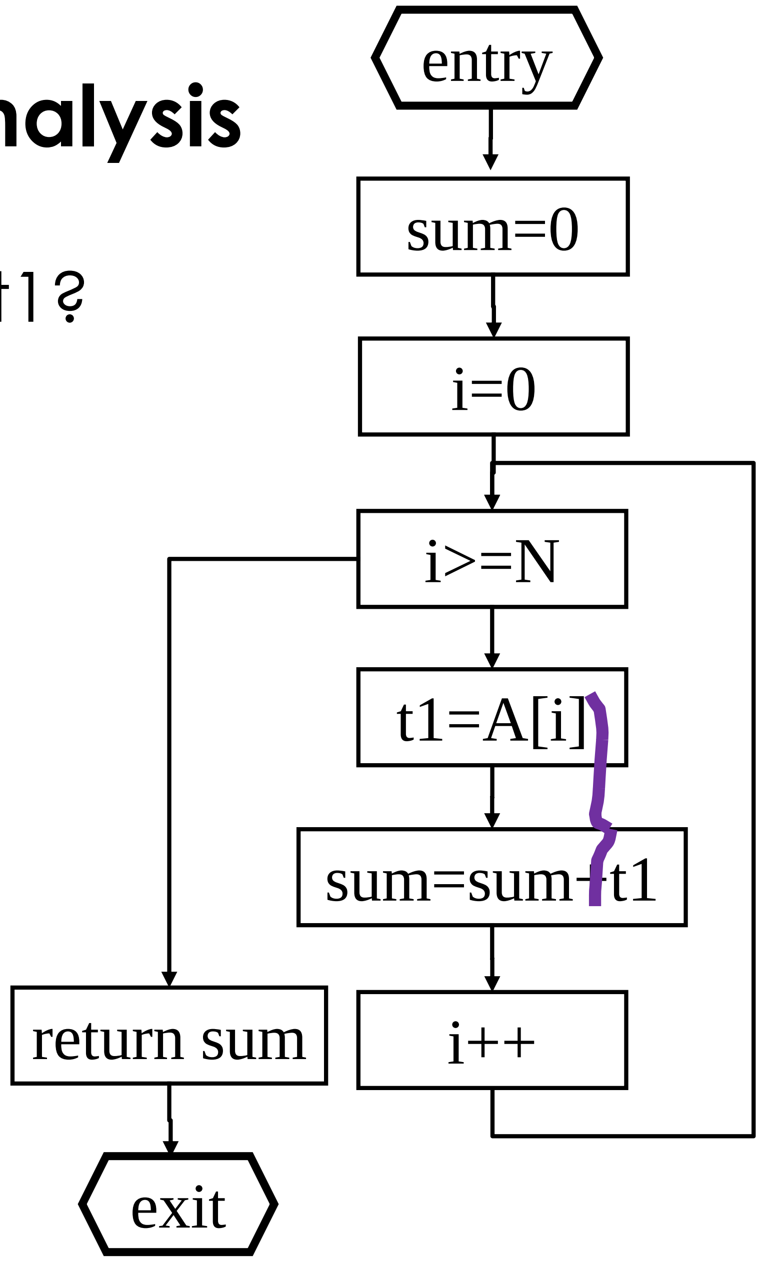
```
        sum = 0;
        i=0;
loop:   If(i>=N) goto end:
        t1=A[i];
        sum = sum + t1;
        i++;
        goto loop;
end:    return sum;
```



Dataflow Analysis

- Live range for variable t1?

