Data Flow Analysis

Daqing Hou

Outline

- Basic blocks and control flow graph
- An example: reaching definitions
- Characteristics of DFA (Data Flow Analysis)

Example

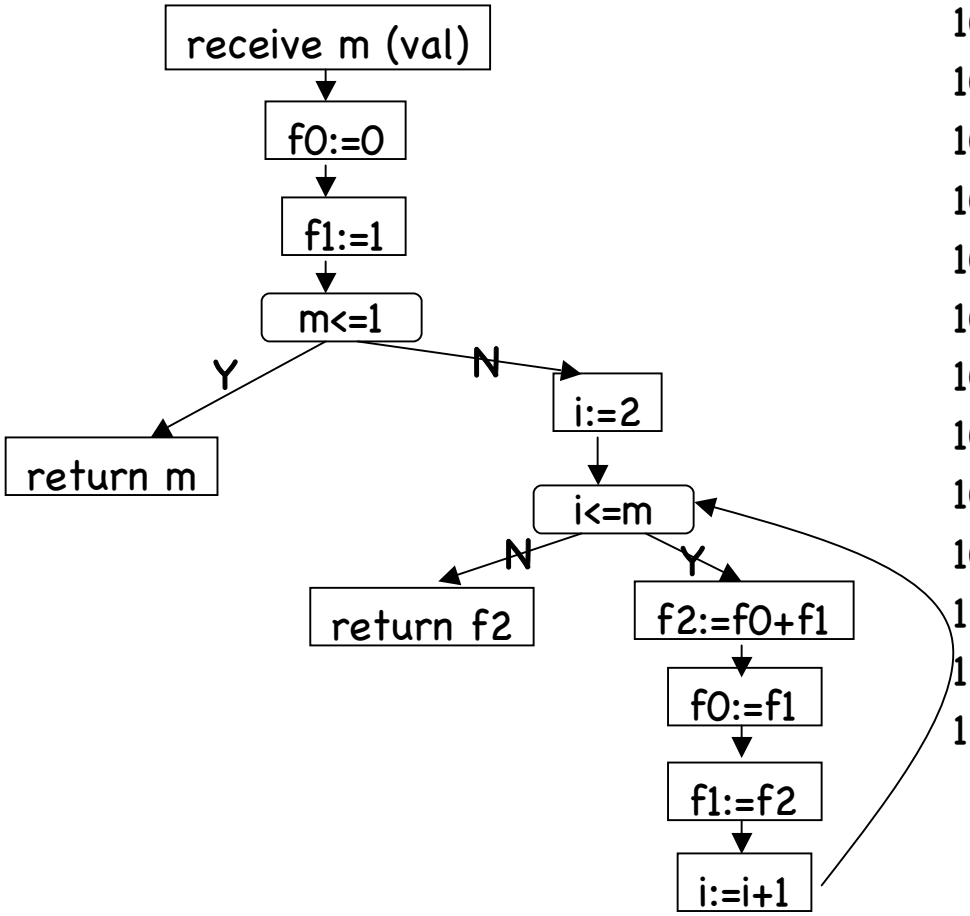
```
unsigned int fib(unsigned int m) {
unsigned int f0=0, f1=1, f2;
if (m<=1) {
    return m;
}
else {
for (int i=2;i<=m; ++i){
    f2=f0+f1;
    f0=f1;
    f1=f2;
}
return f2;
}
```

a procedure computing Fibonacci number

```
receive m (val)
f0 := 0
f1 := 1
if m<=1 goto L3
i := 2
L1: if i<=m goto L2
    return f2
L2: f2:=f0+f1
    f0:=f1
    f1:=f2
    i:=i+1
    goto L1
L3: return m
```

