**Short documentation on Real analysis**

**Question**

**How to create real number from scratch?**

**What is an ordered field ?**

In order to describe ordered filed first we understand what is field and what is ordered relation

What is a field?

It is set f with two binary operations -> Addition , Multiplication

Any set which satisfy asscotivity, commutativity, distributy,commutativity, inverse then such called call set of field OR we can say these are the property of field set

Example: Set of rational number, Set of Real numbers

**Answer:**

**A field(f + .) is an ordered field if the following properties are satisfied**

1. **Trichotomy property**
2. **Transitive property**
3. **Addition composition**
4. **Multiplication composition**

Some other points:

Supremum: In simple words least upper bound is called supremum

Infimum: Greatest lower bound

**complete** **ordered field:**

**When an ordered field is bounded above and bounded below. In other words**

**When supremum and infimum values are coming in ordered field then we will say that field is complete ordered field**

Construction of real number

1. Synthetic approach (Reference from wikipedia)

Synthetic approach defined real number system as a **complete** **ordered field**

Such that real number system consist of set R two distinct elements 0 and 1 of **R**, two [binary operations](https://en.wikipedia.org/wiki/Binary_operation) + and × on **R.**

1. Cauchy Sequence

Whats is cauchy sequence ?

A sequence is said to be a cauchy sequence for every epsilon greater than zero

There is a positive integer m *m*, *n* > *N* then |*am*- *an*| < *ε*. For all m>n

-------------------------------------------------------------------------

First we will construct Natural Numbers:

Constructing Real numbers with Peano’s Axiom

Axioms:

1. There exist a natural number 1
2. If a is a natural number, then its successor s(a) = a+1 is also a natural number.
3. 1 is not the successor is any natural number
4. If two numbers have the same successor then they must be the same number
5. Given a set S, if 1 belongs to S and contains the successor of any number in S then S must contain all of the natural numbers.

----------------------------------

**Hol Light**

1.Referencing link for HOL LIGHT -><https://github.com/jrh13/hol-light/>

2. Getting errors -> Cannot find file camlp5o.cma and Cannot find file /Users/craftandcode/pa\_j.cmo.

3.Resolved above by using

#require “camlp5” in utop

And running utop from cloned hol-light folder

4. HOL Basics

a.Terms are enclosed in back quotes “`” ex. `x+1`

b.subst will be used to replace a term by another

ex. subst [`y + 2`,`x:num`] `x + 5 \* x`;;

c.Theorem has type “thm”

eg1. Checking the reflexivity of equality operator

# REFL `x:real`;;

returns -> val it : thm = |- x = x

eg2. Assume is used to assume a statement

# ASSUME ` x = 2`;;

returns -> val th2 : thm = x = 2 |- x = 2

eg3. INST is used to instantiate a variable in a theorem to some value

let th3 = INST [`0`,`x:num`] th1;;

returns -> val th3 : thm = |- 0 + y = 0 <=> 0 + y = 0

eg4. concl is used to conclude a theorem

concl th1;;

returns -> val it : term = `x + y = 0 <=> x + y = 0`

Some Issues faced:

6/01/2020

Some basic definition:

Least element: Let A is subset of Q where Q is a set of all Rational number, r belongs to Q then r is said to be least element of A only if

1. r belongs to A
2. r<= x , for all x belongs to A

Largest element: Let A is subset of Q where Q is a set of all Rational number, u belongs to Q then u is said to be Largest element of A only if

1. u belongs to A
2. u>= x, for all x belongs A

(This definition is in my own words may be not like standard definition)

**Dedekind Cuts**: Since the set of rational is an ordered field so we can arrange this set of rational number Q on number line. Suppose we mark a cut P on that number line such that it divides number line in two parts or in two sub set of Q.

What we will have after dividing the number line:

1. There will be two part one on left hand “L” called lower class element and other on Right hand side “U” called Upper class elements.
2. Lower class “L” elements are smaller that Upper class elements “U”.
3. There are two possibilities for cut “P”. It could be Rational number or irrational number.
4. If P is irrational number then all rational either belongs to “L” or belongs “U” and P itself become the largest element of U
5. If cut P is a rational number then means P is in the form of p/q

Then P belongs to “U”. Means P will be element of Right hand set

**Standard Definition from above explanation**

A Dedekind cut x = (L, U)in Q is a pair of subset L,U of Q satisfying the following condition

1. L union U = Q , L intersection U = {}, L and U are non empty sets
2. If l belongs L and u belongs U then l<u
3. L will not have the largest element

Hol Light Study:

**Ref. Book -> Hol Light (John Horrison)**

**Derived rules**

Summary -> Theorem that needs to be proved only by rearrangement/simplification of terms can be proved by ARITH\_RULE

Eg. # ARITH\_RULE `(a\*x+b\*x) = x\*(a+b)`;;

Returns -> val it : thm = |- a \* x + b \* x = x \* (a + b)

**Propositional Logic**

Notations used

|  |  |  |
| --- | --- | --- |
| ⊥ | F | Falsity |
| ⊤ | T | Truth |
| ¬ | ̃ | Not |
| ∧ | /\ | And |
| ∨ | \/ | Or |
| ⇒ | ==> | Implies (‘if ...then ...’) |
| ⇔ | <=> | Iff (‘...if and only if ...’) |

A => B is ¬A ∨ B

A ⇔ B is (A => B) ∧ (B => A)

To get the precedence and associativity of an infix operators .

