# Week 1

## Introduction to Formal Methods

* A formal method is a systematic, mathematically based approach to determine whether a program has specific desirable properties.

Software development life cycle🡺

Requirement analysis 🡺 System specification 🡺 Design 🡺 prototyping & implementation 🡺 verification, testing & integration 🡺 maintenance & enhancement

Abstract machine is a specification of:

* What a system should be like
* How it should behave
* Not how a system is to be built

Abstract machine has 🡪 name , local state (encapsulated variables), & interface (collection of operations)

**MACHINE**

**CONSTRAINS**

**SETS**

**CONSTANTS**

**PROPERTIES**

**VARIABLES**

**INVARIANTS**

**INITIALISATION**

**OPERATIONS**

# Week 2

## Set

🡺 collection of well-defined (same kind) objects (elements or members)

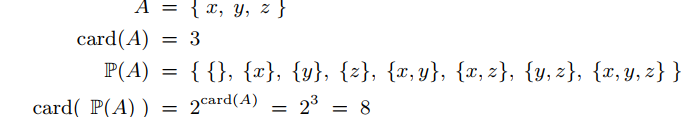
🡺 do not include duplications

🡺 The order of the set doesn’t have a meaning

🡺 B language ensures that the set contains the same kind of things, the type is used to check it

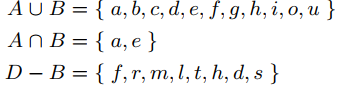
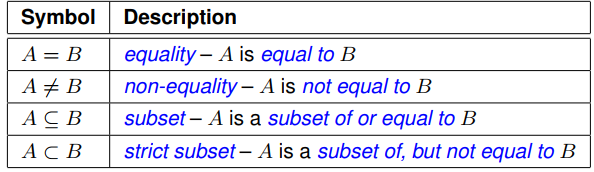
Integers 🡺

Natural numbers 🡺 

Power set 🡺 (set containing all subsets of X) 

x..y 🡺 range of numbers from x up to y inclusive

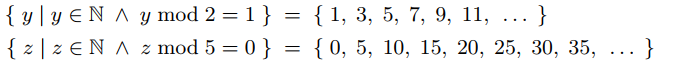


+

### SET Relations

Construction of sets 🡺 given condition must hold all the members in the set

General form 🡺

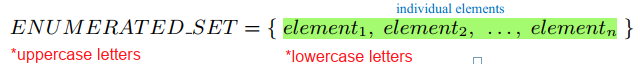
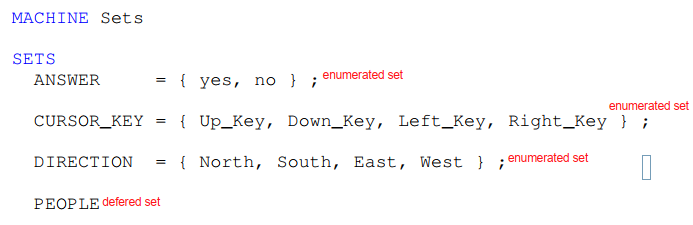
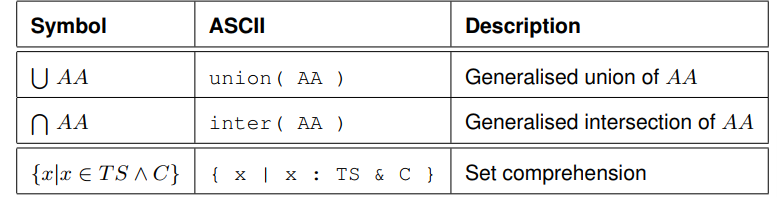
 E.g.:



Abstract machine

* state variables of the system and their types
* state invariants 🡺 properties that variables must satisfy
* Operations 🡺 transform and update the variables

**Sets** are used to achieve formalisation & abstraction

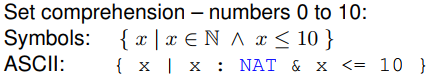
* Enumerated 🡺 defined sets
* Deferred 🡺 incomplete and definition delayed
  + High degree of abstraction

member of 🡪 if a single value variable or constant

var : TYPESET

subset of 🡪 if set values variable or constant

var <: TYPESET

 element of a power set 🡪 set valued variable

var : POW(TYPESET)

# Week 3

B model is a high-level, abstract modification specification that provides what the system should be like and how it should behave rather than how it should be implemented.