MIR representation

Flow chart

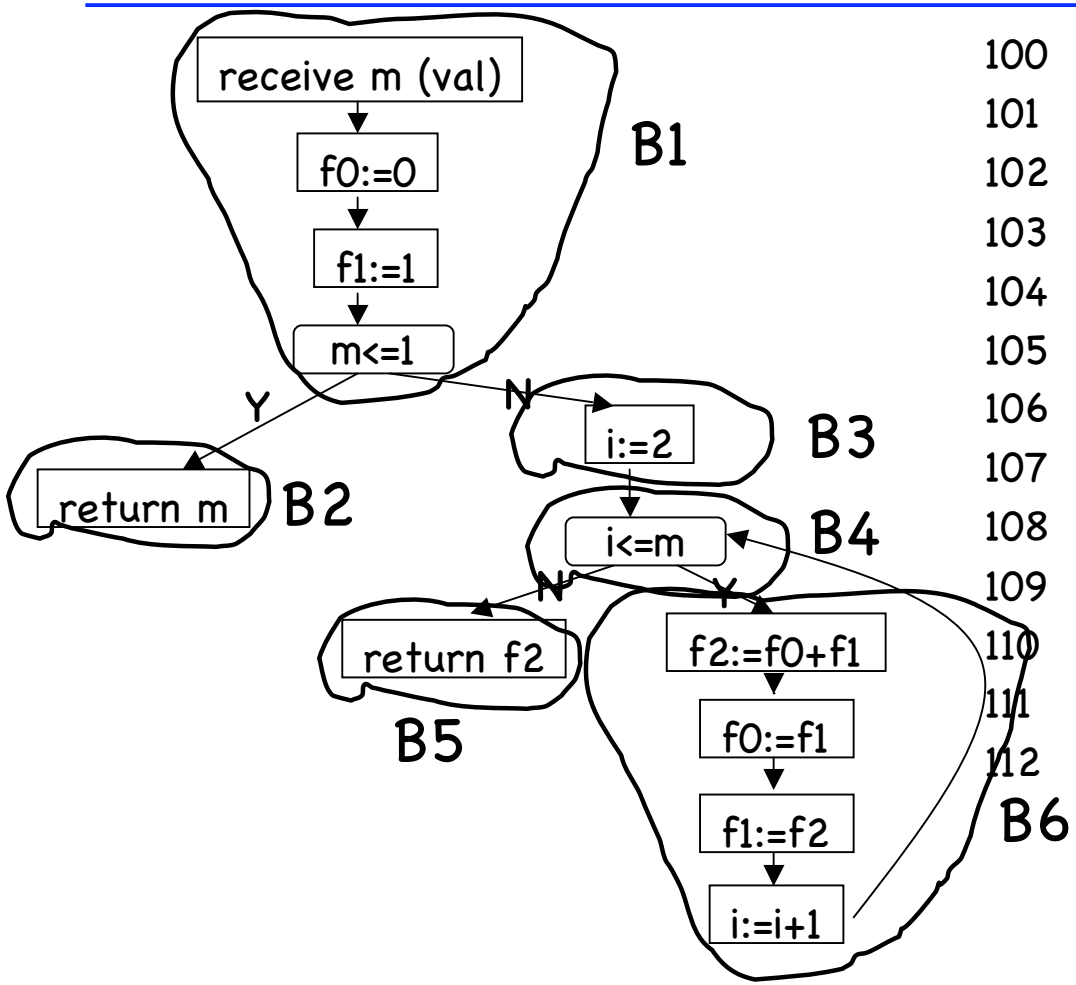



```
100   receive m (val)
101   f0 := 0
102   f1 := 1
103   if m<=1 goto L3
104   i := 2
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1
108       f0:=f1
109       f1:=f2
110       i:=i+1
111       goto L1
112   L3: return m
```

