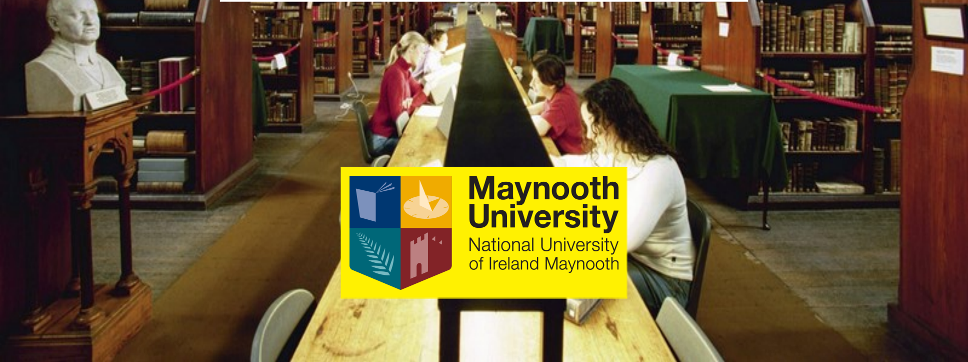
CS172: Computer Systems II

Lecture 20

Functions

- classifying

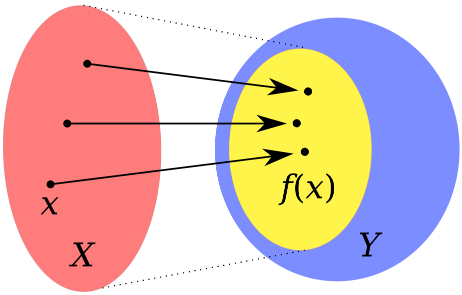
James Power



Domain, codomain and image: definition

Domain, codomain and image: definition

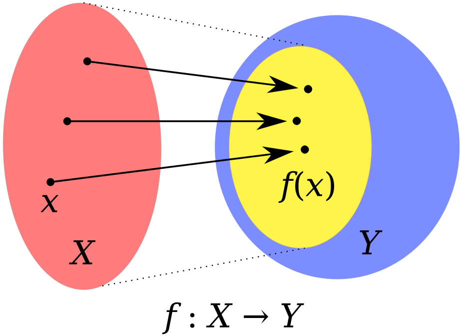
Suppose we are given some sets X and Y and a function $f \subseteq X \times Y$.



$$f : X \to Y$$

Domain, codomain and image: definition

Suppose we are given some sets X and Y and a function $f \subseteq X \times Y$.

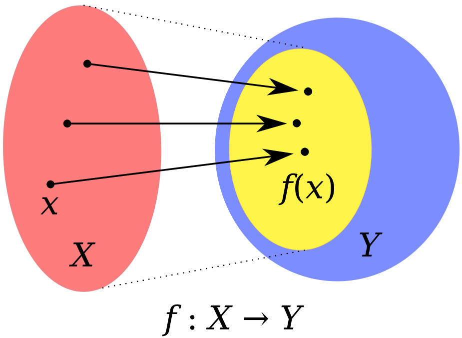


- The set X is called the domain of the function f

Image from Wikipedia

Domain, codomain and image: definition

Suppose we are given some sets X and Y and a function $f \subseteq X \times Y$.

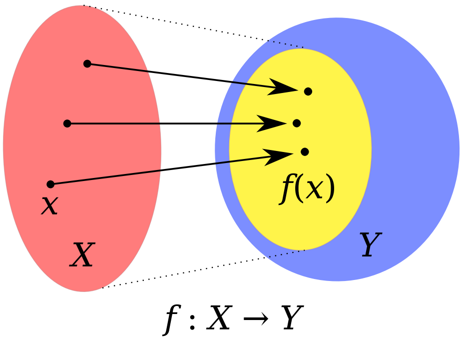


- The set X is called the domain of the function f
- The set Y is called the codomain (or range) of the function f

Image from Wikipedia

Domain, codomain and image: definition

Suppose we are given some sets X and Y and a function $f \subseteq X \times Y$.



- The set X is called the domain of the function f
- The set Y is called the codomain (or range) of the function f
- The subset of the codomain Y containing elements mapped-to by f is called the image of the function f

Image from Wikipedia

Domain, codomain and image: example

For example, consider the two functions $sq, db \subseteq \mathbb{Z} \times \mathbb{Z}$ defined as

- $sq \triangleq (\lambda\, x \in \mathbb{Z} \cdot x^2)$
- $db \triangleq (\lambda\, x \in \mathbb{Z} \cdot 2 * x)$

Domain, codomain and image: example

For example, consider the two functions $sq, db \subseteq \mathbb{Z} \times \mathbb{Z}$ defined as