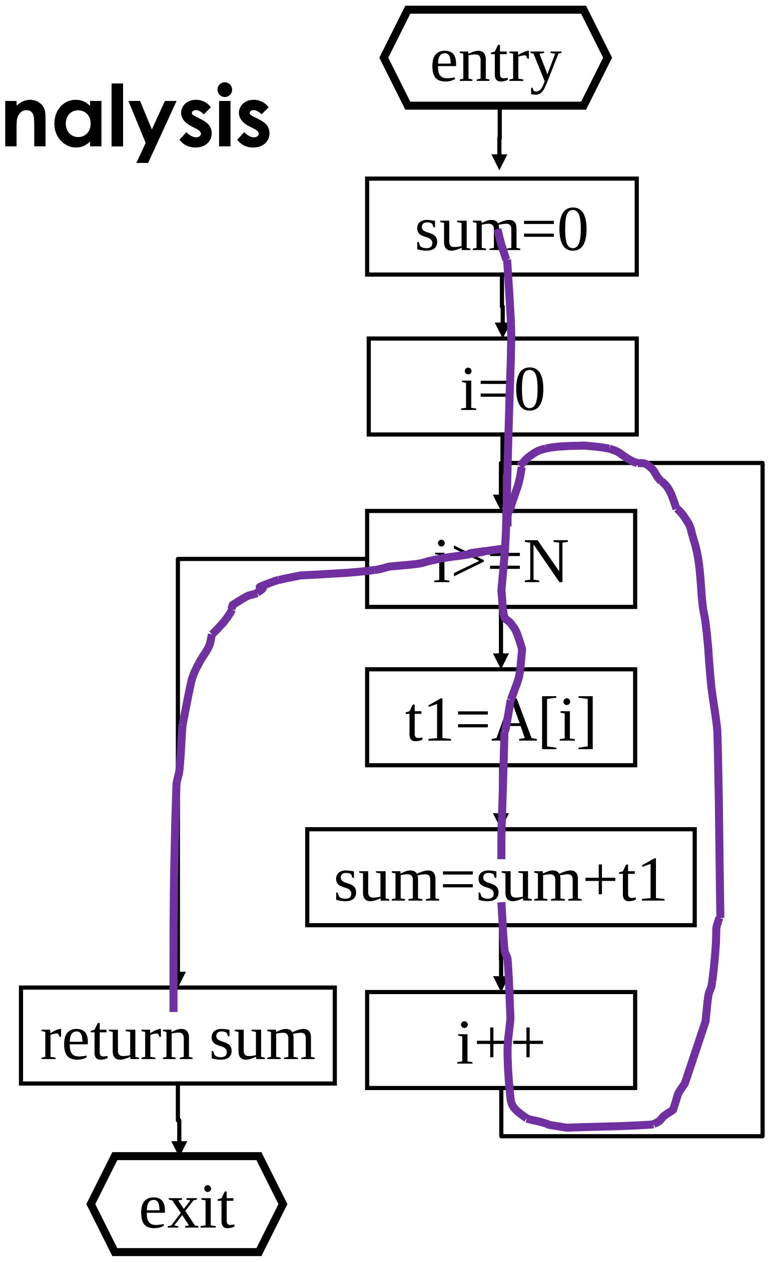
```
       sum = 0;
       i=0;
loop:  If(i>=N) goto end:
       t1=A[i];
       sum = sum + t1;
       i++;
       goto loop;
end:   return sum;
```



Data Flow Analysis

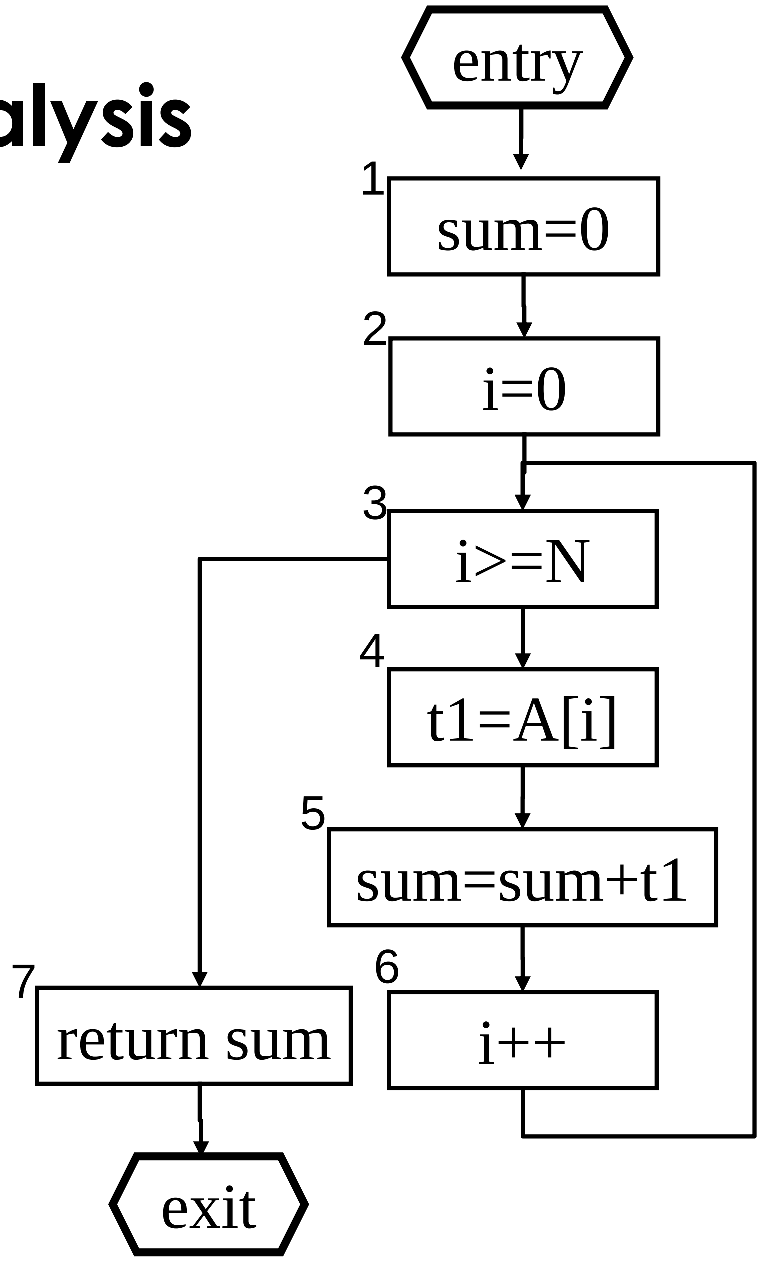
- Live range for variable sum?

```
        sum = 0;
        i=0;
loop:   If(i>=N) goto end:
        t1=A[i];
        sum = sum + t1;
        i++;
        goto loop;
end:    return sum;
```



