Stepwise Refinement
Top Down Design

On Top Down Design

- Useful in creating a function or algorithm when the input and output data structures correspond
  - » **If the input and output data structures do not correspond then one needs communicating processes to correctly design an implementation**

Program $\neq$ function

- **NOT USEFUL** for designing programs

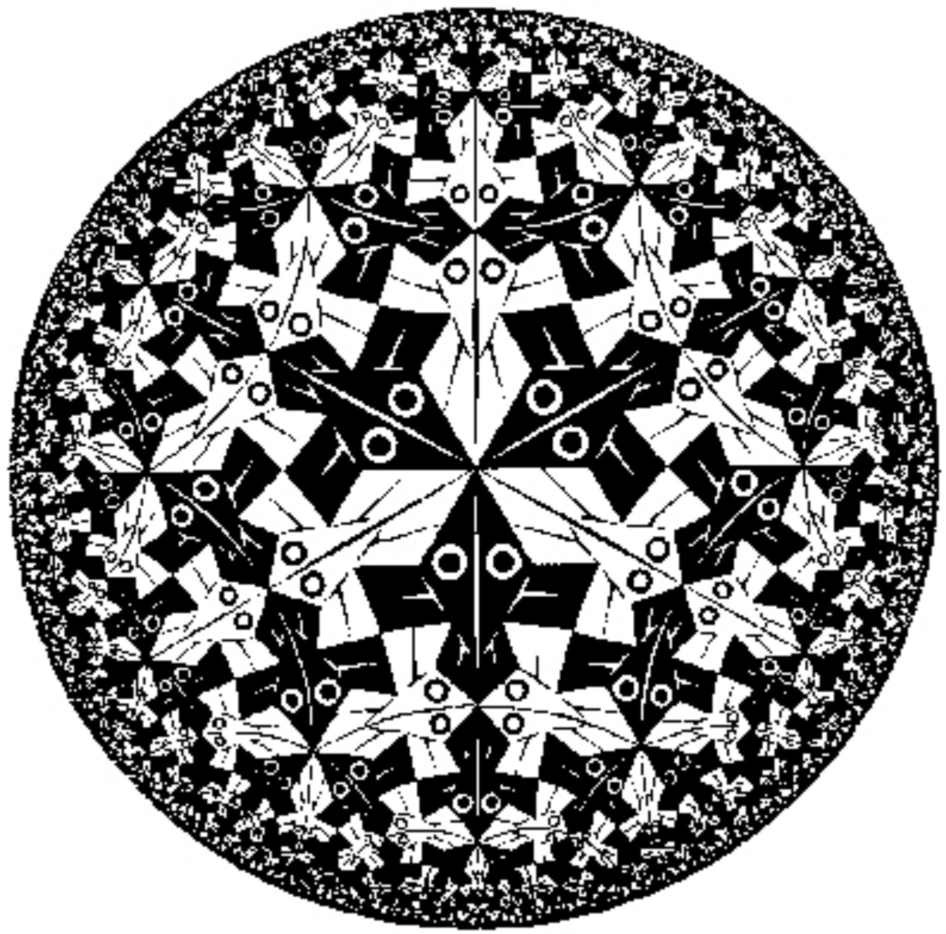
Real systems have no top

On Mathematics

I saw a high wall and as I had a premonition of an enigma, something that might be hidden behind the wall, I climbed over it with some difficulty . . . . On the other side I landed in a wilderness and had to cut my way through with a great effort until – by a circuitous route – I came to the open gate, the open gate of mathematics.

M.C. Escher

Escher – Circle Limit 1 (1958)



Escher – Plane Filling 1 (1951)

