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 ≠ function

- **NOT USEFUL** for designing programs

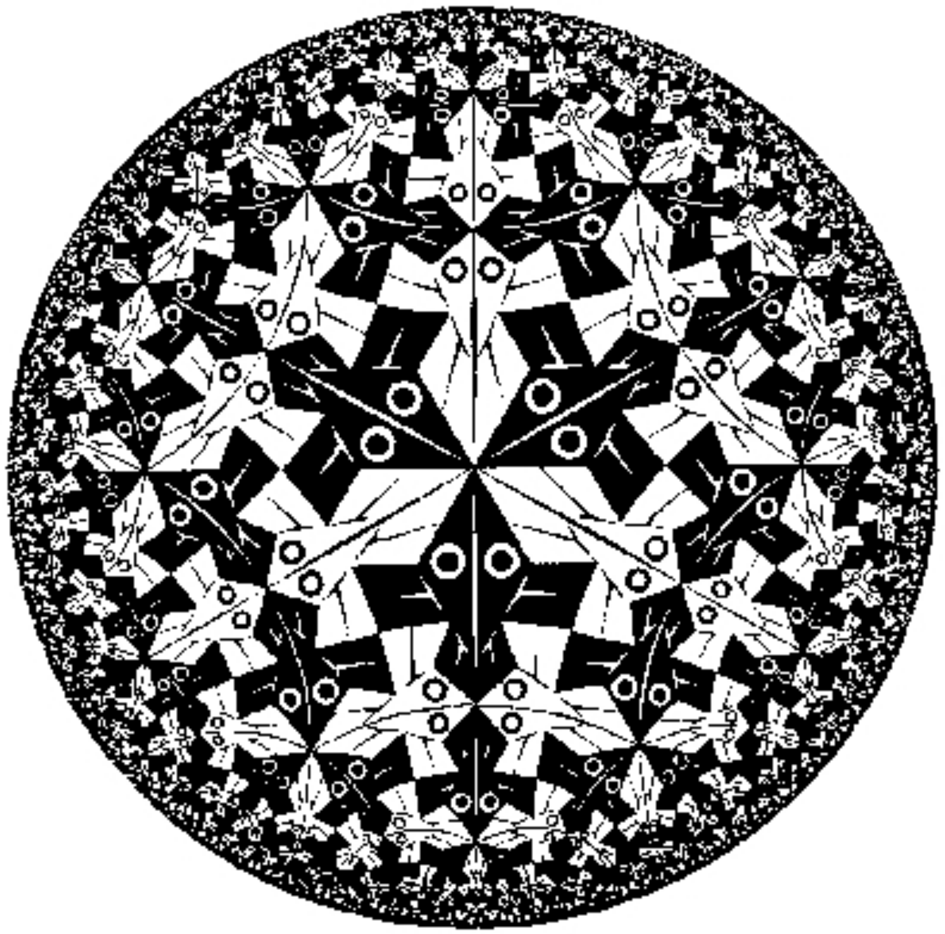
Real systems have no top

On Mathematics

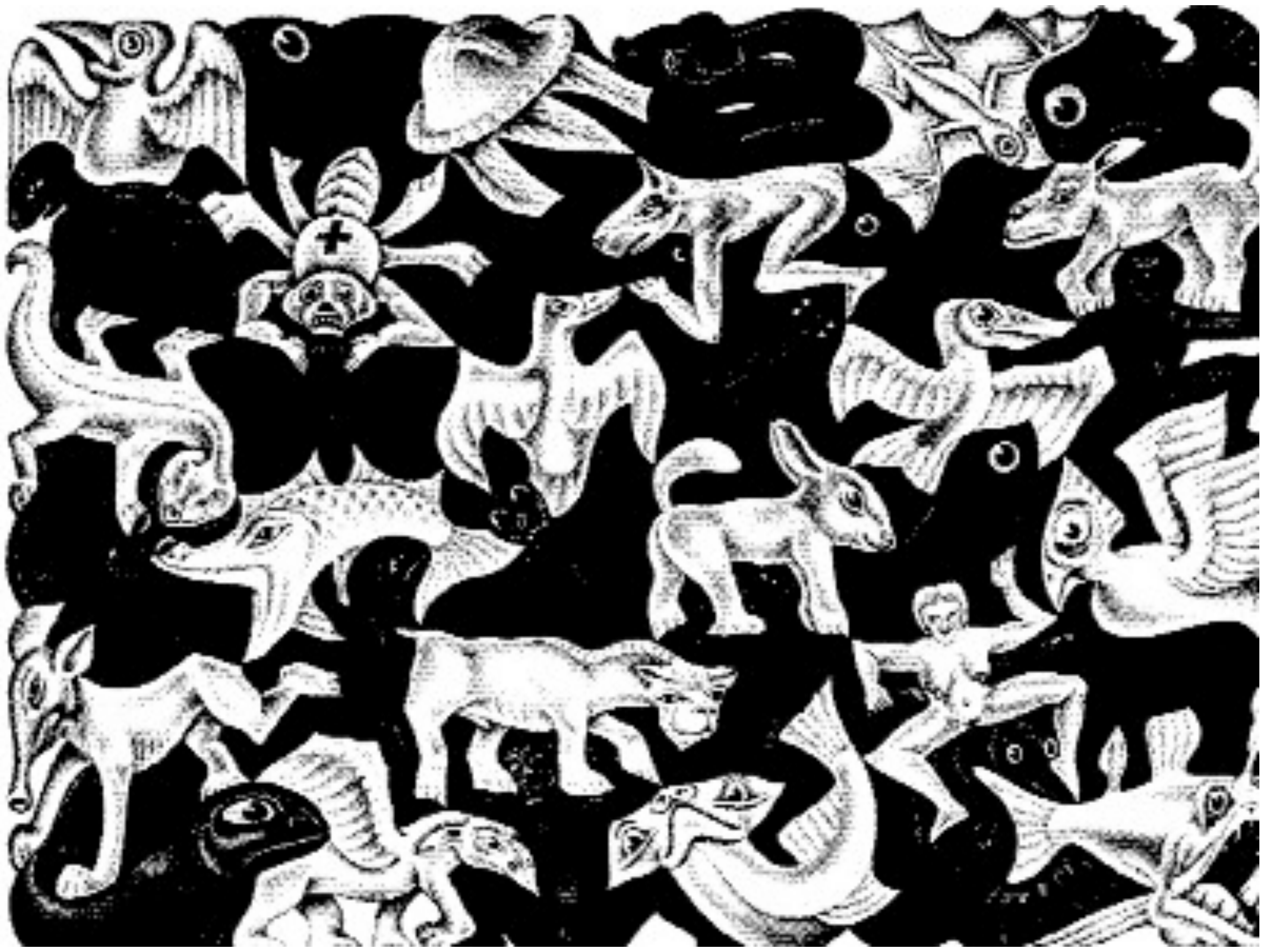
I saw a high wall and as I had a premonition of an enigma, something that might be hidden behind the wall, I