Manual de Usuario

Problema 1.

Los hechos para definir el árbol genealógico son los siguientes.

```
hijo(persona1, persona2).
padre/madre(persona1, persona2).
pareja(persona1, persona2).
hermano/hermana(persona1, persona2).
imprimir_arbol(persona1).
```

Para poder obtener a los hijos de una persona que sea padre la regla es la siguiente:

```
hijos(X,Y) :-
   findall(Hijo, hijo(Hijo,X),Y).
```

Para saber si una persona tiene pareja o no, se tiene la siguiente regla:

```
tiene_pareja(X,Y) :- pareja(Y,X) ; pareja(X,Y).
```

Para poder imprimir el arbol se tienen estas reglas:

```
imprimir_arbol(X) :-
        tiene_pareja(X,Y),
        atom(Y) ->
            write(X),write(' - '),write(Y),
            nl,
            hijos(X,Z),
            list_to_set(Z,H),
            imprimir_hijos(H),
            nl
        ;
            tab(4),write(X).
```

La manera en que funciona es la siguiente, se obtiene la pareja si es que la tuviese, si tiene la pareja, es posible que tenga hijos, por lo que se obtienen los hijos y se llama una regla que se llama imprimir_hijos, que lo que hace es que recorre uno a uno la lista, y esa lista vuelve a llamar nuevamente a imprimir_arbol.

Para obtener el asesion se obtienen las reglas, si el abuelo de la persona es Bruce, su primo es Clark y su tia Pepper que es la esposa de Barry.

```
asesino(X) :- es_abuelo(X,bruce),tio(X,pepper),primo(X,clark),X==mary.
```

Problema 3.

Reverso de Lista:

El reverso de la lista se obtiene recorriendo la lista uno a uno y usando un acumulado que es el que va guardando la lista.

```
reversaAcumulador([H|T],A,R):-  reversaAcumulador(T,[H|A],R).
```

```
reversaAcumulador([],A,A).

reversa(L,R):-  reversaAcumulador(L,[],R).
```

Palindroma:

Para saber si una lista es palindroma se utiliza la misma función de reverso, con la diferencia de que en lugar de obtener una lista, enviamos la misma lista, esto para saber si ambas listas al transformarla, es la misma.

```
palindroma(L) :- reversa(L,L).
```

Duplicar Lista:

Para poder duplicar esta lista se recorre uno a uno cada uno de los elementos de la lista, y se utiliza una funcion llamada atom_concat, que recibe dos atomos y los junta.

```
duplicar1([],[]).
duplicar1(L,D) :- L=[H|T],atom_concat(H, H, K),D=[K|Y],duplicar1(T,Y).

duplicar(L,D) :- duplicar1(L,D).
```

Insertar a Posición:

same_length, lo que haces es ver si ambos tienen el mismo tamaño, luego adjuntando a una lista, para luego comparar si el tamaño de la lista es igual a la posición que se esta mandando, si es así se inserta en esa posición.

```
insertar(Val,Pos,L,R) :-
    same_length([Val|L],R),
    append(A,L0,L),
    length(A,Pos),
    append(A,[Val|L0],R).
```

Dividir:

Se van colocando uno a uno en las listas A y B, obteniendo el tamaño de estas, para luego comparar si tienen el mismo tamaño. si es así se corta el flujo.

```
dividir(L, A, B) :-
    append(A, B, L),
    length(A, O),
    length(B, N),
    ((O-1)=:=N;(O+1)=:=N;O=:=N),
    !.
```

Problema 4.

Para resolver el ultimo problema se obtienen cada una de las posiciones, y con la función different si todos los elementos son distintos, eso se hace fila por fila, columna por columna, luego cuadrante por cuadrante, superior izquierdo, superior derecho, inferior izquierdo e inferior derecho, si todas esas se cumplen, el sudoku esta resuelto.