## Stack states

System invariants 🡪 0 <= no. of items <= maximum size

### Operations

Start 🡪 Empty stack (valid state)

Push 🡪 if the stack is not full (Results in the number of items > maximum size)

Pop 🡪 if the stack is not empty (Results in the number of items < zero)

Abstract machine:

* Name & Parameters
* Local state represented by encapsulated state variables
* State invariant that state must satisfy
* Initialisation of states
* Operations to update and manipulate the state variables

Main clauses:

* MACHINE 🡪 declaration of name of the abstract machine and optional list of parameters
* CONSTRAINTS 🡪 properties that machine parameters must satisfy
* SETS 🡪 deferred & enumerated sets (uppercase)
* CONSTANTS 🡪 constants declaration (lowercase)
* PROPERTIES 🡪 sets and constants must satisfy
* VARIABLES 🡪 variable declaration (machine state is defined by variables)
* INVARIANT 🡪 variable types defined by invariant properties
  + Static specifications 🡪 something that must be true of every state the machine can reach
* INITIALISATION 🡪 variable initialization
* OPERATIONS 🡪 declaration of operations that update variables (dynamic specification of machine)
  + Must always preserve the invariants
  + Separated by “**;”**
  + Pre-conditions 🡪 restrictions on parameters and give type of input parameters

## Abstract Machine Notation

Elements:

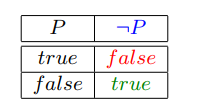
* Predicate logic 🡺 express properties of all data
  + - E.g.: invariants, preconditions (proof obligations)
* Expressions 🡺 describe data type and value
  + Using numbers, sets, relations, functions & sequences
* Substitutions 🡺 mathematical notations that describe dynamic aspects as state changes (body of an operation)
* Components 🡺 Machines, refinements, & Implementations
* Assigned to 🡪 **:=**
* Do nothing 🡪 skip (explicit state)

|  |  |
| --- | --- |
| IF condition | CASE statement |
| Parallel Execution  xx := 4 || yy := 7 similar to xx, yy := 4, 7 |  |

# Week 4: logic

## Propositional Logic

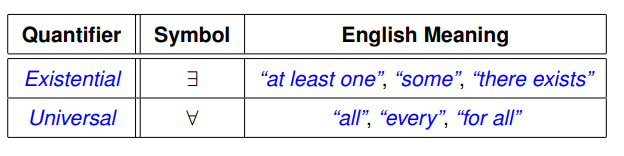
A proposition is a logical expression which is true or false.

Law of Excluded Middle 🡺 Whenever a proposition is true the opposite proposition of it is false

Preposition symbol :

* P \/ Q is true if and only if P is true or Q is true
* P/\ Q is true if and only if P and Q are both true
* P => Q is false if and only if P is true and Q is false (P & Q doesn’t need to be connected in any way)
* P ⬄ Q is true if and only if P 🡺Q & Q🡺P are true (both either true-true or false-false)
* (P) is true if and only if P is true
* **Not 🡪 And 🡪 OR 🡪 Implication 🡪 Equivalence**
* Tautology –true for every interpretation
* Inconsistency – false in every interpretation
* Contingency – true at least once and false at least once

## Predicate Logic

* Either true or false
* Contains:
  + Objects (Variables)
  + Properties of objects
  + Relationships between objects
* Predicate with 0 arguments is a proposition
* Valid – if the predicate is true for all values
* Satisfiable – if the predicate is true for some values
* Unsatisfiable – if the predicate is false for all values

