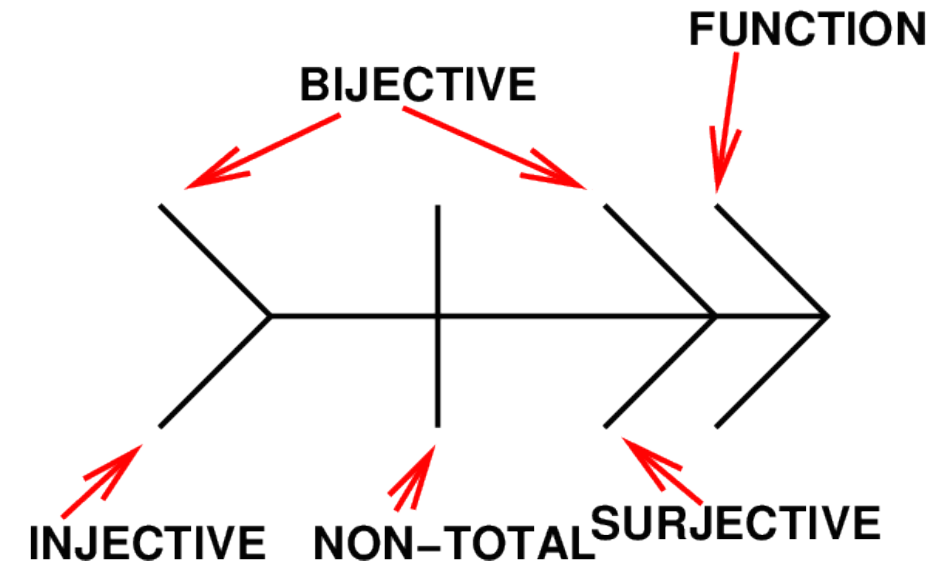
**Formal Specification in Z**

* **Formal specification**
  + Writing the description of a system in a language (e.g. logic)
  + Each formula has a unique meaning, i.e. unambiguous
  + System specification describes what the system should do, not how it should work
* **The Z specification language**
  + The execution of a system is a sequence of states, where each state is a mapping (variables → values)
  + The following are described:
    - Types of data manipulated by the system
    - Constants & their types
    - System elements, their types, and invariants about relationships between them
    - Initial system state
    - System operations & their effects on system elements
* Schemas
  + Signature – introduce elements w/ their types
  + Predicate – contains wffs
    - When describing system state, invariants are listed here – must be true in all states
  + Schema inclusion – adds the signatures and predicates from the included schema
  + Next value of variable = variable-prime
    - E.g. X = current value; X’ = next value
  + Can include schemas with current and next values
    - ΔSchema = equivalent to
      * Schema
      * Schema’
  + Input – p?
  + Output – p!
* Operations
  + Schema for operations should include a Δ schema of system state
    - E.g. ΔSystem
    - X’ = X + 1
    - Y’ = Y’ – 1 etc.
  + Preconditions – conditions on current state & inputs
  + Postconditions – conditions on next state & outputs
* Constants – not declared in state or operation schemas because their values don’t change during execution
* Types
  + Built-in types – N (natural numbers), Z (integers) etc.
  + Generic types – [Person] (does not provide specific elements)
  + Free/enumerated types – Colours ::= Red | Blue | Green (provide specific elements)
  + Compound types



* Function
  + Each domain element is mapped to at most one range element
  + All functions are partial functions
  + The set of partial functions from A to B =
    - A -|-> B = {f : A ↔ B | ∀x : A . ∀y, z : B . (x, y) ∈ f ∧ (x, z) ∈ f ⇒ y = z}
  + Function space – a set of functions
  + Functions can be described with set comprehension
    - E.g. double = {(x, 2x) . x : N}
* Total function
  + Defined for every domain element (a mapping exists for every element)
  + The set of total functions from A to B =
    - A --> B = {f : partial function from A to B | dom(f) = A} ⊆ A -|->
* Surjective/onto
  + Every range element is mapped to by a domain element
  + |domain| ≥ |range|
  + f from A to B is surjective iff:
    - ∀b : B . ∃a : A . f(a) = b
  + Partial surjective: A -|->> B = {f : A -|-> B | ran(f) = B}
  + Total surjective: A -->> B = {f : A --> B | ran(f) = B}
* Injective/one-to-one
  + Each domain element is mapped to a unique range element
  + i.e. its inverse is also an injective function
  + f from A to B is injective iff:
    - ∀x, y : A . f(x) = f(y) ⇒ x = y
  + Partial injective: A >-|-> B = {f : A -|-> B | f~ ∈ B -|-> A}
  + Total injective: A >--> B = {f : A --> B | f~ ∈ B -|-> A}
* Bijective
  + Both surjective & injective
* Function composition
  + (f; g)(x) = (g ° f) = g(f(x)) provided that x ∈ dom(f) and f(x) ∈ dom(g)
* Zai-schema (Ξ)
  + Asserts that the state elements do not change
  + Equivalent to including a Δ schema and writing X’ = X, Y’ = Y, etc.
* Exception handling
  + For an operation with precondition(s):
    - Primary operation – desired effect when the precondition is satisfied
    - Exception handling operations – system behaviour when precondition is not satisfied
  + Exception handling schemas
    - Do not change system elements, instead issue an error message
    - Use Ξ schema
    - Use Report ::= ReportType1 | ReportType2 etc.
  + Multiple exceptions with the same error message can be grouped in one handler schema
    - Disjunctions must be stated explicitly – multiple preconditions are conjuncted by default
* **Working with tuples**
  + If the tuple is part of a total function, f(x) = second element associated w/ x