- $sq \triangleq (\lambda\, x \in \mathbb{Z} \cdot x^2)$
- $db \triangleq (\lambda\, x \in \mathbb{Z} \cdot 2 * x)$

In each case the domain and codomain is the set $\mathbb{Z}$, but

- The image of sq is the set $$\begin{aligned} & \{n \in \mathbb{N} \mid n^2\} \\ = \; & \{0, 1, 4, 9, 16, \ldots\} \end{aligned}$$
- The image of db is the set $$\begin{aligned} & \{n \in \mathbb{Z} \mid 2 * n\} \\ = \; & \{\ldots, -6, -4, -2, 0, 2, 4, 6, \ldots\} \end{aligned}$$

Injective and Surjective Functions

Injective and Surjective Functions

For any sets sets X and Y and a function $f \subseteq X \times Y$, we say that the function f is:

- injective (one-to-one) if each element in the image is mapped to by *just one* element in the domain; that is, if

$$(\forall x, y \cdot (x \in X \land y \in X \land f(x) = f(y)) \rightarrow x = y)$$

Injective and Surjective Functions

For any sets sets X and Y and a function $f \subseteq X \times Y$, we say that the function f is:

- injective (one-to-one) if each element in the image is mapped to by *just one* element in the domain; that is, if

$$(\forall x, y \cdot (x \in X \wedge y \in X \wedge f(x) = f(y)) \rightarrow x = y)$$

Thus the *inverse* of an injective function is always a function (from the image to the domain).

Injective and Surjective Functions

For any sets sets X and Y and a function $f \subseteq X \times Y$, we say that the function f is:

- injective (one-to-one) if each element in the image is mapped to by *just one* element in the domain; that is, if

$$(\forall x, y \cdot (x \in X \land y \in X \land f(x) = f(y)) \rightarrow x = y)$$

- surjective (onto) if each element in the codomain is mapped to by *at least one* element in the domain; that is, if

$$(\forall z \cdot z \in Y \rightarrow (\exists x \cdot x \in X \land f(x) = z))$$

Injective and Surjective Functions

For any sets sets X and Y and a function $f \subseteq X \times Y$, we say that the function f is:

- injective (one-to-one) if each element in the image is mapped to by *just one* element in the domain; that is, if

$$(\forall x, y \cdot (x \in X \wedge y \in X \wedge f(x) = f(y)) \rightarrow x = y)$$

- surjective (onto) if each element in the codomain is mapped to by *at least one* element in the domain; that is, if

$$(\forall z \cdot z \in Y \rightarrow (\exists x \cdot x \in X \wedge f(x) = z))$$

For a surjective function the image is the *whole* codomain.

Injective and Surjective Functions

For any sets sets X and Y and a function $f \subseteq X \times Y$, we say that the function f is:

- injective (one-to-one) if each element in the image is mapped to by *just one* element in the domain; that is, if

$$(\forall x, y \cdot (x \in X \land y \in X \land f(x) = f(y)) \rightarrow x = y)$$

- surjective (onto) if each element in the codomain is mapped to by *at least one* element in the domain; that is, if

$$(\forall z \cdot z \in Y \rightarrow (\exists x \cdot x \in X \land f(x) = z))$$

- bijective if it is both injective and surjective.

Injective and Surjective Functions

For any sets sets X and Y and a function $f \subseteq X \times Y$, we say that the function f is:

- injective (one-to-one) if each element in the image is mapped to by *just one* element in the domain; that is, if

$$(\forall x, y \cdot (x \in X \land y \in X \land f(x) = f(y)) \to x = y)$$

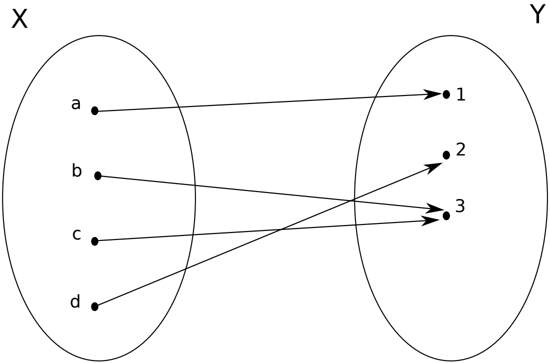
- surjective (onto) if each element in the codomain is mapped to by *at least one* element in the domain; that is, if

$$(\forall z \cdot z \in Y \to (\exists x \cdot x \in X \land f(x) = z))$$

- bijective if it is both injective and surjective.

Thus if there is a bijection between two sets, then those sets have the same number of elements.