Definition: basic blocks

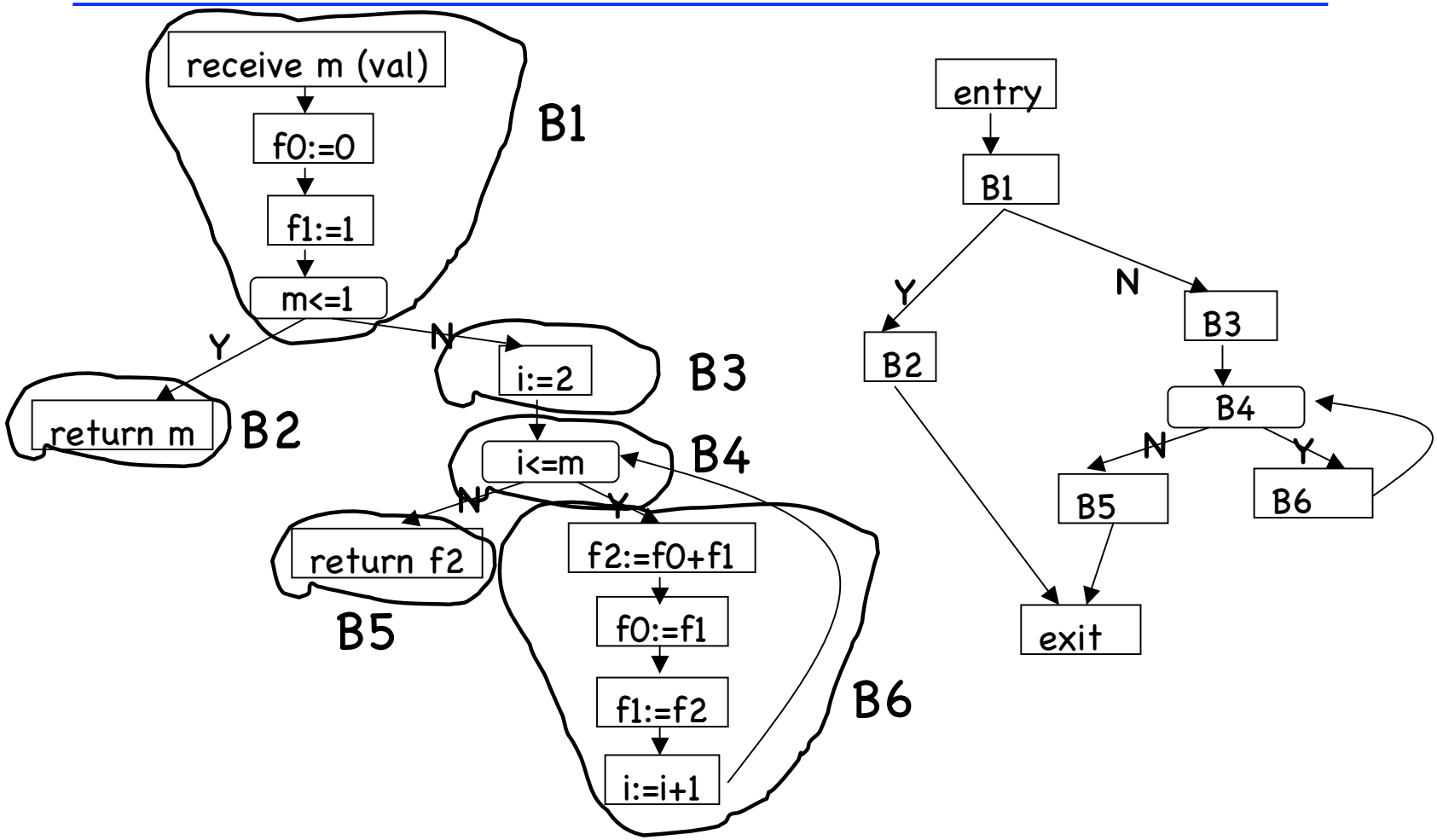
A **basic block** consists of a **longest** consecutive sequence of instructions, where control enters the block ONLY at the first instruction and exits ONLY from the last one.

Example: basic blocks




```
100   receive m (val)
101   f0 := 0
102   f1 := 1
103   if m<=1 goto L3
104   i := 2
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1
108       f0:=f1
109       f1:=f2
110       i:=i+1
111       goto L1
112   L3: return m
```

Flow graph


receive m (val)
f0:=0
f1:=1
m<=1
B1
Y
N
return m
B2
i:=2
B3
i<=m
B4
N
Y
return f2
B5
f2:=f0+f1
f0:=f1
f1:=f2
i:=i+1
B6
entry
B1
Y
N
B2
B3
B4
N
Y
B5
B6
exit

Summary: basic blocks and flow graph

- Program can be represented as a sequence of instructions
- A rep. has two kinds of instructions: expressions and control (if, goto)
- **Basic blocks**: a consecutive sequence of instructions, where control enters the block ONLY at the first instruction and exits ONLY from the last one.
- Control flows from blocks to blocks, forming a **flow graph** for the program

HW#5

Weight: 3%

Draw a CFG (control flow graph) for the following function find, which, given an array a with N integers, will return its f'th one in the descending order of the array.e.g., for a={4,1,3,3,2}, and f=2f(a, 5, 2) will return 3.

```
int find(int a[], int N, int f){ int
    m=0, n=N-1; while (m<n){   int
    r=a[f], i=m, j=n;   while (i<=j){
    while (a[i]<r) ++i; // 16
    while (a[j]>r) --j;         if
    (i<=j){             int tmp = a[i];
    a[i] = a[j]; a[j]=tmp;
    ++i; --j; // 10         }   }   if
    (f<=j) {n=j;}   else if (f>=i)
    {m=i;}   else {m=n=f;} //break; }
```

