Control Flow in Programs (Part II: Iterative and Procedural control)

Venkatesh Choppella

International Institute of Information Technology, Hyderabad

Outline

Iterative Control

Procedural Control

Conclusion

Topic

Iterative Control

Procedural Control

Conclusion

Iterative Control Flow

Iteration = Process of repetition. Iteration continues as long as a condition is satisfied.

```
0  i = read()
1  a = 1
2  while i > 0:
3      a = a * i
4      i = i - 1
5      continue
6  x = a
7  # end
```

1. A While statement ($L_2 - L_5$) has three parts:
   - The `while` keyword
   - A test expression (L_2)
   - A body block ($L_3 - L_5$)
2. In the concrete syntax, the body block is indented.
3. The body block ends with the `continue` keyword.

Structural Abstraction

Program

```
i = read()
a = 1
while i > 0:
    a = a * i
    i = i - 1
    continue
x = a
# end
```

Structural Abstraction

```
expression assignment
expression assignment
while:
    expression assignment
    expression assignment
    continue
expression assignment
# end
```

Control Transfer Functions

Structural Abstraction

```
expression assignment
expression assignment
while:
    expression assignment
    expression assignment
    continue
expression assignment
# end
```

Control Transfer Functions

Control Transfer Functions

Structural Abstraction

```
0
  expression assignment
1
  expression assignment
2
  while:
3
      expression assignment
4
      expression assignment
5
      continue
6
  expression assignment
7
  # end
```

Control Transfer Functions

i	next	true	false	error
0	1			7
1	2			7
2		3	6	7
3	4			7
4	5			7
5	2			
6	7			7
7				

Control Flow Graph

Control Transfer Functions

i	next	true	false	error
0	1			7
1	2			7
2		3	6	7
3	4			7
4	5			7
5	2			
6	7			7
7				

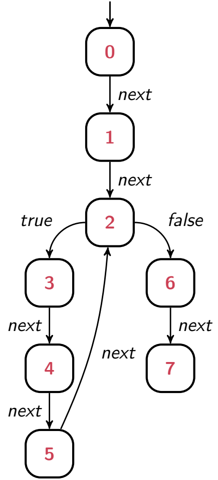
Control Flow Graph with Error edges implicit

Control Flow Graph

Control Transfer Functions

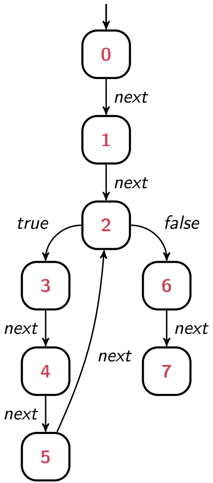
i	next	true	false	error
0	1			7
1	2			7
2		3	6	7
3	4			7
4	5			7
5	2			
6	7			7
7				

Control Flow Graph with error edges implicit



Structurally Feasible Executions

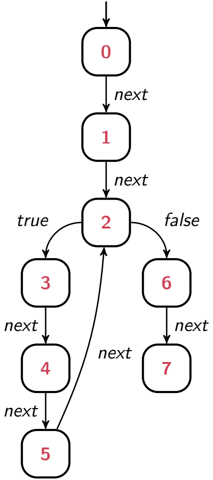
CFG:



Structurally Feasible Executions:

Structurally Feasible Executions

CFG:



Structurally Feasible Executions:

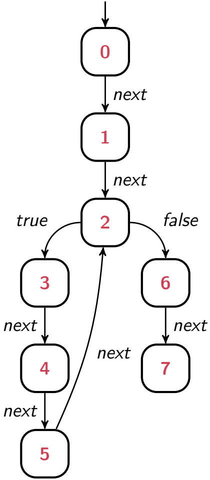
1. $L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2 \xrightarrow{true} L_3 \xrightarrow{next} L_4 \xrightarrow{next} L_5 \xrightarrow{next} L_2 \xrightarrow{false} L_6 \xrightarrow{next} L_7$
2. Executions with multiple such iterations of the body block including an infinite number of iterations
3. $L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2 \xrightarrow{false} L_6 \xrightarrow{next} L_7$
4. All error executions

Logically Feasible Executions

Program:

```
0
   i = read()
1
   a = 1
2
   while i > 0:
3
        a = a * i
4
        i = i - 1
5
        continue
6
   x = a
7
   # end
```

CFG:



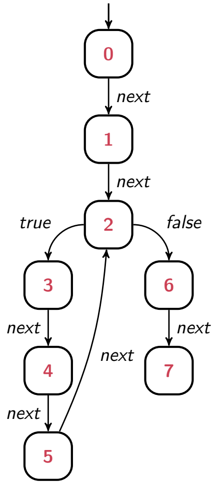
Logically Feasible Executions:

Logically Feasible Executions

Program:

```
0
  i = read()
1
  a = 1
2
  while i > 0:
3
      a = a * i
4
      i = i - 1
5
      continue
6
  x = a
7
  # end
```

CFG:



Logically Feasible Executions:

1. $i > 0$:

   $L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2 \xrightarrow{true} L_3 \xrightarrow{next}$
   $L_4 \xrightarrow{next} L_5 \xrightarrow{next} L_2 \xrightarrow{false} L_6 \xrightarrow{next} L_7$

   Multiple such possible executions based on the value of i

2. $i \leq 0$:

   $L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2 \xrightarrow{false} L_6 \xrightarrow{next} L_7$

3. i is not a number:

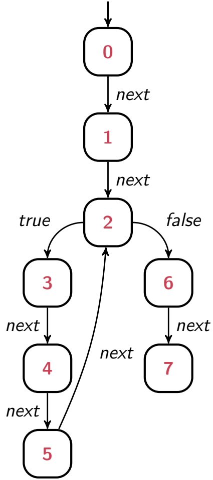
   $L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2 \xrightarrow{error} L_7$

Actual Execution

Program:

```
0
  i = read()
1
  a = 1
2
  while i > 0:
3
      a = a * i
4
      i = i - 1
5
      continue
6
  x = a
7
  # end
```

CFG:



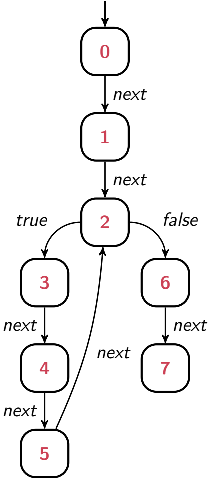
Actual execution given that value read at L_0 is the number 1.

Actual execution

Program:

```
0
  i = read()
1
  a = 1
2
  while i > 0:
3
      a = a * i
4
      i = i - 1
5
      continue
6
  x = a
7
  # end
```

CFG:



Actual execution, given that value read at L_0 is the number 1.

$$L_0 \xrightarrow{next} L_1 \xrightarrow{next} L_2 \xrightarrow{true} L_3 \xrightarrow{next}$$
$$L_4 \xrightarrow{next} L_5 \xrightarrow{next} L_2 \xrightarrow{false} L_6 \xrightarrow{next} L_7$$

Topic

Iterative Control

Procedural Control

Conclusion

Functions in Mathematics vs. Procedures in Programming

$$f : A \to B$$

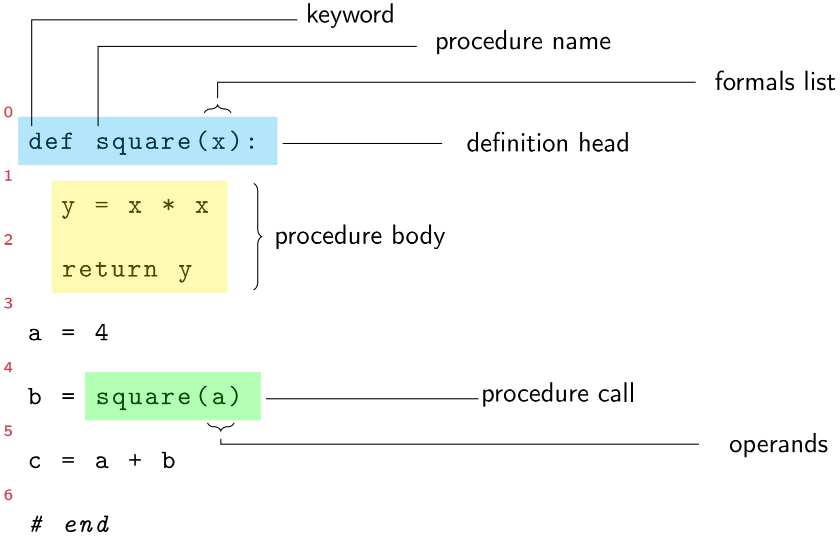
Mathematics

- For each $a : A$, $f(a)$
- returns a unique $b : B$.

