**Listing(funcion).**

**Make.**

**\+ --> Negación de una función**

**Función (+/-/? X)**

+ -> La variable debe tener un valor

- -> La variable debe estar vacia

? -> La variable puede estar vacia o no

(SOLO SE PONE EN LA DOCUMENTACION)

-git commit -a -m -> Hace un commit del documento para que pueda subirse a la repo

**Principio de inducción matemática:**

queremos probar que P es cierta para todo elemento de S

S - Sea ordenable

- Existe un primer elemento n0

1) P(n0) 🡪 (caso base)

2) P(n-1) -> P(n)

n>n0

**CLAUSULAS DE HORN**

1. a˄b → c - C. de Horn

2. a → b˄c No es C. de Horn => Equivalencia {a → b, a → c

3. a˅b → c No es C. de Horn => Equivalencia {a → c, b → c

4. a → b˅c No es C. De Horn

**SECCIÓN 1. UNIFICACIÓN**

Diremos que dos términos (Cualquier cosa) unifican:

1) Si no tienen variables, unifican si son idénticos.

Ej:

1=1 – true

1+1 = 1+1 – true

1+1 = 2 – false

1+1 = +(1,1) – true (Notación infija = Notación prefija)

1+ (1+1) = (1+1) +1 – false

(1+1) = 1+1 – true (El paréntesis no altera nada)

2) Si tienen variables, unifican si es posible encontrar una sustitución de las variables de forma que lleguen a ser idénticos.

Ej:

A=1. – A=1. ¡OJO! Si hay un punto, esa variable ya “ha acabado”, podemos asignarle otro valor

A=1+1 – A=1+1.

A=2 , A=1+1. – false.

**Operadores**

**”is” evaluador de expresiones aritméticas:**

X is 1+1.

X = 2

**“=:=” evaluador y comparador de expresiones aritméticas:**

X =:= 1+1. – ERROR

2 =:= 1+1. – true

**SECCIÓN 2. TIPOS DE DATOS**

**2.1 Listas**

Notaremos las listas entre corchetes, separando los elementos por comas. Éstas son algunas características de las listas en Prolog:

Las listas contienen términos (variables, constantes o estructuras) en general, con lo que no se restringe a que contengan valores de uno de esos tipos en concreto o con una notación uni-forme (sólo números, sólo variables, etc.). Es posible anidar listas.Notaremos la lista vacía (“[]”) con un corchete de apertura y otro de cierre.

Ejemplos de listas:

[1,2,3,4]

[]

[a,b,c,d]

[1,’a’,b,X,[1,2,3]]

[[1,2],[3,4],[5,6]]

[se\_riegan(Plantas),[3,4],pedro]

**Operador “|”**

Utilizaremos el operador “|” (barra vertical), para separar el primer elemento de una lista del resto. Este operador es muy útil para construir predicados recursivos y lo utilizaremos frecuentemente.

Ejemplos:

?- [Cabeza | Resto ] = [1,2,3,4].

Cabeza = 1,

Resto = [2, 3, 4]

2 ?- [Cabeza|Resto] = [1]

Cabeza=1

Resto=[]

3 ?- [C1,C2|Resto] = [1,2,3]

C1=1

C2=2

Resto=[3]

**Ejercicio 1 listas**

**/\***

num\_elem(+Lista, -Num)

es cierto si Num unifica con el número de elementos de la lista Lista

**\*/**

num\_elem( [ ] , 0 ). %🡪 caso base (lista vacia con 0 elementos)

%num\_elem( lista, ) :- num\_elem( lista -1, )

num\_elem( [Cabeza | Resto ] , R2 ) :- num\_elem( Resto, R ) ,

R2 is R +1.

**Concatenar Listas**

**Aritmética de peano**

Solo hay un solo símbolo (0)

Un operador siguiente(\_) -> s(\_)

0-0

1 – s ( 0 )

2 – s ( s ( 0 ) )