Definitions

A **definition** is an assignment of some value to a variable at a certain point (including := and read)

```
100   receive m (val)
101   f0 := 0
102   f1 := 1
103   if m<=1 goto L3
104   i := 2
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1
108       f0:=f1
109       f1:=f2
110       i:=i+1
111       goto L1
112   L3: return m
```

Reaching definitions

Reaching definitions for a particular point are all definitions that **may** reach the point by some control-flow path on which the variable is not re-defined

```
100   receive m (val)
101   f0 := 0
102   f1 := 1
103   if m<=1 goto L3
104   i := 2
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1
108       f0:=f1
109       f1:=f2
110       i:=i+1
111       goto L1
112   L3: return m
```

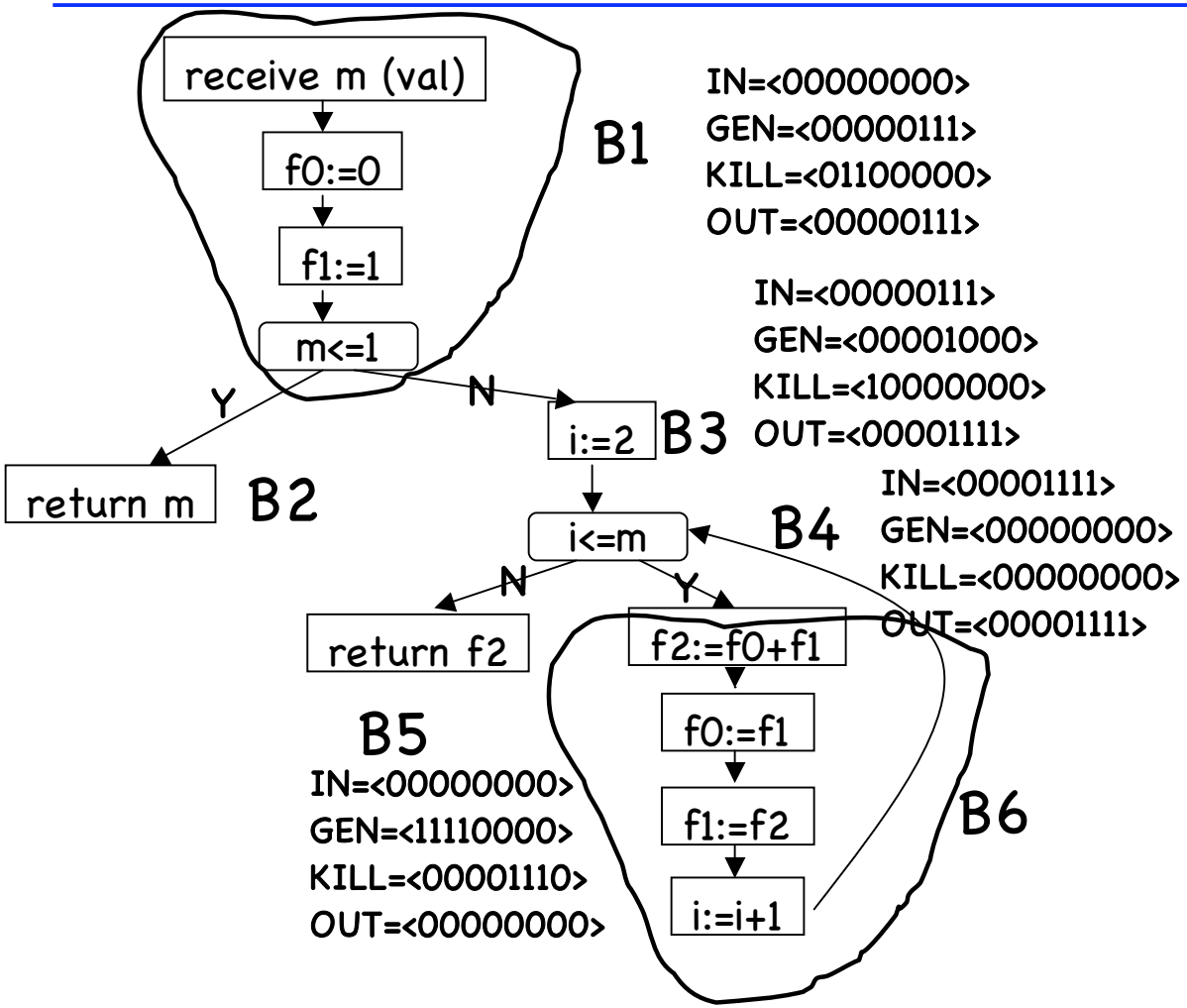
Bit vector representation

- Two control points per instruction: before and after
- One bit vector per control point
- One bit in vector per def
- 1 for def and 0 otherwise
- GEN represents def's *generated* by an instruction
- KILL represents def's *killed (assigned)* by an instruction