Escher Waterfall 1961

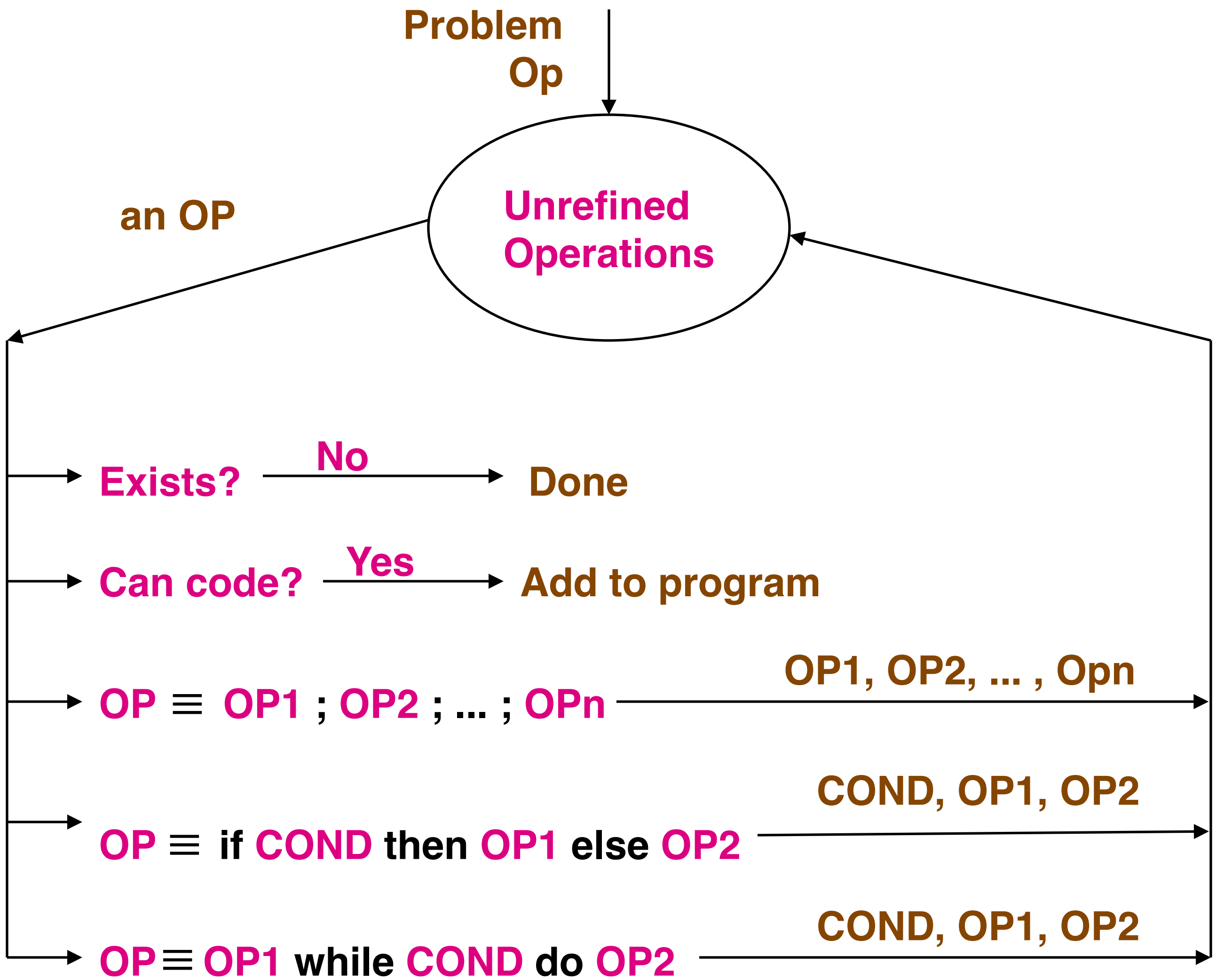
Stepwise Refinement

- Also known as **functional decomposition** and top down design
- Given an operation, there are only the following three choices for refinement
  - » **Sequence of sub-operations**
    - > **OP ≡ OP1 ; OP2 ; ... ; OPn**
  - » **Choice of sub-operations**
    - > **OP ≡ if COND then OP1 else OP2**
  - » **Loop over a sub-operation**
    - > **OP ≡ OP1 while COND do OP2**

Stepwise Refinement

- Is an recursive process of applying one of the previous three choices (with variations) to sub-operations until program text can be written

Stepwise Refinement Procedure



Sequence Questions

$$OP \equiv OP1\ ;\ OP2\ ;\ \ldots\ ;\ Opn$$

Does the sequence of operations **OP1** followed by **OP2** followed by ... followed by **OPn** accomplish the upper level operation **OP**

precondition OP $\longrightarrow$ precondition OP1

postcondition OP1 $\longrightarrow$ precondition OP2

postcondition OP2 $\longrightarrow$ precondition OP3

...

postcondition OPn-1 $\longrightarrow$ precondition OPn

postcondition Opn $\longrightarrow$ postcondition OP

Choice Questions

OP $\equiv$ if COND then OP1 else OP2

- Does the operation **OP1** accomplish the operation **OP** when the condition **COND** is **true**