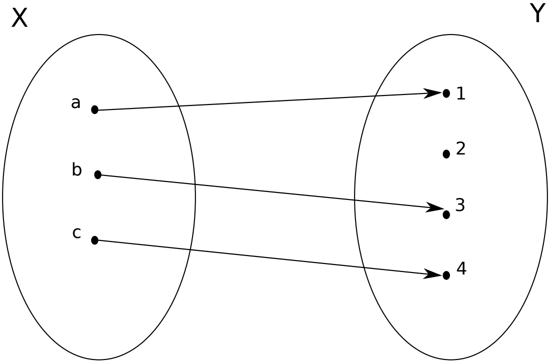
Example: Surjective but not injective



Injective? ✗

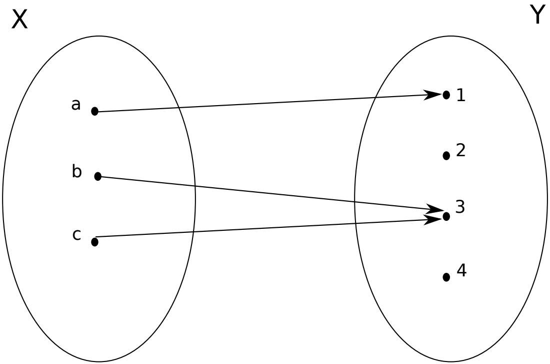
Surjective? ✓

Example: Injective but not surjective



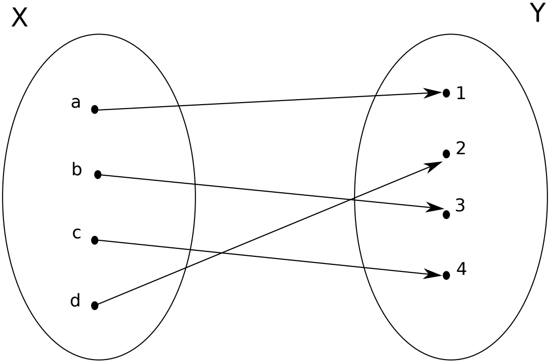
Injective? ✓
Surjective? ✗

Example: Neither injective nor surjective



Injective? ✗
Surjective? ✗

Example: Both injective and surjective (so, bijective)



Injective? ✓

Surjective? ✓

Injective and Surjective: Java example 1

Consider the function from the variable names {a,b,c} to the objects on the heap created by the following Java code:

```
Person a = new Person("Tom");
Person b = new Person("Dick");
Person c = new Person("Harry");
```

Injective and Surjective: Java example 1

Consider the function from the variable names {a,b,c} to the objects on the heap created by the following Java code:

```
Person a = new Person("Tom");
Person b = new Person("Dick");
Person c = new Person("Harry");
```

- There is a bijective function from the variable names {a,b,c} to the three objects on the heap.

Injective and Surjective: Java example 2

Consider the function from the variable names {a,b,c} to the objects on the heap created by the following Java code:

```
Person a = new Person("Tom");
Person b = new Person("Dick");
Person c = b;
```

Injective and Surjective: Java example 2

Consider the function from the variable names {a,b,c} to the objects on the heap created by the following Java code:

```
Person a = new Person("Tom");
Person b = new Person("Dick");
Person c = b;
```

- There is now a surjective (but not injective) function from the variable names {a,b,c} to the *two* objects on the heap.

Injective and Surjective: Java example 3

Consider the function from the variable names {a,b,c} to the objects on the heap created by the following Java code:

```
Person a = new Person("Tom");
Person b = new Person("Dick");
Person c = new Person("Harry");
c = new Person("Fred");
```

Injective and Surjective: Java example 3

Consider the function from the variable names {a,b,c} to the objects on the heap created by the following Java code:

```
Person a = new Person("Tom");
Person b = new Person("Dick");
Person c = new Person("Harry");
c = new Person("Fred");
```

- There is an injective (but not surjective) function from the variable names {a,b,c} to the *four* objects on the heap.

Injective and Surjective: Java example 3

Consider the function from the variable names {a,b,c} to the objects on the heap created by the following Java code:

```
Person a = new Person("Tom");
Person b = new Person("Dick");
Person c = new Person("Harry");
c = new Person("Fred");
```

- There is an injective (but not surjective) function from the variable names {a,b,c} to the *four* objects on the heap.
- Calling the *garbage collector* would make this function surjective again.

Injective: Hash Tables Example

- Hash tables are a data structure used to map keys to values.
- Like an array, except the index (key) can be of any type.

  Sometimes called an *associative array* (or, if the keys are strings, a *dictionary*).
- To implement this, you need a hash function that maps the key type to an integer.

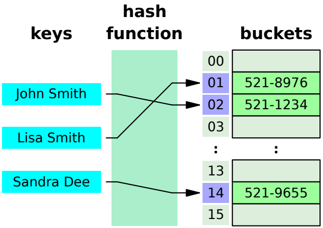


Image from Wikipedia

Injective: Hash Tables Example

- Hash tables are a data structure used to map keys to values.
- Like an array, except the index (key) can be of any type.

