Functions

Ordered Pairs

We have seen elements, `a`, and sets of elements `{a,b}`. Adding an ordering relation creates a lattice of ordered functions. Each function is specified by a collection of ordered pairs, `(a,b)`.

Example:

The logical function `(if a then b)` is defined by a collection of three ordered pairs of the form `(a,b)`, where the values of `a,b` are in the set `{0,1}`:

```
if a then b =def= {(0,0),(0,1),(1,1)}
```

The sixteen different ways of collecting the four possible ordered pairs, `N` at a time, `N=0..4`, define the sixteen different Boolean functions of two variables.

Functions and Relations

relation: `xRy isTrue` *function:* `f(x)=y isTrue`

The set of all first values of a set of ordered pairs is called the ***Domain***.

The set of all second values of a set of ordered pairs is called the ***Range***.

A ***relation*** is a collection of ordered pairs over two sets, the domain set and the range set.

A ***function*** is a relation `(x,f(x))`, such that

1. Every member of the domain is associated with a member of the range, and

2. No element in the domain is associated with more than one element in the range.

Perspectives on Functions

1. *Formal constraints on a relation*

existence: `all x inDomain . exists y inRange`

uniqueness: `all pairs (x,f(x)) . if x1=x2 then f(x1)=f(x2)`

2. *Graph*

Domain on x-axis, Range on y-axis
uniqueness permits the graph to cross any vertical line (i.e. x-value) *only once*.

3. *Lookup table*

x	f(x)
1	1
2	4
3	9

4. *Static relation between variables*

```
x = y + 5
```
"=" is an equivalence relation

5. *Dynamic relation between variables*

```
f(x) = y
```
x is the independent variable (controlled measurement)
y is the dependent variable (observed measurement)

6. *Pure operation*

```
(lambda (#)  #^2 + # + 1)
```

is the formal parameter of the function which binds to any value

7. *Sequence of combinators*

```
fac = (Y) lf.ln.(((0)n)1) ((*)n) (f) (-1) n
```

A tree of substitution instructions

8. *Rule of correspondence/algorithm*

take a number	x
double it	2*x
add 3	2*x + 3

9. *Set transformation*

```
Domain                Range
  a   ----------->   b
  b   ----------->   c
  c   ----------->   d
  d   ----------->   d
```

10. *Input-output machine*

```
      x
 \        /
  \     /
    |   |
    |   |

    f(x)
```

11. *Way of finding and assigning names to unnamed objects*

    `2^100` is the short name of a large number

12. *Digraph*

```
(1) ---> (3) ---> (5)
```

Types of Functions

Surjective, Onto, Epic	all `y` inRange, exists `x` inDomain . `f(x) = y`
Injective, 1-to-1, Monic	`if f(x1) = f(x2) then x1 = x2`
Bijective	1-to-1 and Onto

Bijective functions have an **inverse**, since every element in both the Domain and the Range are in correspondence:

two-way existence	`all x inD, exists y inR . f(x) = y` `all y inR, exists x inD . f(x) = y`
two-way uniqueness	`all (x,f(x)) . x1 = x2 iff f(x1) = f(x2)`
inverse:	Exists `f-inverse` iff `f` is onto and one-to-one

Special Functions

Identity

```
f(x) = x
```

Characteristic

```
f(x)  =  1  if x inA
      =  0  if x not inA
```

Permutations

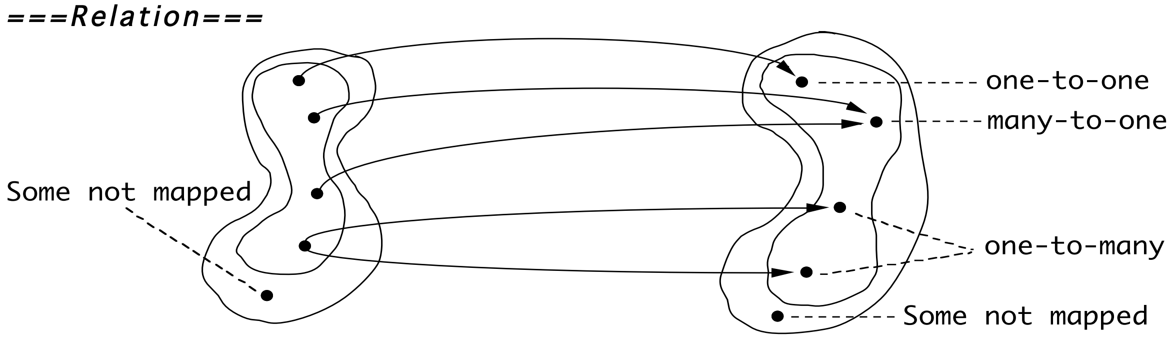
```
(1,2,3)  <-->  (3,1,2)  <-->  (2,3,1)
```

