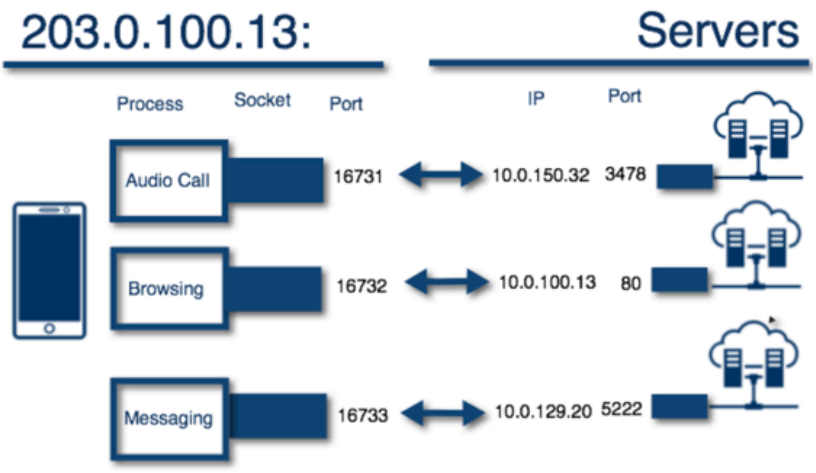
UDP

Connecting programs



A socket is an end-point in a two-way communication channel

x

What are the attributes of UDP?

1. UPD is connectionless
    1. There is no handshake protocol between the Source and the Destination.
2. Port-to-Port
3. Process-to-Process communication
4. No ordering - No retry

Relations and Functions

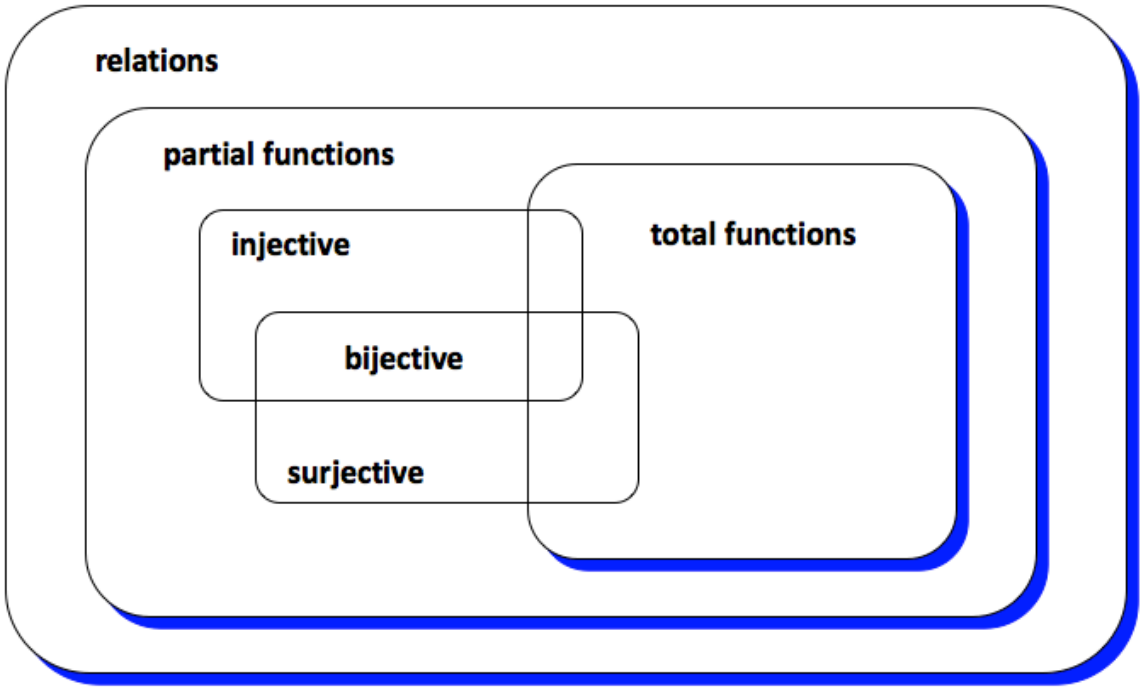


Figure 1. Types of Mathematical Functions.

Relations are encoded as a set of pairs. The first elements of the pairs are part of the *source* or *domain* of the relation, and the second element of each pair is a member of the *target* or *range* of the relation. A function is a relation such that no two distinct pairs contain the same element. In other words, any source element is mapped to at most one element of a target.