climbed over it with some difficulty . . . . On the other side I landed in a wilderness and had to cut my way through with a great effort until – by a circuitous route – I came to the open gate,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

M.C. Escher

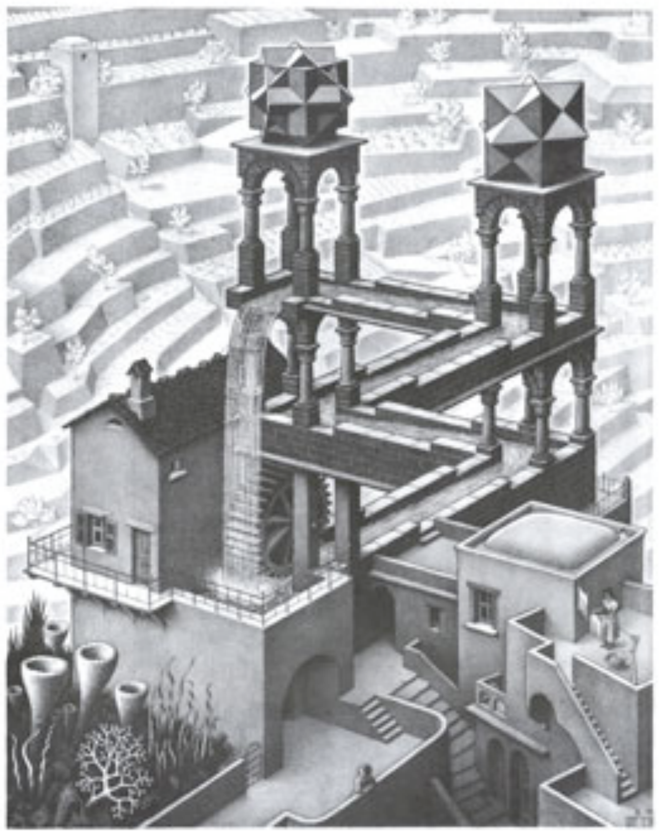
Escher – Circle Limit 1 (1958)