- IN(i) = | (out(j)), j is a pred of i
- OUT(i) = (IN(i) & ~KILL(i)) | GEN(i)

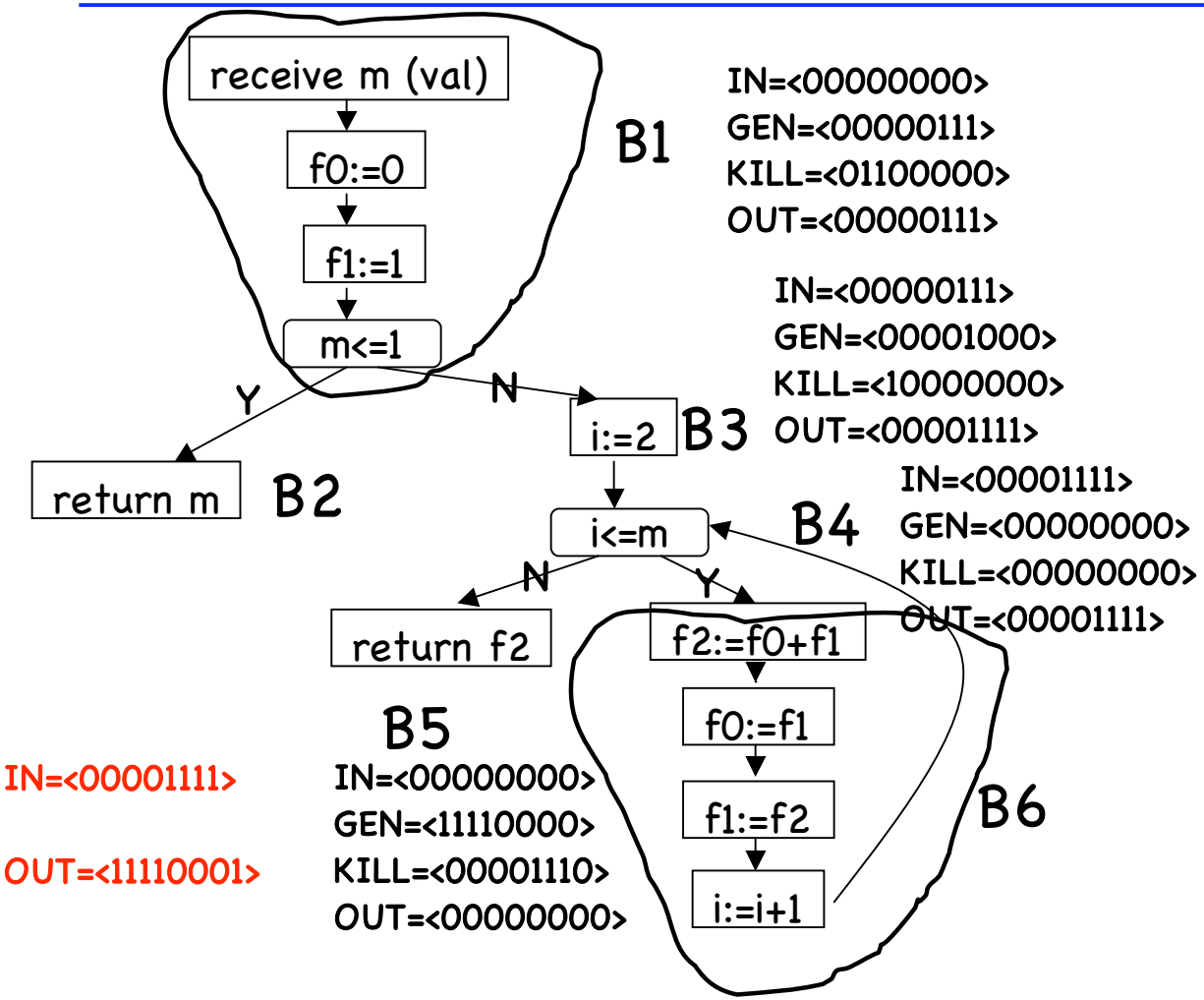
```
100   receive m (val)      (1)
101   f0 := 0              (2)
102   f1 := 1              (3)
103   if m<=1 goto L3
104   i := 2               (4)
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1        (5)
108       f0:=f1           (6)
109       f1:=f2           (7)
110       i:=i+1           (8)
111       goto L1
112   L3: return m
```