Quantifiers:

**Existential Quantification**

🡺 if *Pred(xx)* is true for at least one value of xx from the set *TYPESET* then the formula is true otherwise false 

**Universal Quantifier**

🡺if *Pred(xx)* is true for all values of xx from the set *TYPESET* then the formula is true otherwise false 

|  |  |
| --- | --- |
| **Bound Variables** | **Free Variables** |

# Week 6

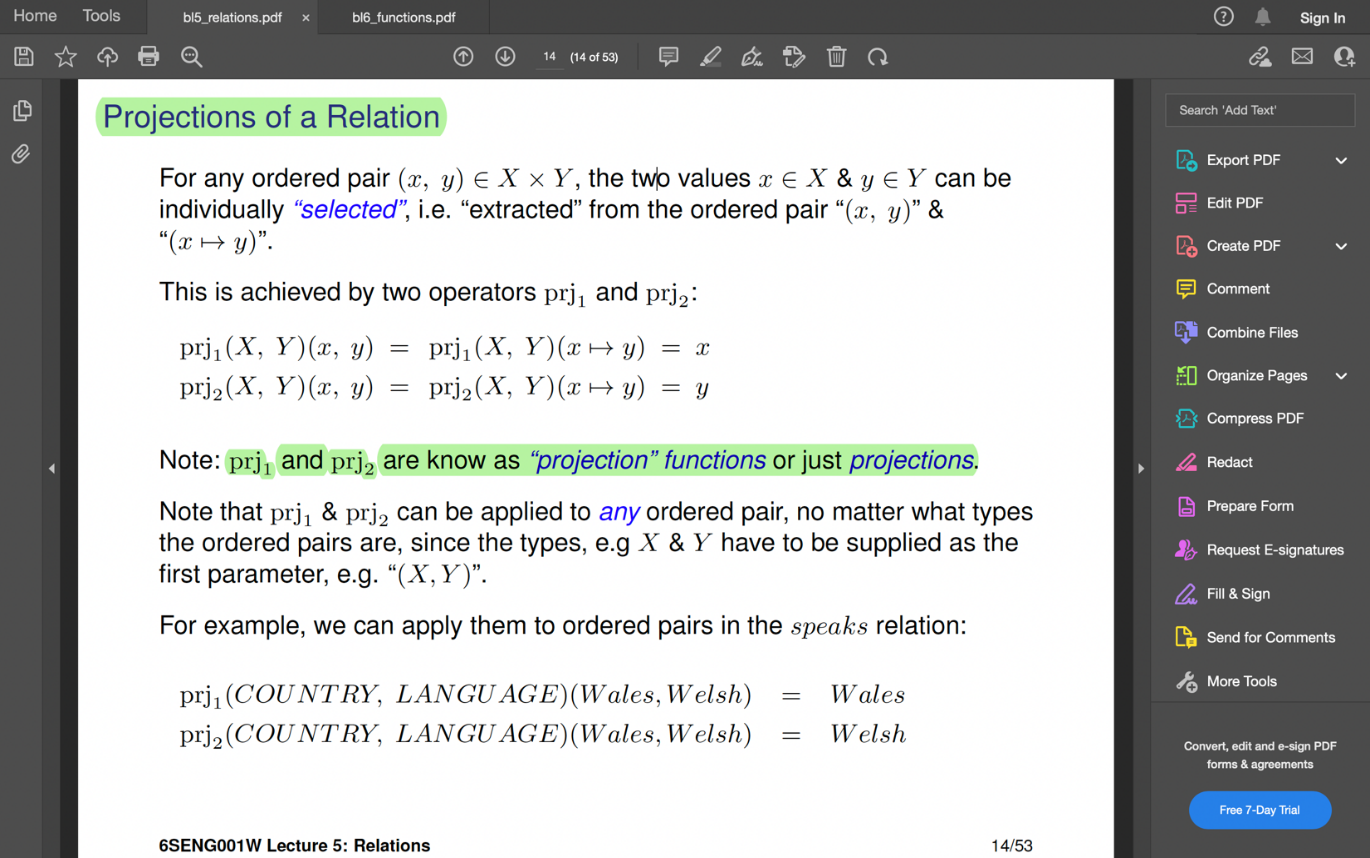
## Relations

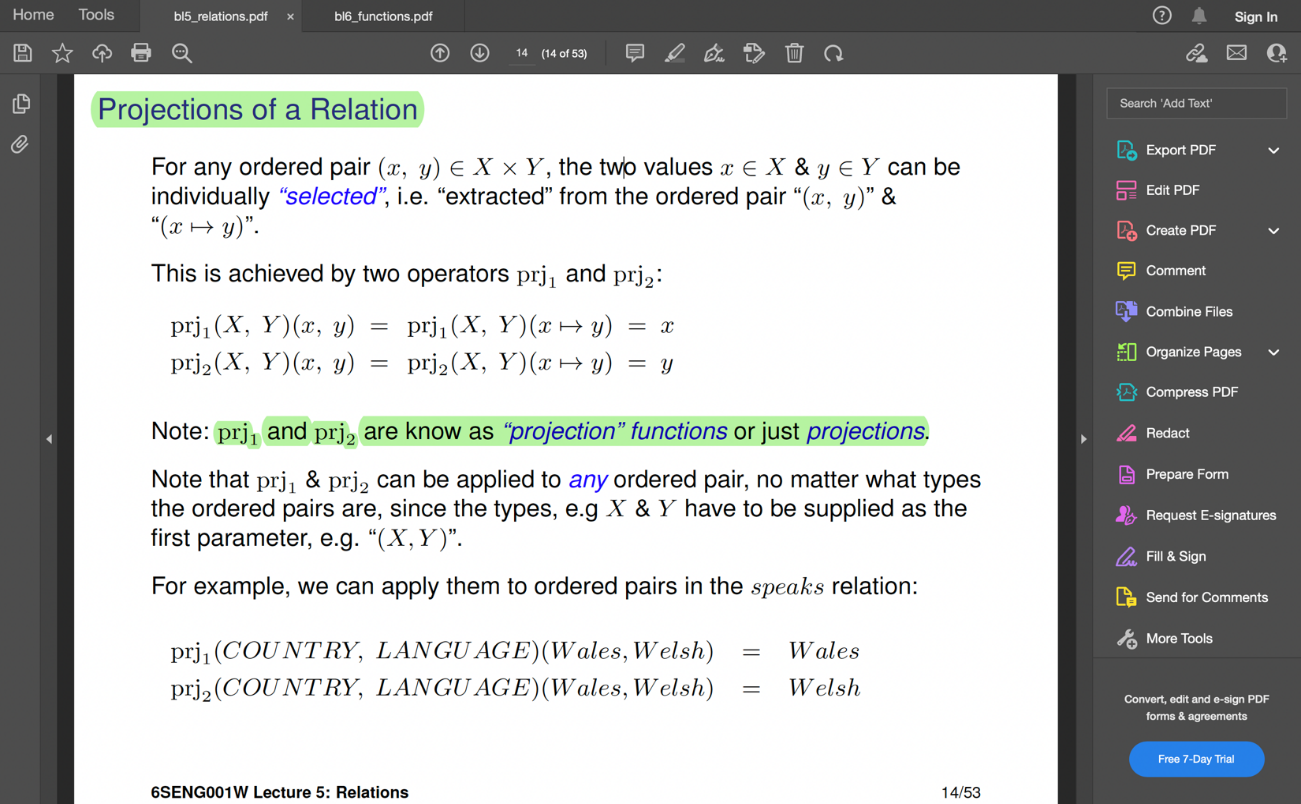
Relation 🡺 Relation is a special kind of set where the relationship of sets is represented in order. 