Escher – Plane Filling 1 (1951)



Escher Waterfall 1961



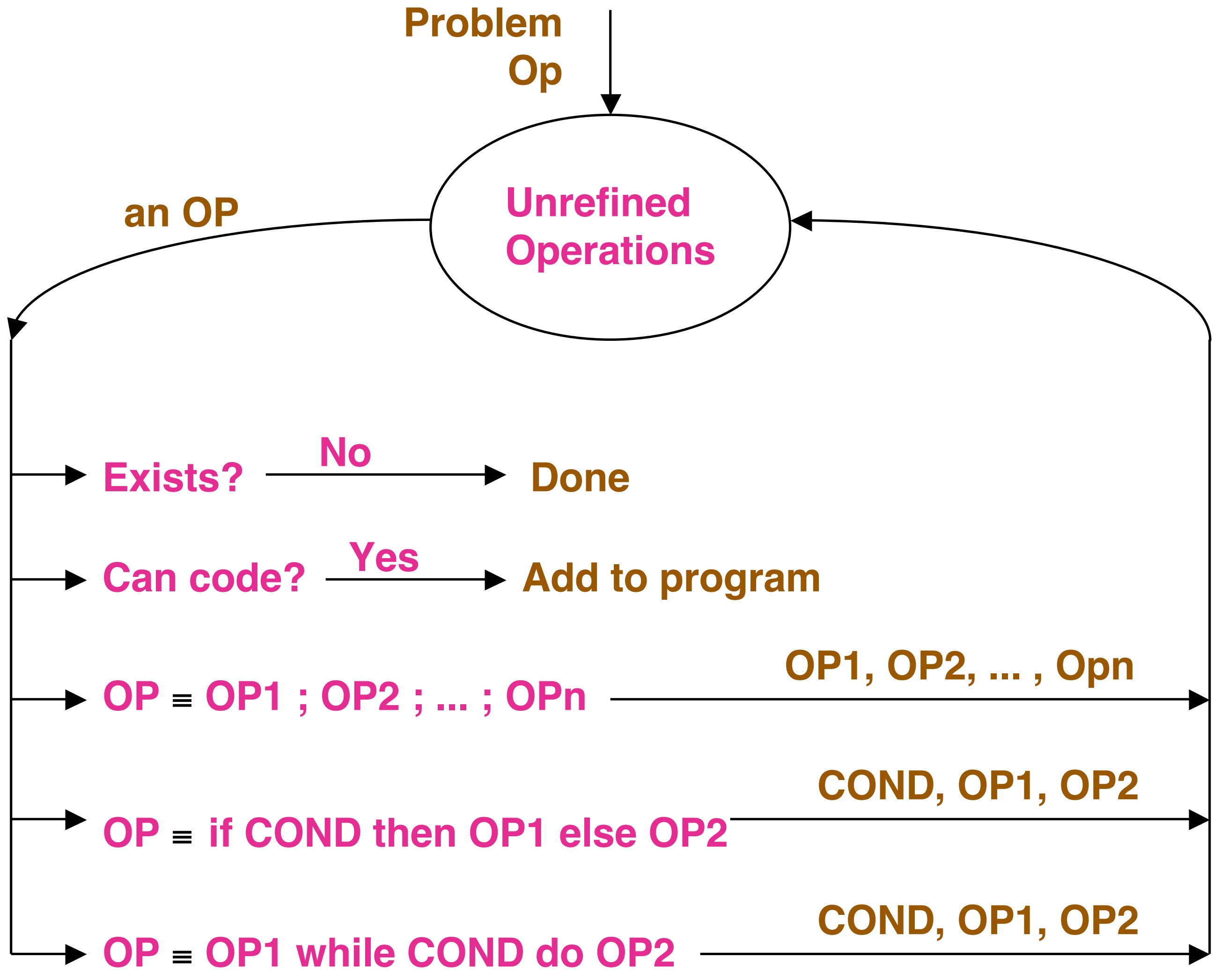
Stepwise Refinement

- 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 ≡ OP1 ; OP2 ; ... ; Opn

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

precondition OP ⇒ precondition OP1

postcondition OP1 ⇒ precondition OP2

postcondition OP2 ⇒ precondition OP3

...