  COND $\rightarrow$

  precondition OP $\rightarrow$ **precondition OP1**

  and postcondition OP1 $\rightarrow$ **postcondition OP**

- Does the operation **OP2** accomplish the operation **OP** when the condition **COND** is **false**

  not COND $\rightarrow$

  precondition OP $\rightarrow$ **precondition OP2**

  and postcondition OP2 $\rightarrow$ **postcondition OP**

Loop Questions – 1 of 4

OP ≡ –0–

OP1

–1–

while COND { OP2 –2– }

–3–

Ask verification question - i -

Let **LI** be a loop invariant, which must always be true after **OP1** is executed – except temporarily within **OP2**

Loop Questions – 2 of 4

Question 0 – What is the **LI**?

- » **In general it is an extremely difficult question to answer. It contains the essential difficulty in programming**
- » **Fundamentally it is the following**

$$\textbf{LI} \equiv \textbf{totalWork} = \textbf{workToDo} + \textbf{workDone}$$

Loop Questions – 3 of 4

OP $\equiv$ –0–
OP1
–1–
while COND { OP2 –2– }
–3–

Question 1 – Is **LI** true after OP1?

precondition(OP) + execution(OP1) $\rightarrow$ LI

Question 2 – Is **LI** true after OP2?

(LI_{before} $\wedge$ COND) + execution(OP2) $\rightarrow$ LI_{after}

Loop Questions – 4 of 4

OP ≡ –0–

OP1

–1–

while COND { OP2 –2– }

–3–

Question 3a – Does the loop terminate?

Does COND eventually become false?

Question 3b – Is postcondition of **OP** true at loop end?

(LI $\wedge$ (not COND)) $\rightarrow$ postcondition OP

Example Loop Design

- Consider a program loop which calculates the division of positive integers.
  - » **D is the divisor and D > 0**
    Q is the quotient
    R is the remainder
    DV is the dividend and DV > 0

$$\begin{array}{r|l} & \overline{\quad Q \quad} \\ D & DV \\ & \ldots \\ & R \end{array}$$

- We are to compute **Q** and **R** from **D** and **DV** such that the following is true.

$$0 \leq R < D \;\wedge DV = D * Q \;+\; R$$

Loop Design – 1

- Question 0 – Find the loop invariant
  - » **After consulting an oracle we have determined that the following is an appropriate loop invariant**
    - > **This is the creative part of programming**

$$LI \equiv DV = D * Q + R \wedge R \geq 0$$

Loop Design – 2

OP ≡ –0–
OP1
–1–
while COND { OP2 –2– }
–3–

LI ≡ $DV = D * Q + R \wedge R \geq 0$

- What we have to do is to determine **COND**, **OP1**, and **OP2** while checking that the verification questions are satisfied
  - » **In practice we iterate between loop invariant and the program until we have a match that solves the problem**

Loop Design – 3

$$LI \equiv DV = D * Q + R \wedge R \geq 0$$

- Question 1 – Make **LI** true at the start

OP1 $\equiv Q \leftarrow 0 \; ; \; R \leftarrow DV$

LI is true

$DV = D * 0 + DV$

$DV > 0$ **from the precondition** $\rightarrow R \geq 0$

Loop Design – 4

$$LI \equiv DV = D * Q + R \wedge R \geq 0$$

while COND { OP2 –2– }

- Question 2 – Is **LI** still true after **OP2** is executed?

  COND $\equiv R \geq D$ **True before OP2 exec**

  OP2 $\equiv Q \leftarrow Q + 1 \;;\; R \leftarrow R - D$

  Therefore $Q' = Q + 1 \wedge R' = R - D$

  » **After OP2 show LI first part is true**

  > $DV = D * Q' + R'$ **LI first part**
  > $= D * (Q + 1) + (R - D)$ **Substitute equality**
  > $= D * Q + D + R - D$ **Rearrange**
  > $= D * Q + R$ **True before OP2, still true**

  » See effect of moving data from **workToDo (D & DV)** to **workDone (Q & R)** while maintaining the invariant.

Loop Design – 5

$$LI \equiv DV = D * Q + R \wedge R \geq 0$$

while COND { OP2 –2– }

- Question 2 – Is **LI** still true after **OP2** is executed?