  Sometimes called an *associative array* (or, if the keys are strings, a *dictionary*).
- To implement this, you need a hash function that maps the key type to an integer.

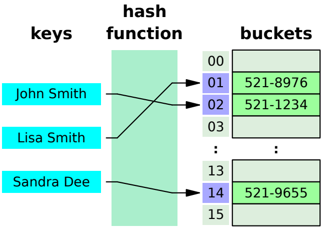


Image from Wikipedia

- Ideally, the hash function would be *injective*.
- When it's not injective, you get collisions.

Total vs. Partial Functions

Total vs. Partial Functions

- Typically in mathematics we assume that a function maps *all* the elements in the domain to an element in the codomain.

Total vs. Partial Functions

- Typically in mathematics we assume that a function maps *all* the elements in the domain to an element in the codomain.
- In certain cases we may want to allow *some* elements not to be mapped: this is known as a partial function. For example:

$$\mathit{reciprocal} \triangleq (\lambda\, a \in \mathbb{Z} \mid a \neq 0 \cdot 1/a)$$

- We can always make a partial function into a total function by restricting the domain to just those elements that are mapped from.

$$(\lambda\, a \in (\mathbb{Z} \setminus \{0\}) \cdot 1/a)$$

Total vs. Partial Functions

- Typically in mathematics we assume that a function maps *all* the elements in the domain to an element in the codomain.
- In certain cases we may want to allow *some* elements not to be mapped: this is known as a partial function. For example:

$$\textit{reciprocal} \triangleq (\lambda\, a \in \mathbb{Z} \mid a \neq 0 \cdot 1/a)$$

- We can always make a partial function into a total function by restricting the domain to just those elements that are mapped from.

$$(\lambda\, a \in (\mathbb{Z} \setminus \{0\}) \cdot 1/a)$$

- assuming we have some way of deciding which elements of the domain it won't work for...

Partial functions in CS

- *Partial* functions are interesting in Computer Science where we might have to allow for a method to be non-halting (e.g. because of an infinite loop).

```
int fact(int n) {
  if (n==0) return 1;
  else return n * fact(n-1);
}
```

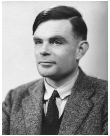
- What is `fact(-1)` ?

Partial functions in CS

- *Partial* functions are interesting in Computer Science where we might have to allow for a method to be non-halting (e.g. because of an infinite loop).

```
int fact(int n) {
  if (n==0) return 1;
  else return n * fact(n-1);
}
```

- What is `fact(-1)` ?



Theorem: It is not possible to write a code-analyser that will detect in advance whether *any* function is total or partial (i.e. whether it will halt or not).

- *Halting Problem*, Alan Turing, 1936

Example: classifying functions

Consider the set

$$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$$

Example: classifying functions

Consider the set

$$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$$

There are 16 possible *subsets* of this set,
thus 16 possible relations whose domain and codomain are the set $\{0,1\}$.

Example: classifying functions

Consider the set

$$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$$

There are 16 possible *subsets* of this set,
thus 16 possible relations whose domain and codomain are the set $\{0,1\}$.

1	relation of size 0
4	relations of size 1
6	relations of size 2
4	relations of size 3
1	relation of size 4

Example: classifying functions

Consider the set

$$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$$

There are 16 possible *subsets* of this set,
thus 16 possible relations whose domain and codomain are the set $\{0,1\}$.

1	relation of size 0	- is a (partial) function
4	relations of size 1	- all are (partial) functions
6	relations of size 2	
4	relations of size 3	
1	relation of size 4	

Example: classifying functions

Consider the set

$$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$$

There are 16 possible *subsets* of this set,
thus 16 possible relations whose domain and codomain are the set $\{0,1\}$.

1	relation of size 0	- is a (partial) function
4	relations of size 1	- all are (partial) functions
6	relations of size 2	
4	relations of size 3	- none are functions
1	relation of size 4	- not a function

Example: classifying functions

Consider the set

$$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$$

There are 16 possible *subsets* of this set,
thus 16 possible relations whose domain and codomain are the set $\{0,1\}$.

1	relation of size 0	- is a (partial) function
4	relations of size 1	- all are (partial) functions
6	relations of size 2	- only 4 are functions
4	relations of size 3	- none are functions
1	relation of size 4	- not a function

Example: classifying functions (continued)

Consider the set $\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$.
There are 9 *functions* whose domain and codomain are the set $\{0,1\}$.

Function	Total	Injective	Surjective	Bijective
$\{\}$				
$\{(0,0)\}$				
$\{(0,1)\}$				
$\{(1,0)\}$				
$\{(1,1)\}$				
$\{(0,0),(1,0)\}$				
$\{(0,0),(1,1)\}$				
$\{(0,1),(1,0)\}$				
$\{(0,1),(1,1)\}$				

Example: classifying functions (continued)

Consider the set $\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$.
There are 9 *functions* whose domain and codomain are the set $\{0,1\}$.