Cartesian Product 🡪 can be used to combine any number of sets. (many-to-many)

Binary Relations 🡪 set of ordered pairs, of related values.

Ordered pair and maplet🡪

Projections🡪



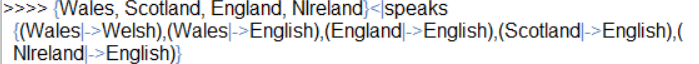
Membership 🡺 To see if a relation relates to a pair of values.

Domain (source) and Range (target)

Relational Image 🡪 set of values from the range of a relationrelated to a set of values from the domain R[A]

What languages are spoken in Canada & Wales? speaks [ {Canada, Wales}] = {French, English, Welsh}

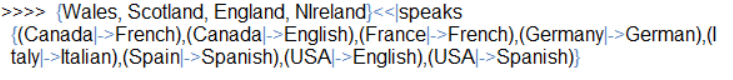
**Domain Restriction** 🡪 create a new relation by **restricting** the **domain** of the relation



**Range Restriction** 🡪 restrict a relation to a part of **range**

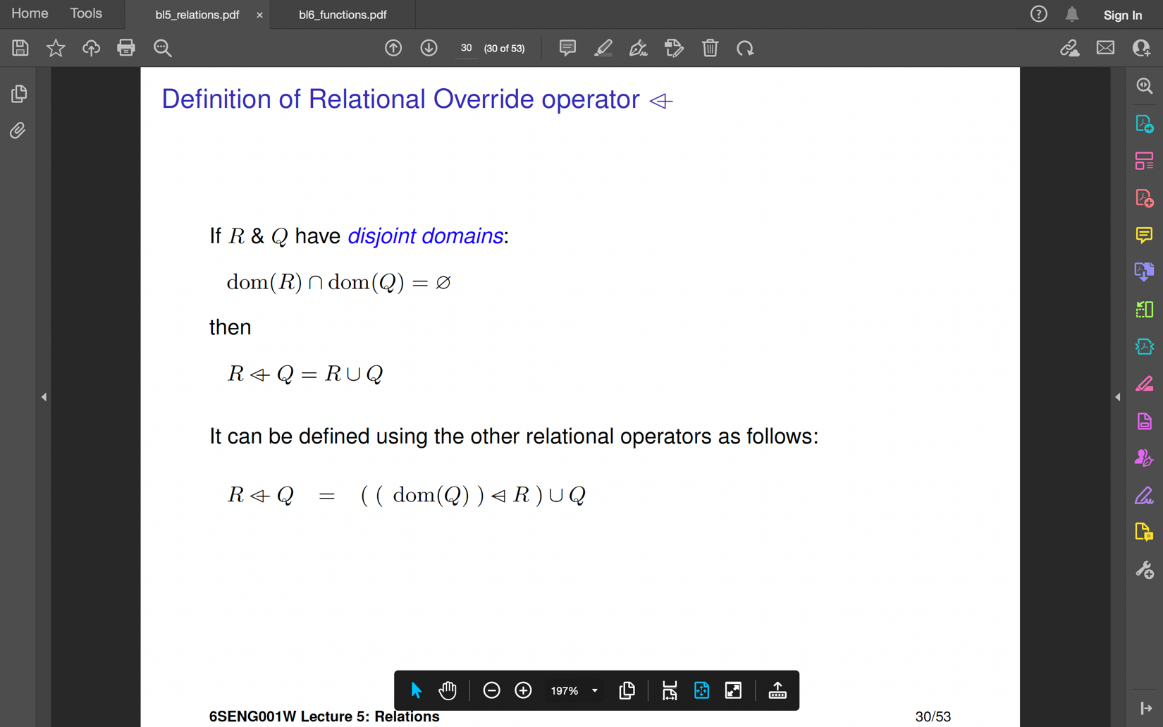


**Domain Anti- Restrictions** (domain subtraction) 🡪 restrict a relation to that part **where the domain is not contained** in a set.