Sequences

```
1 .. n  <-->  1/1 .. 1/n
```

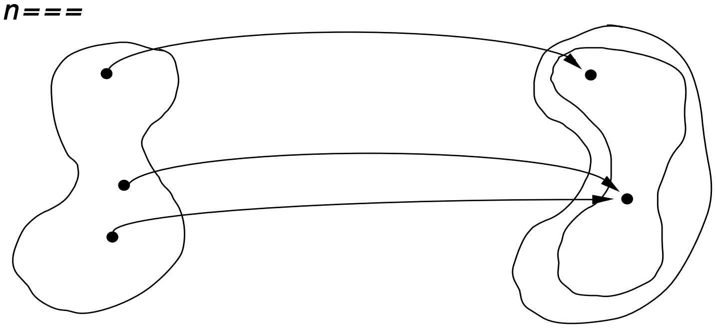
Mappings

===Relation===



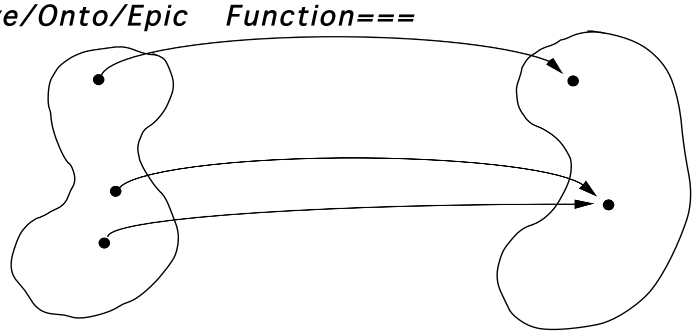
===Function===

EXISTENCE
=def=
all mapped



UNIQUENESS
=def=
no one-to-many

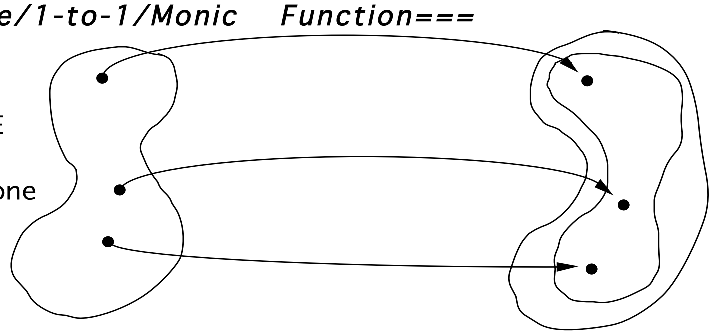
===Surjective/Onto/Epic Function===



ONTO
=def=
all mapped

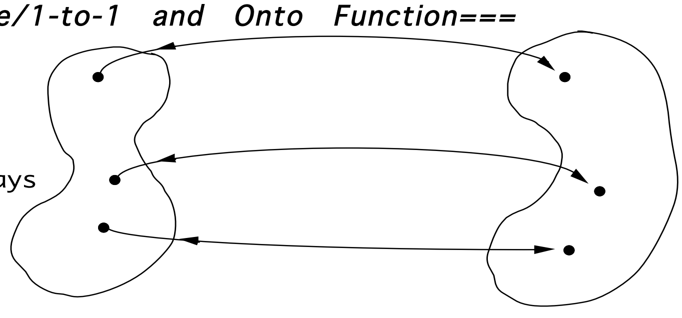
===Injective/1-to-1/Monic Function===

ONE-TO-ONE
=def=
no many-to-one



===Bijective/1-to-1 and Onto Function===

INVERSE
=def=
maps both ways



ONE-To-ONE and ONTO
=def=
no many-to-one and
all mapped in both
Domain and Range

Function Composition

`(f o g) =` All pairs `(x,z)` Exists `y` such that `(x,y)` in `g` and `(y,z)` in `f`
Note that the Range of g is a subset of the Domain of f

`(f o g)(x) = f( g(x) )`

Associative: `(f o g) o h = f o (g o h)`

Not commutative: `f o g =/= g o f`

Maintains the type of the function:

if `f` and `g` are functions, then `(f o g)` is a function
if `f` and `g` are onto, then `(f o g)` is onto
if `f` and `g` are one-to-one, then `(f o g)` is one-to-one

Composition of a function with its inverse:

`f o f-inverse = identity I` on Range of `f`
`f-inverse o f = identity I` on Domain of `f`

Inverse of a composition: `(f o g)-inverse = g-inverse o f-inverse`

Binary Functions

Binary functions are a mapping of ordered pairs onto elements: `((a,b) c)`
e.g.: `a + b = c` `+ = {((a,b),c) such that (a,b) in S X S and c inS}`

The domain consists of ordered pairs rather than single elements.

If `a`,`b`, and `c` are in the Domain,
then the Domain is closed with regard to the function:

`All x1,x2 inD such that f(x1,x2) inD`