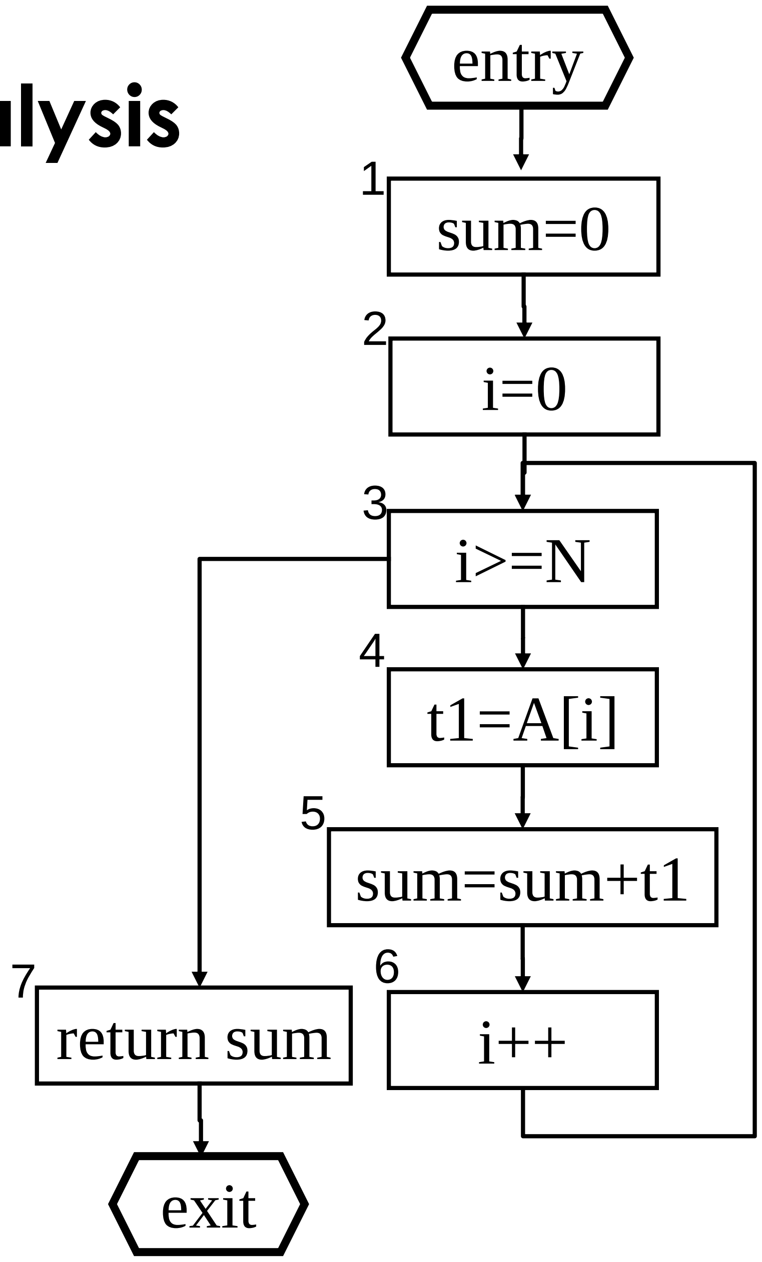
Dataflow Analysis

- Given a node n in a flow graph, there are
  - Out-edges that lead to sucessor nodes
  - In-edges that come from predecessor nodes
- Sets:
  - ***succ[n]*** is the set of sucessors
    - ***succ[3]*** = {4, 7}
  - ***pred[n]*** is the set of predecessors
    - ***pred[3]*** = {2, 6}



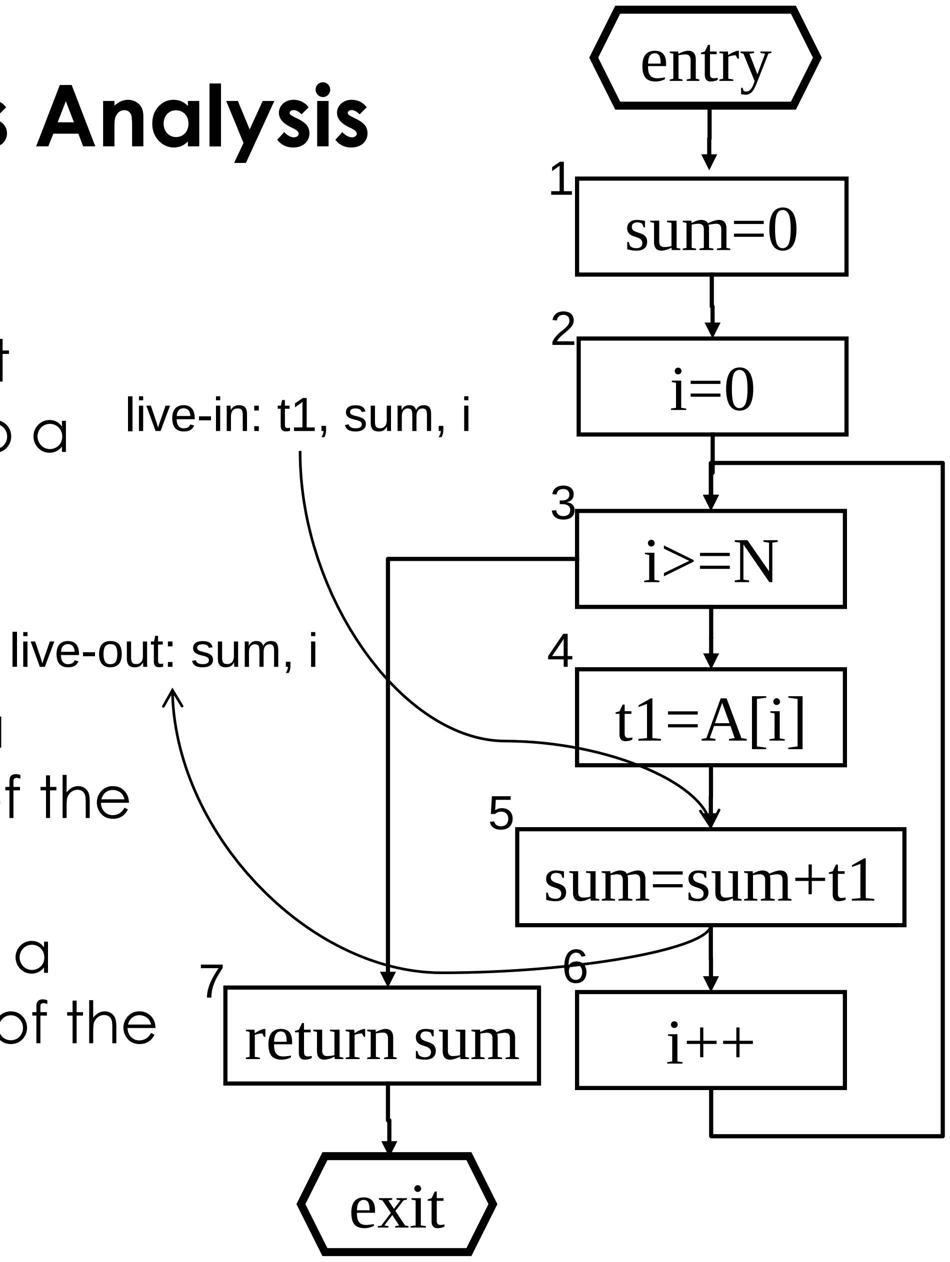
Liveness Analysis

- An assignment to a variable or temporary <u>defines</u> the variable
  - ***def[n]*** is the set of variables defined in n
  - ***def[5]*** = {sum}
- Na occurence of a variable or temporary in the right-hand side of na assignment (or in other expressions) <u>uses</u> the variable
  - ***use[n]*** is the set variables used in n
  - ***use[5]*** = {sum, t1}