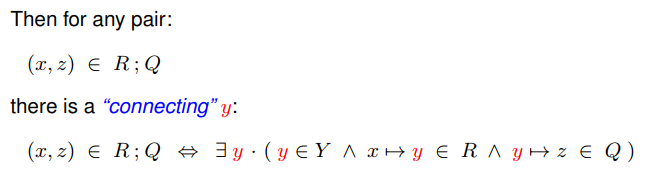
**Range Anti-Restrictions**🡪 restrict a relation to that part where the **range is not contained** in a set.



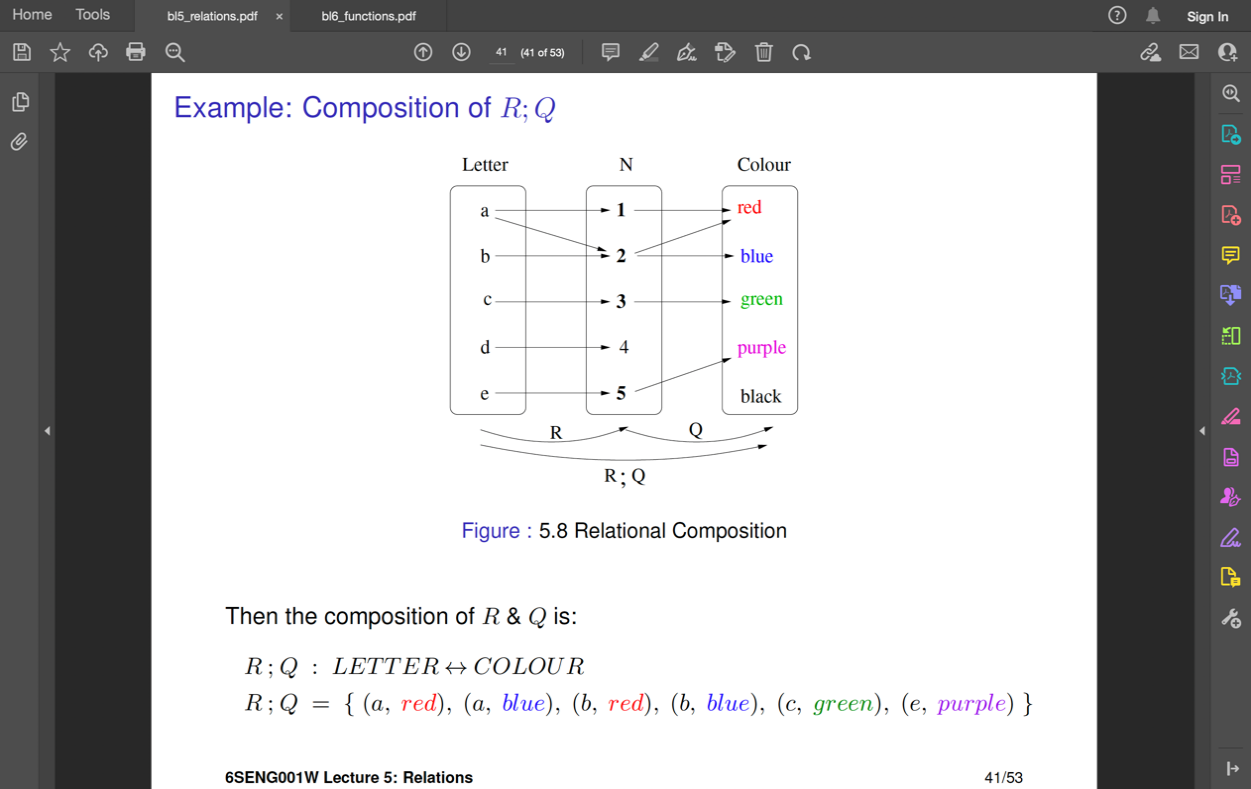
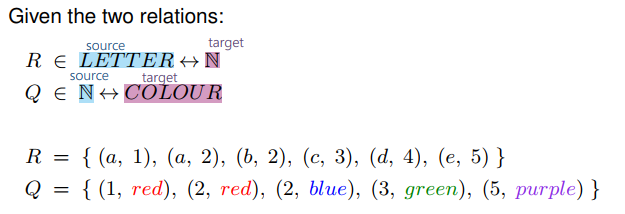
**Relational Override Operator** 🡪 A relation can be modified by adding pairs, removing pairs or by altering.