Function	Total	Injective	Surjective	Bijective
$\{\}$	✗	✓	✗	✗
$\{(0,0)\}$				
$\{(0,1)\}$				
$\{(1,0)\}$				
$\{(1,1)\}$				
$\{(0,0),(1,0)\}$				
$\{(0,0),(1,1)\}$				
$\{(0,1),(1,0)\}$				
$\{(0,1),(1,1)\}$				

Example: classifying functions (continued)

Consider the set $\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$.
There are 9 *functions* whose domain and codomain are the set $\{0,1\}$.

Function	Total	Injective	Surjective	Bijective
$\{\}$	✗	✓	✗	✗
$\{(0,0)\}$	✗	✓	✗	✗
$\{(0,1)\}$	✗	✓	✗	✗
$\{(1,0)\}$	✗	✓	✗	✗
$\{(1,1)\}$	✗	✓	✗	✗
$\{(0,0),(1,0)\}$				
$\{(0,0),(1,1)\}$				
$\{(0,1),(1,0)\}$				
$\{(0,1),(1,1)\}$				

Example: classifying functions (continued)

Consider the set $\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$.
There are 9 *functions* whose domain and codomain are the set $\{0,1\}$.

Function	Total	Injective	Surjective	Bijective
$\{\}$	✗	✓	✗	✗
$\{(0,0)\}$	✗	✓	✗	✗
$\{(0,1)\}$	✗	✓	✗	✗
$\{(1,0)\}$	✗	✓	✗	✗
$\{(1,1)\}$	✗	✓	✗	✗
$\{(0,0),(1,0)\}$	✓	✗	✗	✗
$\{(0,0),(1,1)\}$				
$\{(0,1),(1,0)\}$				
$\{(0,1),(1,1)\}$				

Example: classifying functions (continued)

Consider the set $\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$.
There are 9 *functions* whose domain and codomain are the set $\{0,1\}$.

Function	Total	Injective	Surjective	Bijective
$\{\}$	✗	✓	✗	✗
$\{(0,0)\}$	✗	✓	✗	✗
$\{(0,1)\}$	✗	✓	✗	✗
$\{(1,0)\}$	✗	✓	✗	✗
$\{(1,1)\}$	✗	✓	✗	✗
$\{(0,0),(1,0)\}$	✓	✗	✗	✗
$\{(0,0),(1,1)\}$	✓	✓	✓	✓
$\{(0,1),(1,0)\}$				
$\{(0,1),(1,1)\}$				

Example: classifying functions (continued)

Consider the set $\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$.
There are 9 *functions* whose domain and codomain are the set $\{0,1\}$.

Function	Total	Injective	Surjective	Bijective
$\{\}$	✗	✓	✗	✗
$\{(0,0)\}$	✗	✓	✗	✗
$\{(0,1)\}$	✗	✓	✗	✗
$\{(1,0)\}$	✗	✓	✗	✗
$\{(1,1)\}$	✗	✓	✗	✗
$\{(0,0),(1,0)\}$	✓	✗	✗	✗
$\{(0,0),(1,1)\}$	✓	✓	✓	✓
$\{(0,1),(1,0)\}$	✓	✓	✓	✓
$\{(0,1),(1,1)\}$				

Example: classifying functions (continued)

Consider the set $\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\}$.
There are 9 *functions* whose domain and codomain are the set $\{0,1\}$.

Function	Total	Injective	Surjective	Bijective
$\{\}$	✗	✓	✗	✗
$\{(0,0)\}$	✗	✓	✗	✗
$\{(0,1)\}$	✗	✓	✗	✗
$\{(1,0)\}$	✗	✓	✗	✗
$\{(1,1)\}$	✗	✓	✗	✗
$\{(0,0),(1,0)\}$	✓	✗	✗	✗
$\{(0,0),(1,1)\}$	✓	✓	✓	✓
$\{(0,1),(1,0)\}$	✓	✓	✓	✓
$\{(0,1),(1,1)\}$	✓	✗	✗	✗

Sets, bags, and sequences

- A set is a collection of objects.

  We can only ask: is $x \in S$?

  We cannot ask:

  - *how many times* does x occur in S?
  - *where* does x occur in S?

Sets, bags, and sequences

- A set is a collection of objects.

  We can only ask: is $x \in S$?

  We cannot ask:

  - *how many times* does x occur in S?
  - *where* does x occur in S?

- A bag is like a set, but we also remember *how many times* an object occurs.
- A sequence is like a set, but we also remember *where* an object occurs.

Bags: basic definition

- A bag (or multiset) is like a set except that members are allowed to occur many times.