Liveness Analysis

- A variable is live on an edge if there is a direct path from that edge to a *use* that does not go through any *def* of the same variable
- A variable is *live-in* at a node if it is live in any of the *in-edges* of the node
- A variable is *live-out* at a node if it is live on any of the *out-edges* of the node

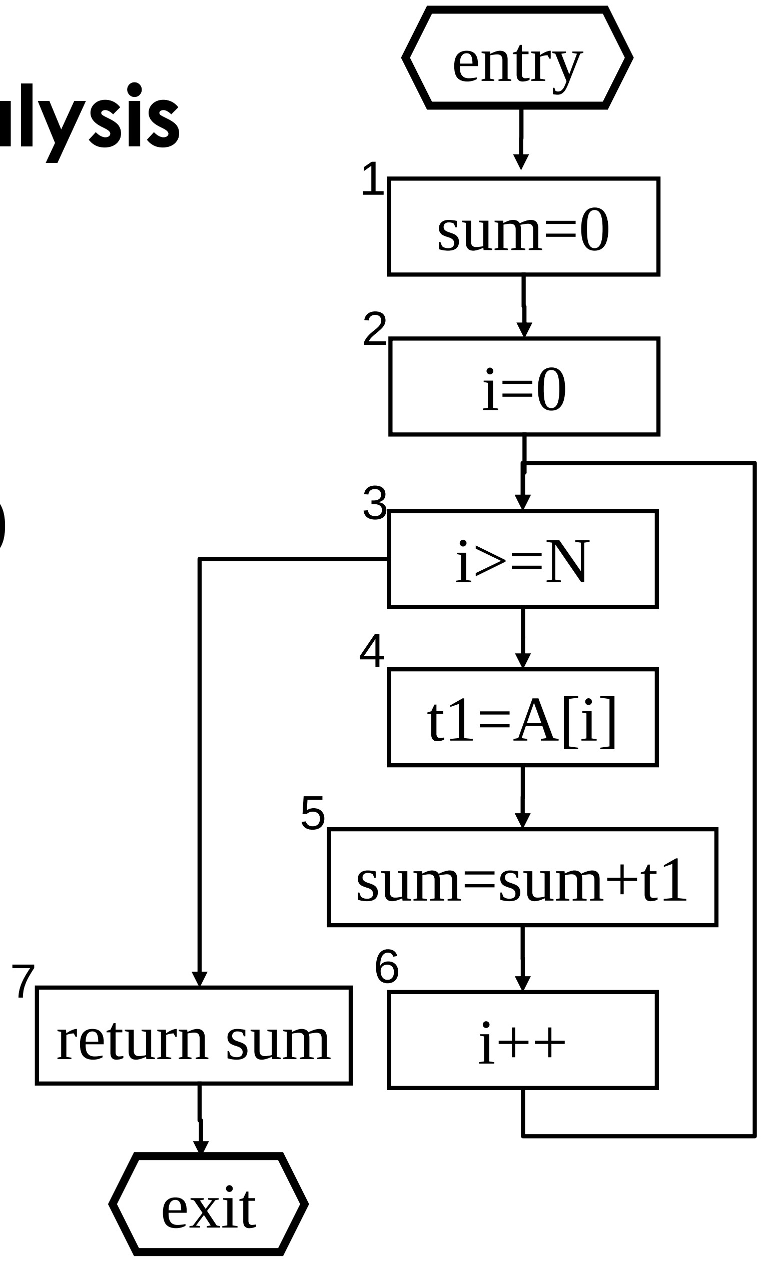


Liveness Analysis

- Dataflow equations for liveness analysis:

$$in[n] = use[n] \cup (out[n] - def[n])$$

$$out[n] = \bigcup_{s \in succ[n]} in[s]$$

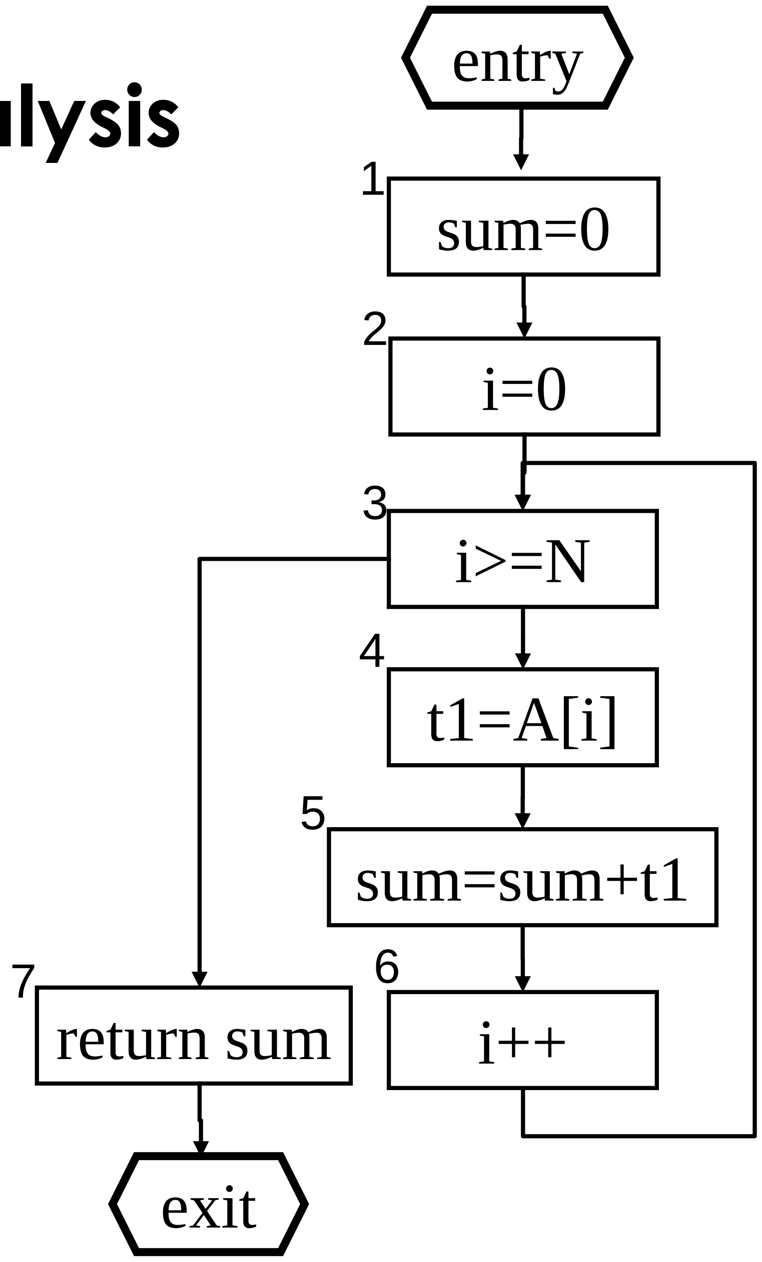


Liveness Analysis

- Computation of liveness analysis by iterative algorithm:

for each n
  in[n] ← {}; out[n] ← {}
repeat
  for each n
    in'[n] ← in[n]; out'[n] ← out[n]
    in[n] ← use[n] ∪ (out[n] – def [n])
    out[n] ← $\bigcup_{s \in succ[n]}$ in[s]

until in'[n] = in[n] and out'[n] = out[n] for all n



Liveness Analysis

- Computation of liveness analysis (forward):

$$in[n] \leftarrow use[n] \cup (out[n] - def\,[n])$$
$$out[n] \leftarrow \bigcup_{s \in succ[n]} in[s]$$

last iteration?