R overridden by the relation Q

|  |  |
| --- | --- |
| **Identity Relation** 🡪 is a relation which maps all the elements of X to themselves. | **Inverse Relation**🡪 |



**Compositions of Relation** 🡪 Relations can be joined together by an operation called composition.



**Reflexive relationship**

X = {1, 2, 3} R = {(1, 1), (1, 2), (2, 1), (2, 2), (3, 3)} \* are reflexive

R1={(1,1), (1,2), (2,1), (2,2), (3,3)} R2={(1,1), (1,2), (2,1), (3,3)}

R1 & R2 are not in reflexive relationship. To make this relationship reflexive, we can make the reflexive closure taking the union with the identity relationship.

Reflexive closure = R U id(x)

={(1,1), (1,2), (2,1), (2,2), (3,3)} U {(1,1), (2,2), (3,3)}

**Transitive** 

**Reflexive -Transitive** 

# Week 7

## Functions

Function 🡺 way of specifying some processing which produces a value as a result

Function either provides – a result value or an augmented value

### B functions

* special relation
* is a set of maples/ordered pairs
* function *f* mapping from *x* to *y* is a simple set of (x,y) or

Functions

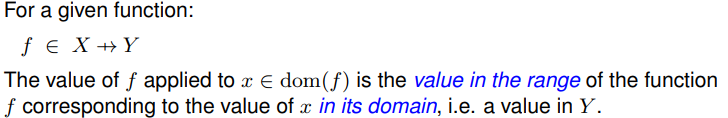
* many-to-many 🡺 is only a relation, so not a function
* 1-to-many 🡺 is only a relation, so not a function
* many-to-1 🡺 is a function
* 1-to-1 🡺 is a function

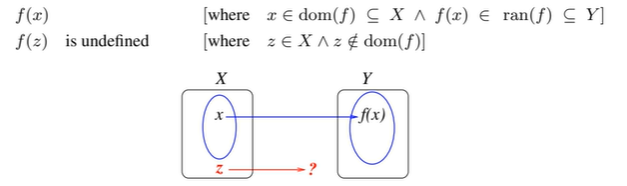
Valid function🡺 many values in the domain of the function *dom(f)* to be all mapped onto the same values of the range of the function *ran(f)*

If y!= z, the property of a functionality rules out as it is a 1-to-many relation.

|  |  |
| --- | --- |
| **Relation** | **Function** |
| * not a function, as the value 3 in the domain is mapped to more than one value | * is a function as no value in the domain is mapped to more than one value |

### Function application





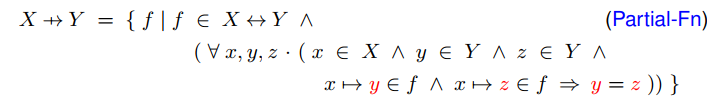
The function is only meaningful for the values of  *x* in the *dom(f)*

#### Partial Functions

* Some source values are not in the domain of the function.

i.e., for *dom(f) is a subset of X (not equal to X)*

* Partial functions can be defined formally as relations that satisfy the properties of a function



\*if ***f*** is a function, the ***x*** element cannot be related to 2 different range elements; therefore, ***y must equal z***