Programming

- For each $a : A$, $f(a)$
  - returns a value $b : B$, or
  - causes an error to be raised, or
  - runs forever

Anatomy of a Procedure Definition and Call



```
def square(x):
    y = x * x
    return y
a = 4
b = square(a)
c = a + b
# end
```

Structural Abstraction

Program

```
def multiply(x, y):
    z = x * y
    return z
def square(m):
    n = multiply(m, m)
    return n
a = multiply(2, 3)
b = square(a)
# end
```

Structural Abstraction

```
def multiply:
    expression assignment
    return
def square:
    call multiply assignment
    return
call multiply assignment
call square assignment
# end
```

Control Transfer Functions

Structural Abstraction

```
def multiply:
    expression assignment
    return
def square:
    call multiply assignment
    return
call multiply assignment
call square assignment
# end
```

Control Transfer Functions

Control Transfer Functions

Structural Abstraction

```
0
  def multiply:
1
      expression assignment
2
      return
3
  def square:
4
      call multiply assignment
5
      return
6
  call multiply assignment
7
  call square assignment
8
  # end
```

Control Transfer Functions

i	next	call	return	error
0	3			
1	2			8
2			{5,7}	8
3	6			
4	5	1		8
5			{8}	8
6	7	1		8
7	8	4		8
8				

Control Flow Graph

Control Transfer Functions

i	next	call	return	error
0	3			
1	2			8
2			{5,7}	8
3	6			
4	5	1		8
5			{8}	8
6	7	1		8
7	8	4		8
8				

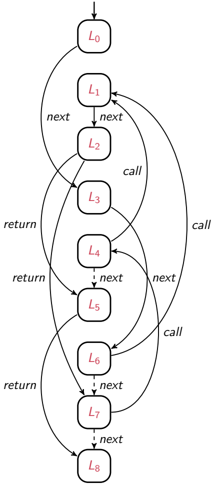
Control Flow Graph with Error edges implicit

Control Flow Graph

Control Transfer Functions

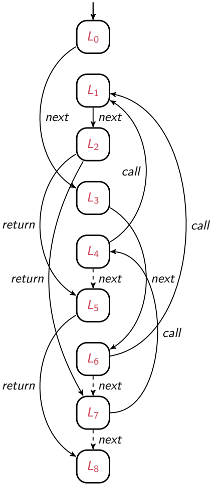
i	next	call	return	error
0	3			
1	2			8
2			{5,7}	8
3	6			
4	5	1		8
5			{8}	8
6	7	1		8
7	8	4		8
8				

Control Flow Graph with error edges implicit



Structurally Feasible Executions

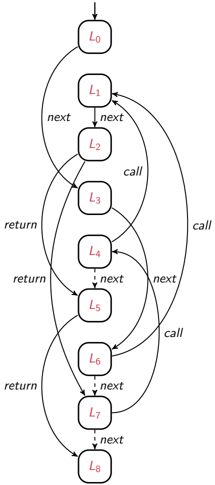
CFG:



Structurally Feasible Executions:

Structurally Feasible Executions

CFG:



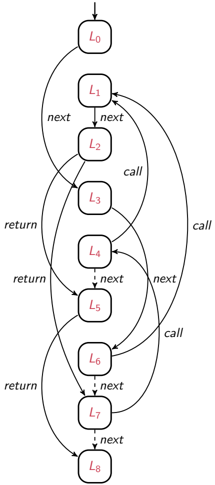
1. $$\begin{tikzcd} L_0 \arrow[r, "\mathit{next}"] & L_3 \arrow[r, "\mathit{next}"] & L_6 \arrow[rr, dashed, "\mathit{next}"] \arrow[d, "\mathit{call}"] & & L_7 \arrow[r, dashed, "\mathit{next}"] \arrow[d, "\mathit{call}"] & L_8 \\ & & L_1 \arrow[r, "\mathit{next}"] & L_2 \arrow[ur, "\mathit{return}"] & L_4 \arrow[r, dashed, "\mathit{next}"] \arrow[d, "\mathit{call}"] & L_5 \arrow[u, "\mathit{return}"'] \\ & & & & L_1 \arrow[r, "\mathit{next}"] & L_2 \arrow[u, "\mathit{return}"'] \end{tikzcd}$$

2. $$\begin{tikzcd} L_0 \arrow[r, "\mathit{next}"] & L_3 \arrow[r, "\mathit{next}"] & L_6 \arrow[d, "\mathit{call}"] & & L_8 \\ & & L_1 \arrow[r, "\mathit{next}"] & L_2 \arrow[ur, "\mathit{error}"] & \end{tikzcd}$$