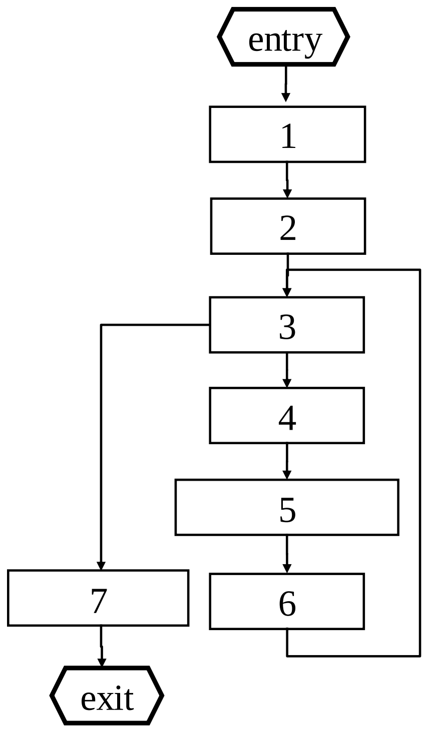
			1st iteration		2nd iteration		3rd iteration		4th iteration		6th iteration	
node	use	def	in	out	in	out	in	out	in	out	in	out
1		s										s
2		i				i		i	s,i		s	s,i
3	i		i		i	s,i	s,i	s,i	s,i	s,i	s,i	s,i
4	i	t	i		i	s,t	s,i	s,t	s,i	s,t,i	s,i	s,t,i
5	s,t	s	s,t		s,t	i	s,t,i	i	s,t,i	i	s,t,i	s,i
6	i	i	i	i	i	i	i	s,i	s,i	s,i	s,i	s,i
7	s		s		s		s		s		s	

Liveness Analysis

- Computation of liveness analysis (backward):

$$out[n] \leftarrow \bigcup_{s \in succ[n]} in[s]$$
$$in[n] \leftarrow use[n] \cup (out[n] - def[n])$$

			1st iteration		2nd iteration		3rd iteration	
node	use	def	out	in	out	in	out	in
7	s			s		s		s
6	i	i		i	s,i	s,i	s,i	s,i
5	s,t	s	i	s,t,i	s,i	s,t,i	s,i	s,t,i
4	i	t	s,t,i	s,i	s,t,i	s,i	s,t,i	s,i
3	i		s,i	s,i	s,i	s,i	s,i	s,i
2		i	s,i	s	s,i	s	s,i	s
1		s	s		s		s	



Liveness Analysis

- It is a backward problem
- Thus we should use a backward dataflow analysis formulation
  - Note that the use of

$$out[n] = \bigcup_{s \in succ[n]} in[s]$$

$$in[n] = use[n] \cup (out[n] - def[n])$$

  - Instead of:

$$in[n] = use[n] \cup (out[n] - def[n])$$

$$out[n] = \bigcup_{s \in succ[n]} in[s]$$

  - Saves iterations

Liveness Analysis

- It is a backward problem
- ***Depth-first search*** can be used for ordering the nodes of the CFG
- What would be the order of nodes for the CFG of the previous example?

https://en.wikipedia.org/wiki/Depth-first_search

Exercise 1

- Computation of liveness analysis (backward):

			1st iteration		2nd iteration		3rd iteration	
stmt	use	def	out	in	out	in	out	in
6								
5								
4								
3								
2								
1								

Example:
1: t1=x*x;
2: t2=a*t1;
3: t3=b*x;
4: t4=t3+c;
5: t5=t4+t2;
6: y=t5;

Exercise 1

- Computation of liveness analysis (backward):

Example:
1: t1=x*x;
2: t2=a*t1;
3: t3=b*x;
4: t4=t3+c;
5: t5=t4+t2;
6: y=t5;

			1st iteration		2nd iteration	
stmt	use	def	out	in	out	in
6	t5	y		t5		t5
5	t4,t2	t5	t5	t4,t2	t5	t4,t2
4	t3,c	t4	t4,t2	t3,c,t2	t4,t2	t3,c,t2
3	b,x	t3	t3,c,t2	b,x,c,t2	t3,c,t2	b,x,c,t2
2	a,t1	t2	b,x,c,t2	a,t1,b,x,c	b,x,c,t2	a,t1,b,x,c
1	x	t1	a,t1,b,x,c	x,a,b,c	a,t1,b,x,c	x,a,b,c

Exercise 2

- Computation of liveness analysis
- Do we need an iterative algorithm for computing liveness analysis in straightline code (as in the example above)?