Let us suppose that `f` is a relation with domain `A` and range `B`, denoted `f: A ↔ B`. If `f` is a function defined for *all* values of `A`, we say that `f` is a *total* function, and we write `f: A → B`. If `f` is defined for some values of `A`, we say that `f` is a *partial* function, and we write `f: A ⇸ B`. If `f` is a function such as no element in the range of `f` is associated with more than one element in the domain of `f`, then we say that `f` is a *one-to-one* or *injective* function, and we write `f: A ↣ B`. If `f` is a function whose range is `B`, we say that `f` is an *onto* or *surjective* function, and we write `f: A ↠ B`. If `f` is both one-to-one and onto, we say that `f` is a *bijection*, and we write `f: A ⤖ B`.

Figure 1 shows a hierarchy of types of relations and functions. Any function is a relation. Functions can be *total* or *partial*. Every total function is a partial function. Functions can be *injective* or *surjective*. Some surjective functions are partial and other ones are total. Some injective functions are partial and other ones are total. Bijective functions are surjective and injective functions. Some bijective functions are partial and other ones are total.

Example: Facebook

Let us suppose that we want to identify the most basic logical structures of a model for Facebook (sets, relations, functions). We are interested in modelling users, social network content, user's pages, content ownership, and permissions on content. We want to model two types of permissions, namely, *view* and *edit* permissions. A page can be modelled as a

relationship from persons to contents: `page ∈ person ↔ content`. This declaration states that `page` is an element of all the possible set of relations that can be formed of elements of persons in the domain and network content in the range of the relation.

A relation is just a set of pairs and thus a user page can be defined as `page ≜ {(John, Photo1), (John, Video1), (Mary, Photo1), (Mary, Video2)}` whereby `John` has two content items, `Photo1` and `Video1`, `Mary` has the same photo as `John`, but has a different video on her page. The domain of `page` is `dom(page) ≜ {John, Mary}`, and its range is `ran(page) ≜ {Photo1, Video1, Video2}`. Notice the use of the symbol "≜" for "is defined as" or "is written as", which should not be confused with the symbol "=", a Boolean operator used for "equals to".

We can enforce a stricter definition for user pages by enforcing that every user has (at least) one content. This is achieved by either asserting the invariant `person = dom(page)` or, equivalently, by making `page` a total relation `page ∈ person ↔ content`. Likewise, one can enforce the property saying that every content is on someone's page by asserting the invariant `content = ran(page)` or, equivalently, by declaring `page` as a surjective relation `page ∈ person ↔ content`. Finally, we can enforce both properties by declaring `page` as a total surjective relation `page ∈ person ↔ content`.

UDP's Software Requirements

#	Invariants
0	UDP communication is connectionless, a service does not need to establish a communication to send or receive data.
1	UDP data transmission consists of two stream of bytes, one for sending data and one for receiving it.
2	UDP data transmissions are port-to-port
3	UPD data packets are not (necessarily) received in orderly manner
4	UDP dropped packets are not resent
5	UDP packets are composed of a source port, a destination port, and the payload.
6	UDP dropped packets are not received at the destination port
7	UDP received packets have necessarily been sent from a source port
8	UDP sent packets are either received at the destination port or they get dropped
9	UDP transmissions are stateless

UDP's Mathematical Model

```
machine udp0 sees ctx0

variables port data datalog sent received dropped

invariants
  @inv1 port ⊆ PORT
  @inv2 data ⊆ DATA
  @inv3 sent ⊆ data
  @inv4 received ⊆ sent
  @inv5 dropped ⊆ sent
  @inv6 sent = received ∪ dropped
  @inv7 datalog ∈ (port × ℙ(data)) ↔ (port ↔ ℙ(data))

events
  event INITIALISATION
    then
      @init0 port ≔ ∅
      @init1 data ≔ ∅
      @init2 sent ≔ ∅
      @init3 received ≔ ∅
      @init4 dropped ≔ ∅
      @init5 datalog ≔ ∅
  end

  event transmit_packet
    any source destination packet dropset
    where
      @grd1 source ∈ port ∧ destination ∈ port
      @grd2 packet ⊆ (data\sent)
      @grd3 dropset ⊆ packet
    then
      @act1 data ≔ data ∪ packet
      @act2 sent ≔ sent ∪ packet
      @act3 received ≔ received ∪ (packet ∖ dropset)
      @act4 dropped ≔ dropped ∪ dropset
      @act5 datalog ≔ datalog  ∪ { (source ↦ packet) ↦ {destination↦packet} }
  end

end
```