postcondition OPn-1 ⇒ precondition OPn

postcondition OPn ⇒ postcondition OP

Choice Questions

OP ≡ if COND then OP1 else OP2

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

  COND ⇒

  precondition OP ⇒ precondition OP1

  and postcondition OP1 ⇒ postcondition OP

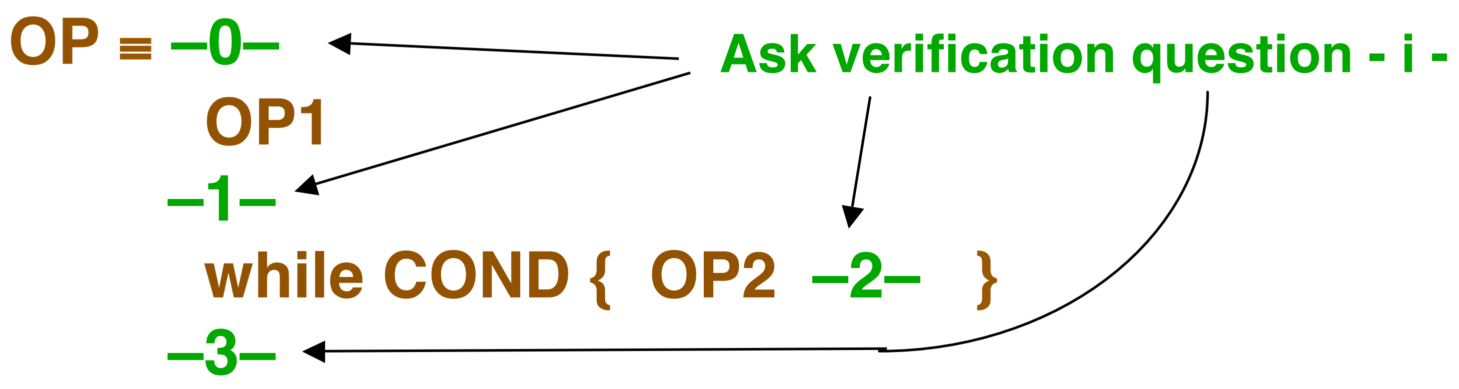
- Does the operation **OP2** accomplish the operation **OP** when the condition **COND** is **false**

  not COND ⇒

  precondition OP ⇒ precondition OP2

  and postcondition OP2 ⇒ postcondition OP

Loop Questions – 1 of 4



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**

LI ≡ totalWork = workToDo + workDone

Loop Questions – 3 of 4

```
OP ≡  –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 $\wedge$ COND) + execution(OP2) $\Rightarrow$ LI

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 ∧ (not COND)) ⇒ 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

$$\begin{array}{r|l} & Q \\ \hline D & DV \\ & \dots \\ & R \end{array}$$

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

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

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–                    LI ≡ DV = D*Q + R ∧ R ≥ 0
       OP1
     –1–
       while COND {  OP2  –2–   }
     –3–
```

- 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

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

- 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, So 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 $Q' = Q + 1$ & $R' = R - D$

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

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

Loop Design – 6

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

```
while $R \geq D$ {
    $Q \leftarrow Q + 1$
    $R \leftarrow R - D$
}
```

- 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

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

COND = $R \geq D$

- Question 3b

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

  » $\sim$ COND $\Rightarrow R < D$

  » LI $\Rightarrow DV = D*Q + R$ & $R \geq 0$

  » Together $\Rightarrow DV = D*Q + R$ & $0 \leq R < D$

  » Equals Problem spec
  $0 \leq R < D$ & $DV = D*Q + R$

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

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

Loop Invariant – Example 1b

- The loop invariant is the following

  In[1 .. N] = Out[1.. i - 1] + aChar + In [i + 1 .. N]

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

  In[1 .. i] = Out[1 .. i - 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

$$\underline{\Sigma_{0}^{n}\, i} = \underline{\text{sum}} + \underline{\Sigma_{p+i}^{n}\, j}$$

totalWork = workDone + workToDo

Loop Invariant – Example 2b

- The loop invariant is the following

$$\sum_{0}^{n} i = sum + \sum_{p+1}^{n} i$$

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

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

To get

$$\sum_{0}^{p} i = sum$$

Loop Invariant – Example 3a

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

```
i ← 1
loop exit when A[i] ≠ B[i] or A[i] = EOS
    i += 1
end loop
```