Simulation of algorithm



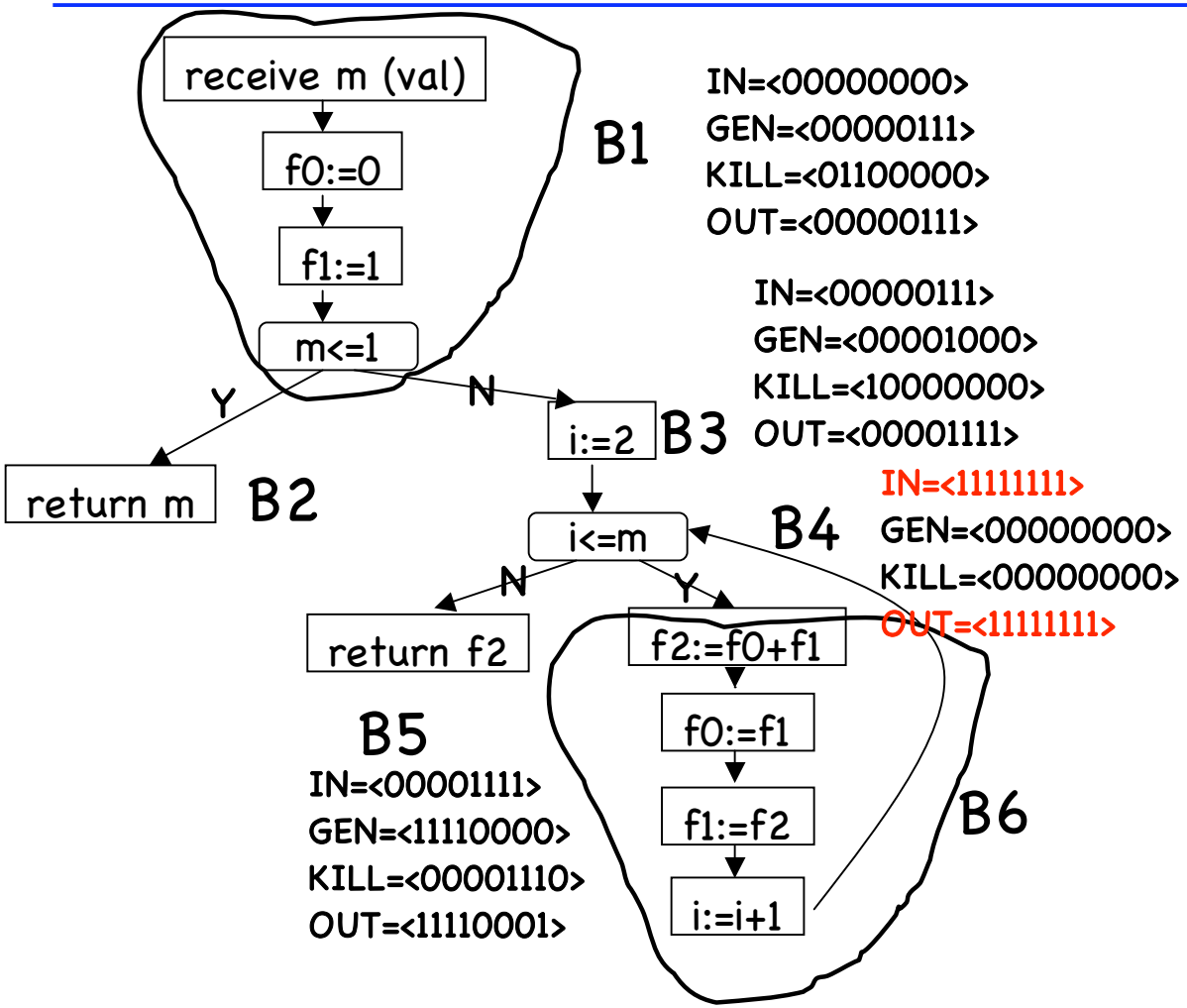

```
100   receive m (val)      (1)
101   f0 := 0              (2)
102   f1 := 1              (3)
103   if m<=1 goto L3
104   i := 2               (4)
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1        (5)
108       f0:=f1           (6)
109       f1:=f2           (7)
110       i:=i+1           (8)
111       goto L1
112   L3: return m
```