Bags: basic definition

- A bag (or multiset) is like a set except that members are allowed to occur many times.
- The number of times an element occurs in a bag is called the multiplicity of that element.
- We typically use a special kind of bracket to indicate that we care about multiplicity;

  For example: $[\![a, a, b, b, b, c]\!]$ is the bag where a occurs twice, b occurs thrice and c occurs once.

Bags: basic definition

- A bag (or multiset) is like a set except that members are allowed to occur many times.
- The number of times an element occurs in a bag is called the multiplicity of that element.
- We typically use a special kind of bracket to indicate that we care about multiplicity;

  For example: $[\![a, a, b, b, b, c]\!]$ is the bag where a occurs twice, b occurs thrice and c occurs once.
- Notes:
  - The notation $[\![\cdots]\!]$ is not very standard.
  - The *order* of elements in a bag doesn't matter.

Bags: set-based definition

- A bag is really a special kind of **function**.
- For any set S, a "bag of S" is a function from S to $\mathbb{N}$, giving the multiplicity of each element.

Bags: set-based definition

- A bag is really a special kind of **function**.
- For any set S, a "bag of S" is a function from S to $\mathbb{N}$, giving the multiplicity of each element.
- Thus $[\![a, a, b, b, b, c]\!]$ is just a shorthand for $\{(a, 2), (b, 3), (c, 1)\}$

Bags: set-based definition

- A bag is really a special kind of **function**.
- For any set S, a "bag of S" is a function from S to $\mathbb{N}$, giving the multiplicity of each element.
- Thus $[\![a, a, b, b, b, c]\!]$ is just a shorthand for $\{(a, 2), (b, 3), (c, 1)\}$
- For example, given some set $F \triangleq \{apple, orange, pear\}$, given the bag
$$myShopping \triangleq [\![apple, apple, pear, pear, pear]\!]$$

Bags: set-based definition

- A bag is really a special kind of **function**.
- For any set S, a "bag of S" is a function from S to $\mathbb{N}$, giving the multiplicity of each element.
- Thus $[\![a, a, b, b, b, c]\!]$ is just a shorthand for $\{(a, 2), (b, 3), (c, 1)\}$
- For example, given some set $F \triangleq \{apple, orange, pear\}$, given the bag

$$myShopping \triangleq [\![apple, apple, pear, pear, pear]\!]$$

we can say

$$myShopping = \{(apple, 2), (orange, 0), (pear, 3)\}$$

Operations on Bags (inherited)

- Since bags are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.

Operations on Bags (inherited)

- Since bags are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.
- Example:

$$[\![a, a, b, b, b, c]\!] \ \cup \ [\![a, b, b, c, c]\!]$$

Operations on Bags (inherited)

- Since bags are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.
- Example:

$$
\begin{aligned}
& [\![a, a, b, b, b, c]\!] \;\cup\; [\![a, b, b, c, c]\!] \\
= \;& \{(a, 2), (b, 3), (c, 1)\} \;\cup\; \{(a, 1), (b, 2), (c, 2)\}
\end{aligned}
$$

Operations on Bags (inherited)

- Since bags are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.
- Example:

$$
\begin{aligned}
& [\![a, a, b, b, b, c]\!] \;\cup\; [\![a, b, b, c, c]\!] \\
= \;& \{(a,2),(b,3),(c,1)\} \;\cup\; \{(a,1),(b,2),(c,2)\} \\
= \;& \{(a,2),(b,3),(c,1),(a,1),(b,2),(c,2)\}
\end{aligned}
$$

Operations on Bags (inherited)

- Since bags are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.
- Example:

$$\begin{aligned} & [\![a, a, b, b, b, c]\!] \cup [\![a, b, b, c, c]\!] \\ = \quad & \{(a, 2), (b, 3), (c, 1)\} \cup \{(a, 1), (b, 2), (c, 2)\} \\ = \quad & \{(a, 2), (b, 3), (c, 1), (a, 1), (b, 2), (c, 2)\} \end{aligned}$$

- This is a relation, but not a function (and thus not a bag)
  - probably not what we intended to happen...

Operations on Bags (new)

- We can define a special union operator just for bags, so that e.g.

$$[\![a, a, b, b, b, c]\!] \uplus [\![a, b, b, c, c]\!] = [\![a, a, a, b, b, b, b, b, c, c, c]\!]$$

Operations on Bags (new)

- We can define a special union operator just for bags, so that e.g.

$$[\![a, a, b, b, b, c]\!] \uplus [\![a, b, b, c, c]\!] = [\![a, a, a, b, b, b, b, b, c, c, c]\!]$$

- Definition: for any two bags A and B,

$$A \uplus B = (\lambda\, x \cdot A(x) + B(x))$$

Operations on Bags (new)

- We can define a special union operator just for bags, so that e.g.

$$[\![a, a, b, b, b, c]\!] \uplus [\![a, b, b, c, c]\!] = [\![a, a, a, b, b, b, b, b, c, c, c]\!]$$

- Definition: for any two bags A and B,

$$A \uplus B = (\lambda\, x \cdot A(x) + B(x))$$

- Notes:
  - Since any bag is a function, $B(x)$ is just the multiplicity of element x in bag B.
  - Can you define bag versions of intersection and difference?
  - Can you define the relation *is-a-subbag-of*?