  COND $\equiv R \geq D$ **True before OP2 exec**

  OP2 $\equiv Q \leftarrow Q + 1 \; ; \; R \leftarrow R - D$

  Therefore $Q' = Q + 1 \wedge R' = R - D$

  » After **OP2** show second part of **LI** is still true

  > $R' \geq 0$ **LI second part**
  >
  > $\rightarrow (R - D) \geq 0$ **Substitute equality**
  >
  > $\rightarrow R \geq D$ **Rearrangement is true from COND**
  >
  > **Therefore** $R' \geq 0$ **is true**

Loop Design – 6

$$LI \equiv DV = D * Q + R \wedge R \geq 0$$

while R ≥ D **{**
    Q ← Q + 1
    R ← R – D
}

- Question 3a – Does **COND** eventually become false?
  - » **Every time around the loop OP2 reduces the size of R by D > 0.**
  - » **In a finite number of iterations R must become less than D.**

Loop Design – 7

$$LI \equiv DV = D * Q + R \wedge R \geq 0$$

$$COND = R \geq D$$

- Question 3b

  Does $\sim COND$ and $LI \rightarrow$ postcondition for OP ?

  - $\sim COND \rightarrow R < D$
  - $LI \equiv DV = D * Q + R \wedge R \geq 0$
  - Together $\rightarrow DV = D * Q + R \wedge 0 \leq R < D$
  - Equals **Problem specification**

    $0 \leq R < D \wedge DV = D*Q + R$

Loop Invariant – Example 1a

- Copy a sequence of characters from input to output

```
read aChar from input
while aChar ≠ EOF
   write aChar to output
   read aChar from input
end while
```

- The loop invariant is the following

$$\frac{\text{In}[\ 1\ ..\ N\ ]\ =\ \text{Out}[\ 1\ ..\ j-1\ ]\ +\ \text{aChar} + \text{In}\ [\ j+1\ ..\ N\ ]}{\text{totalWork}\ =\ \text{workDone}\ +\ \text{workToDo}}$$

Loop Invariant – Example 1b

- The loop invariant is the following

  In[1 .. N] = Out[1 .. j – 1] + aChar + In [j + 1 .. N]

- The loop invariant can be simplified by removing **Input[i + 1 .. N]** from each side of the relationship

  In[1 .. j] = Out[1 .. j – 1] + aChar

- It is the simplified form that one sees most often

Loop Invariant – Example 2a

- Compute the sum of the integers 1 to N

  sum ← 0 ; p ← 0

  loop exit when p = N

  p += 1 ; sum += p

  end loop

- The loop invariant is the following

$$\underbrace{\sum_{1}^{n} j}_{\text{totalWork}} = \underbrace{\text{sum}}_{\text{workDone}} + \underbrace{\sum_{p+1}^{n} j}_{\text{workToDo}}$$

Loop Invariant – Example 2b

- The loop invariant is the following

$$\sum_{1}^{n} j \ = \ \textbf{sum} \ + \sum_{p+1}^{n} j$$

- Simplify by removing the following expression from each side of the relationship

$$\sum_{p+1}^{n} j$$

To get

$$\sum_{1}^{p} j \ = \ \text{sum}$$

Loop Invariant – Example 3a

- Compare string **A [1 .. p]** with string **B [1 .. p]**.
  Last character in string must be **EOS**

J ← 1
loop exit when A[j] ≠ B[j] or A[j] = EOS
j += 1
end loop

A[1 .. p] ? B[1 .. p] **totalWork**

= A[1 .. j – 1] = B[1 .. j – 1] **workDone**

+ A[j .. n] ? B[j .. n] **workToDo**

∧ j ≤ p ∧ A[p] = B[p] = EOS

Support conditions

Loop Invariant – Example 3b

- The loop invariant is the following.

$$A[1..p] ? B[1..p]$$

$$= A[1..j-1] = B[1..j-1]$$

$$+ A[j..p] ? B[j..p]$$

$$\wedge\ j \leq p\ \wedge\ A[p] = B[p] = EOS$$

- The simplified loop invariant

$$A[1..j-1] = B[1..j-1]$$

$$\wedge\ j \leq p\ \wedge\ A[p] = B[p] = EOS$$

Context for loop inductive step

Inductive step

LI before loop – old values

Backward chain proof

Loop body

```
k++
a[k] := k^2
```

Forward chain proof

LI after loop – new values

Loop question 2

$$(LI_{before} \wedge COND) + execution(OP2) \Rightarrow LI_{after}$$