Example:

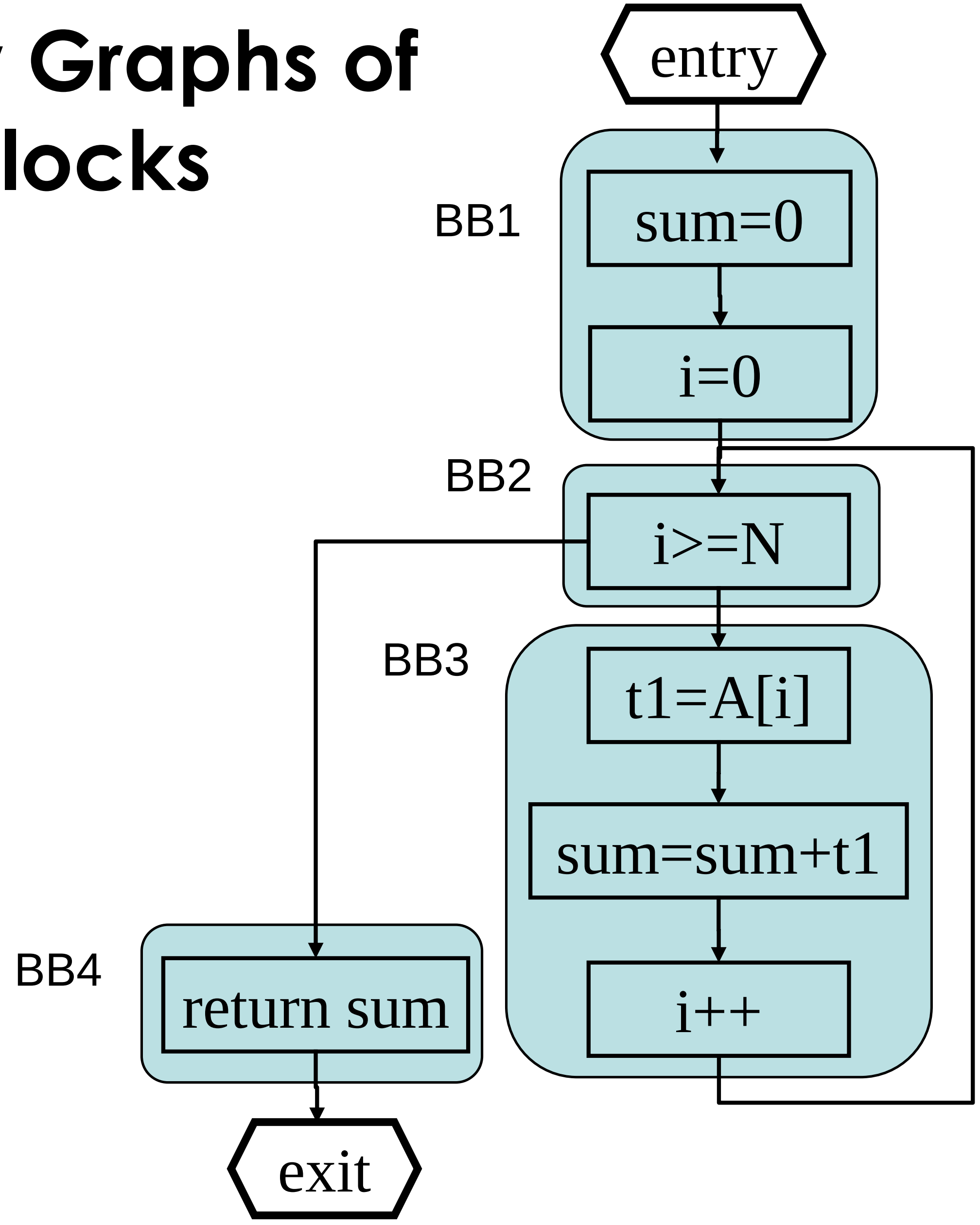
```
1: t1=x*x;
2: t2=a*t1;
3: t3=b*x;
4: t4=t3+c;
5: t5=t4+t2;
6: y=t5;
```

Dataflow Analysis Issues

- Speed of the iterative algorithm depends
  - on the number of nodes of the CFG (control flow graph) and
  - on the visiting order of the nodes of the CFG
- It is common to use CFGs consisting of basic blocks (see next slide) instead of one node per instruction
- Order of visiting should be according to the flow direction of the problem (e.g., liveness analysis is a backward problem)

Control Flow Graphs of Basic Blocks

- CFGs consisting of basic blocks (BBs)
- Basic blocks are the largest groups of instructions where
  - the control flow begins at the first instruction and only exit in the last one
  - Once in a basic block, all its Instructions are executed



Complementary Information

Applications of Dataflow Analysis

- Used for many optimizations
  - liveness analysis
  - def-use and use-def computations
  - constant propagation
  - copy propagation
  - dead-code elimination
  - common subexpression elimination
  - etc.
- Example:
  - How to determine the definitions of a variable that reach a certain use of the variable?