3. All error executions.

More Structurally Feasible Executions

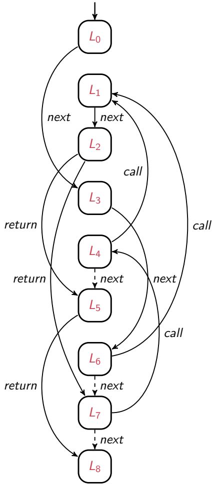
CFG:



1.
$$\begin{tikzcd}
L_0 \arrow[r, "next"] & L_3 \arrow[r, "next"] & L_6 \arrow[rr, dashed, "next"] \arrow[d, "call"] & & L_7 \arrow[d, "call"] & L_8 \\
& & L_1 \arrow[r, "next"] & L_2 \arrow[ur, "return"] & L_4 \arrow[d, "call"'] & \\
& & & & L_1 \arrow[uur, "error"'] &
\end{tikzcd}$$

Structurally Infeasible Executions

CFG:



1. $L_0 \xrightarrow{next} L_2 \xrightarrow{next} L_3 \xrightarrow{error} L_8$

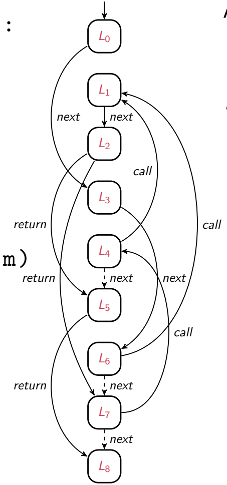
   (This is not a valid path in the CFG.)

2. $$\begin{tikzcd} L_0 \arrow[r, "next"] & L_3 \arrow[r, "next"] & L_6 \arrow[d, "call"] & & L_5 \arrow[r, "error"] & L_8 \\ & & L_1 \arrow[r, "next"] & L_2 \arrow[ur, "return"] & & \end{tikzcd}$$

   (Note this is a valid path in the CFG.)

Logically feasible and Actual Execution

```
0
  def multiply(x, y):
1
    z = x * y
2
    return z
3
  def square(m):
4
    n = multiply(m, m)
5
    return n
6
  a = multiply(2, 3)
7
  b = square(a)
8
  # end
```



Actual execution same as logically feasible execution

$$\begin{tikzcd}
L_0 \arrow[r, "next"] & L_3 \arrow[r, "next"] & L_6 \arrow[rr, dashed, "next"] \arrow[d, "call"] & & L_7 \arrow[r, dashed, "next"] \arrow[d, "call"] & L_8 \\
& & L_1 \arrow[r, "next"] & L_2 \arrow[ur, "return"] & L_4 \arrow[r, dashed, "next"] \arrow[d, "call"] & L_5 \arrow[u, "return"] \\
& & & & L_1 \arrow[r, "next"] & L_2 \arrow[u, "return"]
\end{tikzcd}$$

Topic

Iterative Control

Procedural Control

Conclusion

Iterative Control (`while`)

1. The **while** statement comprises the `while` instruction and a body, which is a block.
2. The `while` instruction has a test expression.
3. The body of a while loop ends with `continue`.
4. The control transfer functions at the while instruction are `true`, `false` and `error`

Simple Expressions

1. Expressions are simple (no procedure call subexpressions).

2. A procedure call occurs by itself on the right hand side of an assignment.

Procedure Definition

1. A procedure consists of a definition instruction and a body, which is a block.

2. The **next** of `def` skips the body.

Procedure call

1. A procedure call has **call**, **next** and **error** transfer functions.

2. The call transfers control to the first location in the body of the procedure.

Return instruction and structurally infeasible executions

1. **return** transfers control to the corresponding call's next location.

2. In the presence of procedure calls, not all labelled paths from the start of the program to the end are structurally feasible executions. This can happen if the return transfers control not to the corresponding call's next location but somewhere else.

The Big Picture: How does a program run?

1. We need a mental model to understand how a program runs.

2. We have seen how control flows in a program, but what about values of variables, storage and output?

Programs run . . . on a machine

1. To understand a computer language, we need to understand the underlying machine.

Machines and languages exist at multiple levels of abstraction

1. Hardware (Voltages)
2. Architecture (Microcode)
3. Processor (Instruction Set)
4. High level (Programming Language)
5. Domain level (Domain specific languages)

Thank you

`venkatesh.choppella@iiit.ac.in`