Sequence: basic definition

- A sequence is like a set except that we remember the order in which members occur.

Sequence: basic definition

- A sequence is like a set except that we remember the order in which members occur.
- The place where an element occurs in a sequence is called the position of that element.
- We typically use a special kind of bracket to indicate that we care about ordering;

  For example: $\langle a, b, c, b, a, a \rangle$ is the sequence where a occurs at positions 1, 5 and 6, b occurs at positions 2 and 4, and c occurs at position 3.

Sequence: basic definition

- A sequence is like a set except that we remember the order in which members occur.
- The place where an element occurs in a sequence is called the position of that element.
- We typically use a special kind of bracket to indicate that we care about ordering;

  For example: $\langle a, b, c, b, a, a\rangle$ is the sequence where a occurs at positions 1, 5 and 6, b occurs at positions 2 and 4, and c occurs at position 3.
- Notes:
  - The notation $\langle \cdots \rangle$ is not very standard.
  - We usually start counting position at 1 (unlike arrays)

Sequences: set-based definition

- A sequence is really a special kind of **function**.
- For any set S, a sequence of S is a (partial) function from $\mathbb{N}$ to S, giving the element at each position.

Sequences: set-based definition

- A sequence is really a special kind of **function**.
- For any set S, a sequence of S is a (partial) function from $\mathbb{N}$ to S, giving the element at each position.
- Thus $\langle a, b, c, b, a \rangle$ is just a shorthand for
$$\{(1, a), (2, b), (3, c), (4, b), (5, a)\}$$

Sequences: set-based definition

- A sequence is really a special kind of **function**.
- For any set S, a sequence of S is a (partial) function from $\mathbb{N}$ to S, giving the element at each position.
- Thus $\langle a, b, c, b, a \rangle$ is just a shorthand for
$$\{(1, a), (2, b), (3, c), (4, b), (5, a)\}$$
- For example, given some set $F \triangleq \{apple, orange, pear\}$, given the sequence
$$myEating \triangleq \langle apple, pear, apple, pear, apple \rangle$$

Sequences: set-based definition

- A sequence is really a special kind of **function**.
- For any set S, a sequence of S is a (partial) function from $\mathbb{N}$ to S, giving the element at each position.
- Thus $\langle a, b, c, b, a \rangle$ is just a shorthand for
$$\{(1, a), (2, b), (3, c), (4, b), (5, a)\}$$
- For example, given some set $F \triangleq \{apple, orange, pear\}$, given the sequence
$$myEating \triangleq \langle apple, pear, apple, pear, apple \rangle$$
we can say
$myEating = \{(1, apple), (2, pear), (3, apple), (4, pear), (5, apple)\}$

Operations on Sequences (inherited)

- Since sequences are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.

Operations on Sequences (inherited)

- Since sequences are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.
- Example:

$$\langle a, a, b \rangle \ \cup \ \langle a, b \rangle$$

Operations on Sequences (inherited)

- Since sequences are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.
- Example:

$$\begin{aligned} & \langle a, a, b \rangle \ \cup \ \langle a, b \rangle \\ = & \ \{(1, a), (2, a), (3, b)\} \ \cup \ \{(1, a), (2, b)\} \end{aligned}$$

Operations on Sequences (inherited)

- Since sequences are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.
- Example:

$$
\begin{aligned}
& \langle a, a, b \rangle \ \cup \ \langle a, b \rangle \\
= & \ \{(1, a), (2, a), (3, b)\} \ \cup \ \{(1, a), (2, b)\} \\
= & \ \{(1, a), (2, a), (2, b), (3, b)\}
\end{aligned}
$$

Operations on Sequences (inherited)

- Since sequences are a special kind of function/relation/set, we can use all the function/relation/set operations with them.
- But note: the union (or intersection) of two functions is a relation, but not necessarily a function.
- Example:

$$
\begin{aligned}
& \langle a, a, b\rangle \cup \langle a, b\rangle \\
= \;& \{(1, a), (2, a), (3, b)\} \cup \{(1, a), (2, b)\} \\
= \;& \{(1, a), (2, a), (2, b), (3, b)\}
\end{aligned}
$$

- This is a relation, but not a function (and thus not a sequence) - probably not what we intended to happen...

Operations on Sequences (inherited)

- One useful inherited operation is *cardinality*:
  - The cardinality of a finite set is the number of elements it contains.
  - **Notation:** If S is a finite set, we write $\#S$ to denote the cardinality of S.

