Resolución de sudokus en PROLOG

Antonio José Morano Moriña.

Para la realización de esta parte práctica de la asignatura, empezaremos explicando que es un sudoku y por que prolog es capaz de resolverlos:

Definicion de sudoku:

Un sudoku es un juego matemático que se inventó a finales de la década de 1970, se suelen estructurar en cuadrículas divididas en cajas de 3x3 celdas en las que hay algunos números escritos de antemano. Para jugar, simplemente se deben rellenar las celdas en blanco de tal forma que cada fila, columna y caja de 3x3 no tenga números repetidos.

Solución en Prolog:

Prolog esta muy bien preparado para resolver problemas combinacionales como el caso de los sudokus, vamos a poner el código usado para la resolución de este problema y procederemos a explicarlo paso a paso:

Código explicado línea a línea:

```
:- use_module(library(clpfd)).
```

Empezamos incluyendo la librería CLPFD, que nos permite asignar dominios a nuestras listas

```
sudoku(Rows) :-

length(Rows, 9), maplist(same_length(Rows), Rows),
```

Indicamos el tamaño del tablero

```
append(Rows, Vs), Vs ins 1..9,
```

Usamos **append** para insertar dominios a nuestras listas , un dominio es como una variable pero sin un valor concreto pero con un rango especifico de valores, los cuales se enccuentran entre 1 y 9 gracias al udo de **ins** de la librería CLPFD.

```
maplist(all_distinct, Rows),
```

Con **maplist**, llamamos al predicado **all_distinct** (de la librería CLPFD) para todas las listas dentro de la lista filas, esto nos asegura que no se repitan valores, como mandan las reglas de los sudokus.

```
transpose(Rows, Columns),
```

Con **transpose** , transponemos filas con columnas

maplist(all_distinct, Columns),

Hemos hecho lo mismo que antes otra vez , con esto hacemos que no se repitan los numeros en las columnas.

Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],

blocks(As, Bs, Cs),

blocks(Ds, Es, Fs),

blocks(Gs, Hs, Is).

Convertimos las filas en bloques de 3 x 3

maplist(label, Rows).

Volvemos a usar **maplist** , para llamar a **label** sobre Rows, lo que asegura que todos los dominios tienen un valor concreto, por si acaso se presenta pas de una solución al sudoku dado.

blocks([], [], []).

Indicamos que en caso de recibir tres listas vacías , nuestro predicado devolverá true.

blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-

Esta es la regla que usará prolog cuando no le pasemos listas vacías

all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),

Una vez mas , usamos **all_distinct** para asegurarnos de que no se repite un mismo numero dentro de los bloques 3x3

blocks(Ns1, Ns2, Ns3).

Recursion sobre el resto de la lista

Anotaciones personales:

1.- Como hemos podido ver a lo largo del código, la mayor parte del ejercicio se resuelve gracias a **all_distinct** que es donde prolog unifica las variables dentro de las filas , columnas y bloques con numeros del 1 al 9, dando la solución al sudoku

2.- El sudoku en la vida real es como una especie de matriz, por lo que por simplicidad, dentro de nuestro código esta implementado como una lista de prolog, una lista plana de 81 elementos (9x9), la lista contendrá inicialmente los numeros iniciales del tablero y los huecos donde no tenemos solucion.

3.- Sabemos que los numeros que nos dan de inicio no van a cambiar nunca , por lo que ya sabemos que son parte de la solución final.

Código sudoku.pl:

```
:- use_module(library(clpfd)).

sudoku(Rows) :-
    length(Rows, 9), maplist(same_length(Rows), Rows),
    append(Rows, Vs), Vs ins 1..9,
    maplist(all_distinct, Rows),
    transpose(Rows, Columns), maplist(all_distinct, Columns),
    Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],
    blocks(As, Bs, Cs),
    blocks(Ds, Es, Fs),
    blocks(Gs, Hs, Is).


blocks([], [], []).
blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-
    all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),
    blocks(Ns1, Ns2, Ns3).
```

Ejemplos de ejecución:

```
?- Puzzle = [
| [8,_,_,_,_,_,_,_,_],
| [_,_,3,6,_,_,_,_,_],
| [_,7,_,_,9,_,2,_,_],
```

| [_,5,_,_,_,7,_,_,_],

| [_,_,_,_,4,5,7,_,_],

| [_,_,_,1,_,_,_,3,_],

| [_,_,1,_,_,_,_,6,8],

| [_,_,8,5,_,_,_,1,_],

| [_,9,_,_,_,_,4,_,_]

|],

| Puzzle = [A,B,C,D,E,F,G,H,I],

| time(sudoku([A,B,C,D,E,F,G,H,I])).

% 934,447 inferences, 0.154 CPU in 0.155 seconds (100% CPU, 6051414 Lips)

Puzzle = [[8, 1, 2, 7, 5, 3, 6, 4|...], [9, 4, 3, 6, 8, 2, 1|...], [6, 7, 5, 4, 9, 1|...], [1, 5, 4, 2, 3|...], [3, 6, 9, 8|...], [2, 8, 7|...], [5, 2|...], [4|...], [...|...]],

N1 = [8, 1, 2, 7, 5, 3, 6, 4, 9],

N2 = [9, 4, 3, 6, 8, 2, 1, 7, 5],

N3 = [6, 7, 5, 4, 9, 1, 2, 8, 3],

N4 = [1, 5, 4, 2, 3, 7, 8, 9, 6],

N5 = [3, 6, 9, 8, 4, 5, 7, 2, 1],

N6 = [2, 8, 7, 1, 6, 9, 5, 3, 4],

N7 = [5, 2, 1, 9, 7, 4, 3, 6, 8],

N8 = [4, 3, 8, 5, 2, 6, 9, 1, 7],

N9 = [7, 9, 6, 3, 1, 8, 4, 5, 2] .

En la ejecución he añadido el predicado 'sudoku' dentro de 'time' para que muestre cuanto tarda la resolución.

Bibliografía:

http://programmablelife.blogspot.com/2012/07/adventures-in-declarative-programming.html

https://stackoverflow.com/questions/44661508/prolog-solve-sudoku

https://www.metalevel.at/sudoku/