1.infixes();;

It will give all the infix operators along with precedence and associativity

2.get\_infix\_status "op”;;

It will give the precedence and associativity of operator “op”

Eg . get\_infix\_status “==>”;;

Returns -> val it : int \* string = (4, "right")

We can make any other symbol as infix or can change precedence existing infixes using

parse\_as\_infix(“op”,(precedence,”Associativity”))

Eg. parse\_as\_infix(“<>”,(12,”right”));

**Tautology** :- A tautology is a formula or assertion that is true in every case.

Eg. x = y or x<> y

Eg."The ball is all green, or the ball is not all green"

Ref-><https://en.wikipedia.org/wiki/Tautology_(logic)>

TAUT will prove the tautologies directly

Eg. TAUT `p \/ ~p `;;

Returns-> val it : term = `p \/ ~p`

Eg. TAUT `p \/ p `;;

Returns -> Exception: Failure "TAC\_PROOF: Unsolved goals".

Some other examples

1.De Morgan’s law

a. TAUT `~(p /\ q) <=> ~p \/ ~q`;;

val it : thm = |- ~(p /\ q) <=> ~p \/ ~q

b. TAUT `~(p \/ q) <=> ~p /\ ~q`;;

val it : thm = |- ~(p \/ q) <=> ~p /\ ~q

2.Peirce’s Law

a. TAUT `((p ==> q) ==> p) ==> p`;;

val it : thm = |- ((p ==> q) ==> p) ==> p

3. TAUT `x < 1 /\ y > 0 ==> x < 1`;;

val it : thm = |- x < 1 /\ y > 0 ==> x < 1

4. TAUT `0 < x /\ x < 7 ==> 1 <= x /\ x <= 6`;;

Exception: Failure "TAC\_PROOF: Unsolved goals”.

Can be resolved by using ARITH\_RULE which analyses arithmetic content

So ARITH\_RULE `0 < x /\ x < 7 ==> 1 <= x /\ x <= 6`;; will work here

07/01/2020

**Equations and Functions**

HOL Terms types

1.Variable -> `x:bool` , `n:num`

2.Constants -> `T` (True) , `F` (False)

3.Applications

4.Abstractions

**Applications** -> Application of a function to an argument , which is a composite term built from subterms for the function and argument.

Eg. `~p` is an application of the constant denoting the negation operator to a Boolean variable

We use customary jargon *rator* (ope-gator = function ) and *rand* (ope-rand = argument) for two components of an application.

eg. rator `~p`;;

returns -> val it : term = `(~)`

eg. rand `~p`;;

returns -> val it : term = `p`

**Curried functions** -> Function may take one argument and yield another function and this can be exploited to get the same effect.

eg. a+b can be represented as ((+) a)(b)

eg. let successor = (+) 1;;

successor 4;;

returns -> val it : int = 5

Function application associates to the left, so *f a b* means *(f a) b*

**Pairing** -> Complex terms can be represented as for an example

eg. `(==>) ((=) ((+) x y) z) (P z)`;;

returns -> val it : term = `x + y = z ==> P z`

Ordered Pair can be constructed by binary infix operator “,”

eg. `1,2`

-> val it : term = `1,2`

type\_of it;;

-> val it : hol\_type = `:num#num`

mk\_comb -> builds an application out of function and argument, taking pair of arguments.

eg. mk\_comb(`(+) x`,`y:num`);;

—> val it : term = `x + y`

CONJ\_PAIR -> breaks a conjunctive theorem into a pair of theorems

eg. CONJ\_PAIR(ASSUME `p /\ q`);;

—> val it : thm \* thm = (p /\ q |- p, p /\ q |- q)

MK\_COMB -> takes a pair of theorems

eg. (ASSUME `(+) 2 = (+) (1 + 1)`,ARITH\_RULE `3 + 3 = 6`);;

—> val it : thm \* thm =

((+) 2 = (+) (1 + 1) |- (+) 2 = (+) (1 + 1), |- 3 + 3 = 6)

MK\_COMB it;;

—> val it : thm = (+) 2 = (+) (1 + 1) |- 2 + 3 + 3 = (1 + 1) + 6

fst and snd are used to select first and second components from an ordered pair

eg. fst(1,2);;

—> val it : int = 1

snd(2,3);;

—> val it : int = 3

**Equational Reasoning** -> Equations are fundamental in HOL

eg. `p /\ x = 1 <=> q`;;

-> val it : term = `p /\ x = 1 <=> q`

Above example will be parsed as (p (x = 1)) q without bracketing

eg. mk\_eq(`1`,`2`);;

—> val it : term = `1 = 2`

dest\_eq it;;

—> val it : term \* term = (`1`, `2`)

lhs `1 = 2`;;

—> val it : term = `1`

rhs `1 = 2`;;

—> val it : term = `1 = 2`

Three fundamental properties of equality relation

1.Reflexive -> t = t

2.Symmetric -> s = t then t = s

3.Transitive -> s = t and t = u then s = u

eg. REFL `1`;;

—> val it : thm = |- 1 = 1

SYM (ARITH\_RULE `1 + 1 = 2`);;

—> val it : thm = |- 2 = 1 + 1

TRANS (ARITH\_RULE `3 - 1 = 2`) it;;

—> val it : thm = |- 3 - 1 = 1 + 1