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

= A[1 .. i -1] = B[1 .. i -1] workDone

+ A[i .. n] ? B[i .. n] workToDo

& $i \le p$ & A[p] = B[p] = EOS

Support conditions

Loop Invariant – Example 3b

- The loop invariant is the following.

$$A[\,1\,..\,p\,]\ ?\ B[\,1\,..\,p\,]$$
$$=\ A[\,1\,..\,i\,\text{-}1\,] = B[\,1\,..\,i\,\text{-}1\,]$$
$$+\ A[\,i\,..\,n\,]\ ?\ B[\,i\,..\,n\,]$$
$$\&\ \ i \le p\ \ \&\ \ A[p] = B[p] = \text{EOS}$$

- The simplified loop invariant

$$A[\,1\,..\,i\,\text{-}1\,] = B[\,1\,..\,i\,\text{-}1\,]$$
$$\&\ \ i \le p\ \ \&\ \ A[p] = B[p] = \text{EOS}$$

On Correspondence

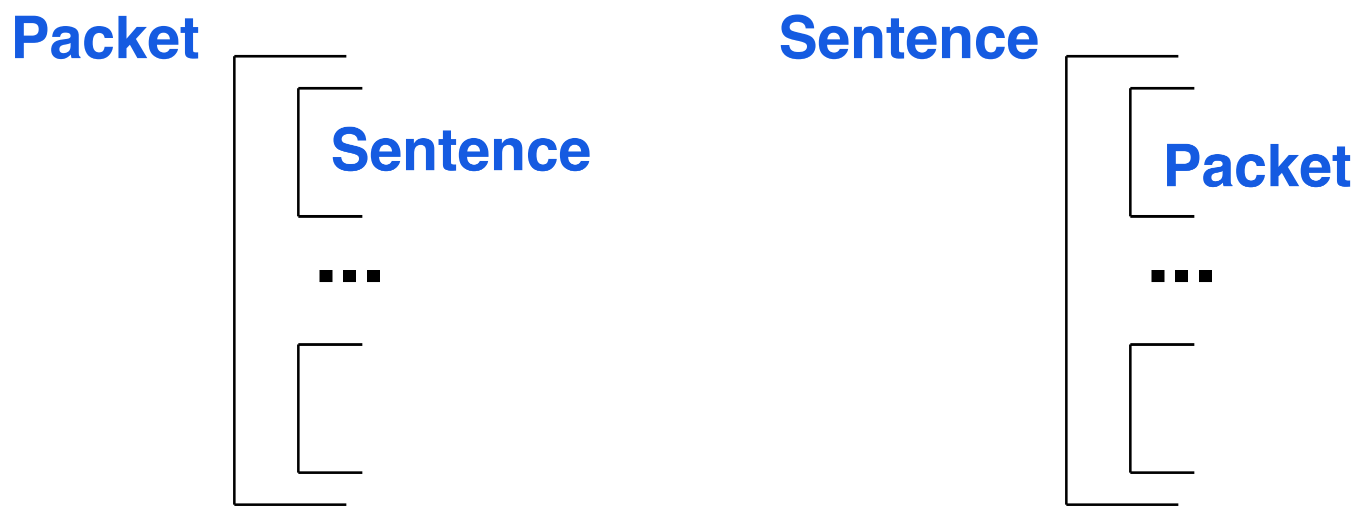
- Algorithm **input** and **output** can frequently be described with **regular expressions** – consisting of sequence, choice and loops over data elements
- Data structures **correspond** when the same loop structure can be used to describe both structures
  - **including loop conditions**
- Data structures do not correspond when their loop structures do not nest within each other or loop conditions are different

Packet & Sentence Example – 1

- Consider a sequence of email packets sent over the network
- Information within the packets is a sequence of sentences
- Loop over packets does not correspond with loop over sentences and vice versa

Packet & Sentence Example – 2

- Sentences span packet boundaries
  - » Do not have an integral number of sentences within every packet
  - » Do not have have an integral number of packets within every sentence



Packet & Sentence Example – 3

- Using the **Direct Mapping Rule** you should be able to point to the program text, draw a box and say
  - » **One packet corresponds to this box**
    - > **No more and no less**
  - » **One sentence corresponds to this box**
    - > **No more and no less**
- In modelling both sentences and packets it is necessary to have explicit loops for each or else you violate the Direct Mapping Rule