#### Total Functions

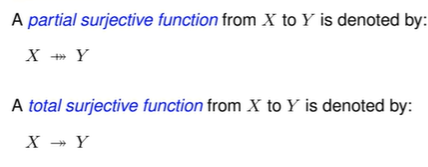
* There is value for every possible *x* in the source set ***X***, so ***f(x)*** is always defined
* dom(f) = source set (X)
* every element of the source set ***(X)*** can find an element from Y



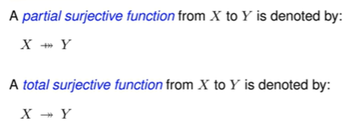
#### Injective Functions (one-to-one function)

* Different values of the source are mapped to different values of the target
* Each value of ***dom(f)*** is mapped to a different value in its ***ran(f)***
* Can be total-injective or partial-injective
* Inverse of an injective function ***f -1***is also an injective function

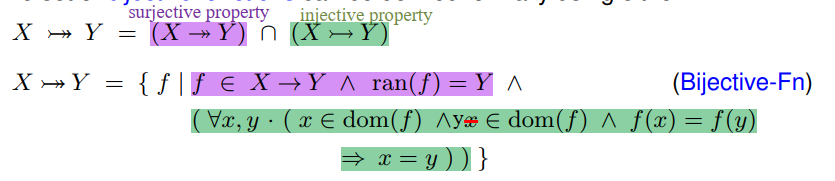
#### Surjective Functions (on-to functions)

* A value in the range for every value of the target





#### Bijective Function (one-to-one relationship)

* Satisfy all the properties of Injective, surjective and the total functions

## Relational Override with functions

Function modification by adding mapping pairs, or removing teams, or changing the mapping between values.

* Function ***f***overridden by function ***g***: 

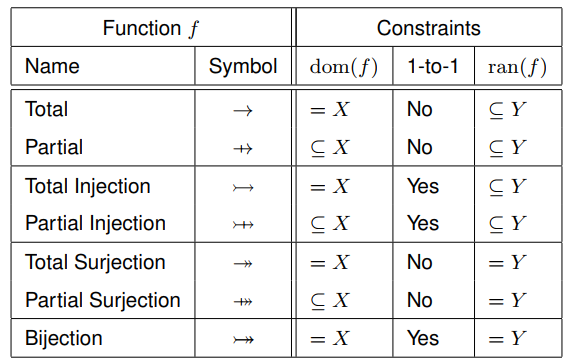


* When function ***f*** is overridden by function ***g***, and the domains are disjoint:



If both ***f*** and ***g*** are functions, the result is a function

If one of them is a relation, the relation is highly likely a relation



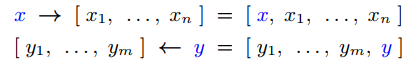
# Week 8

## Sequences

* Needed for:
  + Distinguish values of a set by position
  + Permit duplicates
  + Impose order of values
* Sequence is a constant can be defined by listing elements in order.
* Empty sequence 🡪 [ ]
* Sequence definition 🡪



* Injective sequence 🡪
* Concatenation 🡪 join sequences (total function)



* Front & End insertion 🡪 (total function)



* Selection 🡪



* First 🡪 (partial function)



* Tail 🡪 (partial function)



* Last 🡪 (partial function)



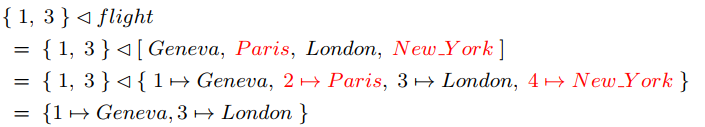
* Front 🡪



* Relationships between operators 🡪
* Front sub sequence 🡪 select the first *n* elements of the sequence
* Tail sub sequence 🡪 remove the first n elements of the sequence



* Reversal 🡪 (total function)
* Relational Domain Restriction 🡪



* Relational Range Restriction 🡪