Simulation of algorithm



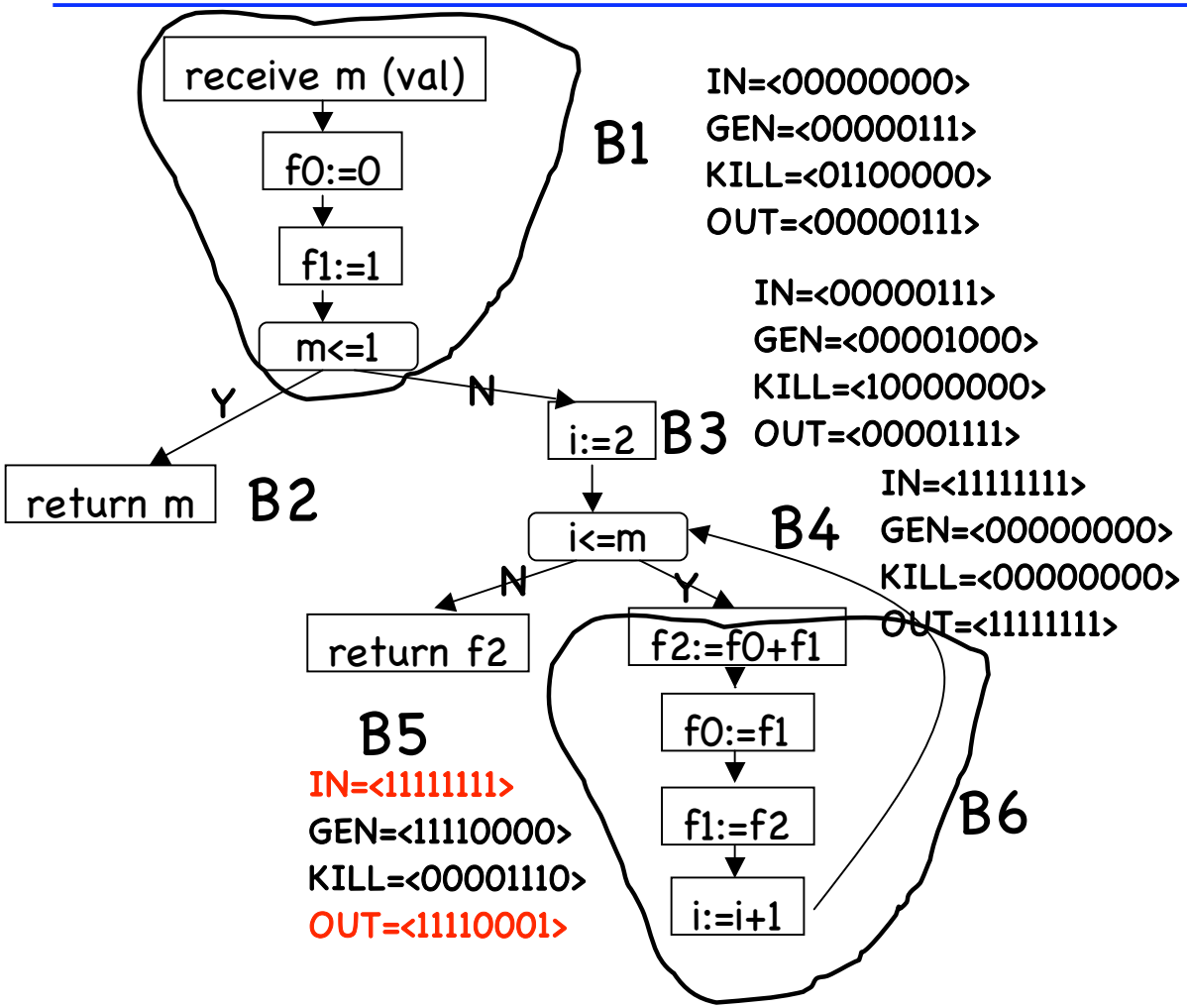

```
100   receive m (val)      (1)
101   f0 := 0              (2)
102   f1 := 1              (3)
103   if m<=1 goto L3
104   i := 2               (4)
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1        (5)
108       f0:=f1           (6)
109       f1:=f2           (7)
110       i:=i+1           (8)
111       goto L1
112   L3: return m
```