- Since a sequence is a set of tuples, the cardinality of a sequence is also its *length*.

  Example:

$$\#\langle a, b, c, b, a \rangle = 5$$

Operations on Sequences (inherited)

- One useful inherited operation is *cardinality*:
  - The cardinality of a finite set is the number of elements it contains.
  - **Notation:** If S is a finite set, we write $\#S$ to denote the cardinality of S.

- Since a sequence is a set of tuples, the cardinality of a sequence is also its *length*.

  Example:

  $$\#\langle a, b, c, b, a \rangle = 5$$

- The **empty set** $\emptyset$ is also a sequence; when we want to refer to it as a sequence we usually write $\langle \rangle$

  Naturally, $\#\langle \rangle = 0$

Operations on Sequences (new)

- One of the most important sequence-specific operations is concatenation.
- This just appends two sequences together, in order:

$$\langle a, b, c \rangle \frown \langle b, a \rangle = \langle a, b, c, b, a \rangle$$

$$\langle a, b, c \rangle \frown \langle b, a \rangle \frown \langle b, a \rangle = \langle a, b, c, b, a, b, a \rangle$$

Operations on Sequences (new)

- One of the most important sequence-specific operations is concatenation.
- This just appends two sequences together, in order:

$$\langle a, b, c\rangle \frown \langle b, a\rangle = \langle a, b, c, b, a\rangle$$

$$\langle a, b, c\rangle \frown \langle b, a\rangle \frown \langle b, a\rangle = \langle a, b, c, b, a, b, a\rangle$$

- Definition: for any two sequences s and t,

$$s \frown t \quad = \quad \lambda n \in \mathbb{N} \cdot \begin{cases} s(n) & \text{if } 1 \leq n \leq \#s \\ t(n - \#s) & \text{if } (\#s + 1) \leq n \leq (\#s + \#n) \end{cases}$$

Operations on Sequences (new)

- One of the most important sequence-specific operations is concatenation.
- This just appends two sequences together, in order:

$$\langle a, b, c \rangle \frown \langle b, a \rangle = \langle a, b, c, b, a \rangle$$

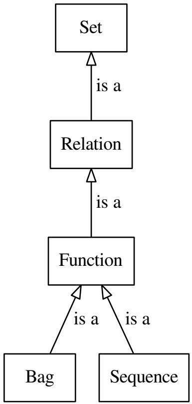
$$\langle a, b, c \rangle \frown \langle b, a \rangle \frown \langle b, a \rangle = \langle a, b, c, b, a, b, a \rangle$$

- Definition: for any two sequences s and t,

$$s \frown t \quad = \quad \lambda n \in \mathbb{N} \cdot \begin{cases} s(n) & \text{if } 1 \leq n \leq \#s \\ t(n - \#s) & \text{if } (\#s + 1) \leq n \leq (\#s + \#n) \end{cases}$$

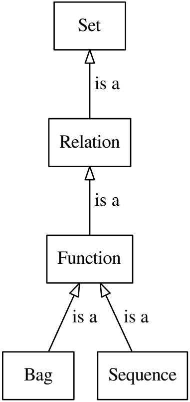
- Notes:
  - Since any sequence is a function, $s(n)$ is just the element at position n in sequence s.
  - Notation: sometimes we write $s.t$ for the concatenation of s and t.

Sets: the full “class hierarchy”



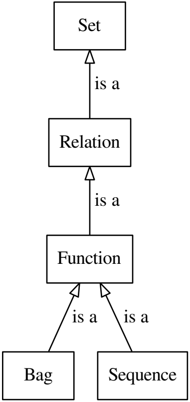
Sets: the full "class hierarchy"

- We took *sets* as the basic concept because they fit well with logic.



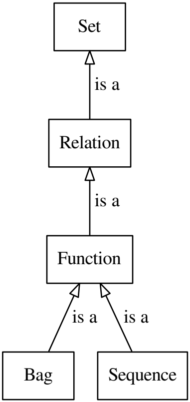
Sets: the full "class hierarchy"

- We took *sets* as the basic concept because they fit well with logic.
- Database theory takes *relations* as the basic concept (Codd's relational algebra).



Sets: the full "class hierarchy"

- We took *sets* as the basic concept because they fit well with logic.
- Database theory takes *relations* as the basic concept (Codd's relational algebra).
- The theory of computation takes *functions* as the basic concept (Church's λ-calculus).



Sets: the full "class hierarchy"

- We took *sets* as the basic concept because they fit well with logic.
- Database theory takes *relations* as the basic concept (Codd's relational algebra).
- The theory of computation takes *functions* as the basic concept (Church's λ-calculus).
- Language theory takes *sequences* as the basic concept (Kleene's regular expressions, Chomsky's grammars etc.).