Simulation of algorithm



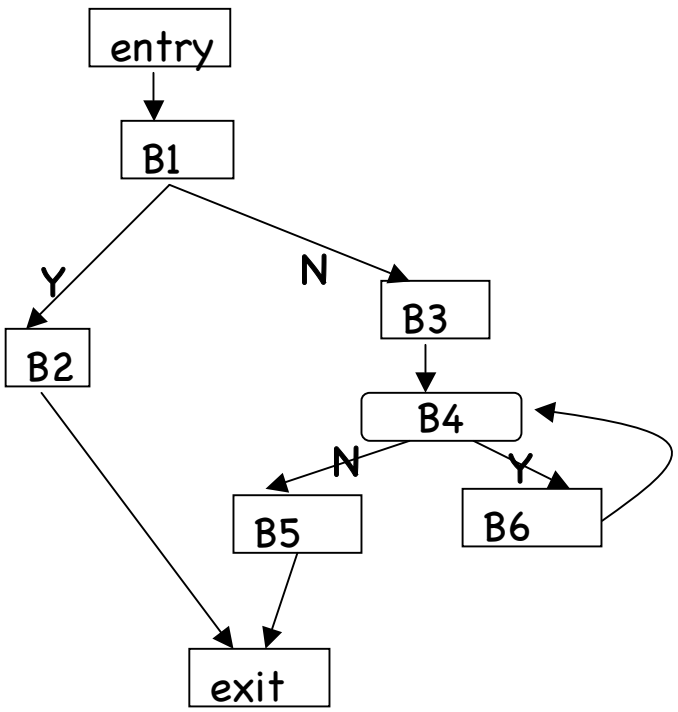

```
100   receive m (val)      (1)
101   f0 := 0              (2)
102   f1 := 1              (3)
103   if m<=1 goto L3
104   i := 2               (4)
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1        (5)
108       f0:=f1           (6)
109       f1:=f2           (7)
110       i:=i+1           (8)
111       goto L1
112   L3: return m
```

Simulation of algorithm




```
100   receive m (val)      (1)
101   f0 := 0              (2)
102   f1 := 1              (3)
103   if m<=1 goto L3
104   i := 2               (4)
105   L1: if i<=m goto L2
106       return f2
107   L2: f2:=f0+f1        (5)
108       f0:=f1           (6)
109       f1:=f2           (7)
110       i:=i+1           (8)
111       goto L1
112   L3: return m
```

Reverse post-order



Post order:

exit B2 B5 B6 B4 B3 B1 entry

Reverse post order:

entry B1 B3 B4 B6 B5 B2 exit

Key: order predecessors before a node

Comments

- This is called an 'iterative forward bit-vector problem'
- What for? See Xie / Engler paper (Free HW:)
- Note: control paths are taken irrespective of whether predicates that control branching along a path are satisfiable or not
- As a result, analysis may say that a def MAY reach a point but in fact at run-time it never does
- When is conservativeness necessary? `conservative', in this case, means to include ALL run-time reaching definitions even at the price of producing spurious ones.

Other data flow analyses

1. Available expressions
   if e is evaluated on every path from entry to a point p and none of e's variables is modified in between, then e is said to be available at p.
2. Live variables
   given a variable v and a point p, v is said to be live at p if from p to exit there is a path where v is used.
3. Upwards exposed uses
   for each definition of a var v, the set of points where v is used.
4. Copy-propagation analysis
5. Constant-propagation analysis
6. Partial-redundancy analysis

Other data flow analyses

1. Available expressions
2. Live variables
3. Upwards exposed uses
4. Copy-propagation analysis
5. Constant-propagation analysis
6. Partial-redundancy analysis

and more … but these are most important in optimization

Discussion: What for?

Characteristics of DFA

- Properties obtained: independent attributes or relations
- Direction of information flow: forward (in direction of program execution) or backward (opposite of program execution)
- Value domain & termination of algorithms

HW#6

Calculate reaching definitions for find(). Show the set of definitions reaching each point using the set notation, not the bit vector, as the latter is hard to read. ({1,2,3} means def's 1, 2 and 3 reach this point.)

Weight: 3%