Primera práctica (Programación Lógica Pura)

Alejandro Náger Fernández-Calvo (c200070)

Código empleado en la práctica:

A continuación se encuentra el código comentado con un lenguaje natural para tartar de explicar cada parte lo mejor posible.

```
:- module(_,_,[]).
author_data('Nager','Fernandez-Calvo','Alejandro','C200070').

%colors && rules
color(o).
color(x).

rule(o,o,o,_,o). % regla nula
rule(x,o,o,r(A,_,_,_,_,_,_),A) :- color(A).
rule(o,x,o,r(_,B,_,_,_,_,_),B) :- color(B).
rule(o,o,x,r(_,_,C,_,_,_,_),C) :- color(C).
rule(x,o,x,r(_,_,_,D,_,_,_),D) :- color(D).
rule(x,x,o,r(_,_,_,_,E,_,_),E) :- color(E).
rule(o,x,x,r(_,_,_,_,_,F,_),F) :- color(F).
rule(x,x,x,r(_,_,_,_,_,_,G),G) :- color(G).

%predicados auxiliares
my_append([], L, L).
my_append([X|Xs], L, [X|Ys]) :-
    my_append(Xs, L, Ys).

my_member(X, [_ | T]) :-
    my_member(X, T).

%is_valid_state/1
is_valid_state(State) :- %verificar si un estado empieza y termina por
celulas blancas
    my_append([o|_], [o], State).

%addIni/3
addIni(E,[],[E]). %anadir elementos a una lista por el principio
addIni(Elemento, [Cabeza | Cola], [Elemento, Cabeza | Cola]) :-
    addIni(Elemento, Cola, _).

%addEnd/3
```

```
addEnd(E,[],[E]). %anadir elementos a una lista por el final
addEnd(Elemento, [Cabeza | Cola], [Cabeza | ColaConElemento]) :-
    addEnd(Elemento, Cola, ColaConElemento).

%cells/3
cells([], _, []). %caso base, lista vacia
cells([X], _, [X]). %caso base, lista con un solo elem
cells(State, Rule, NewState) :-
    is_valid_state(State), %comprobamos si el estado inicial es valido
    addIni(o, State, NewState2), %anadimos dos celulas blancas al
principio y otras dos al final para aplicar las reglas sobre un nuevo
automata. Al coger las células de 3 en 3 obtendremos su evolucion con
los dos nuevos estados.
    addIni(o, NewState2, NewState3),
    addEnd(o, NewState3, NewState4),
    addEnd(o, NewState4, NewState5),
    use_rule(NewState5, Rule, NewState6), %aplicamos las reglas con
una func. auxiliar
    quitarBlancos(NewState6, NewState).

%quitarBlancos/2
quitarBlancos(List, Res) :-
    my_append(Res, [_,_], List).

%use_rule/3
use_rule([],_,[]).
use_rule([X],_,[X]).
use_rule([X,Y],_,[X,Y]).
use_rule([A,B,C|Rest], Rule, [D|NewState]) :-
    rule(A, B, C, Rule, D), %aplicamos reglas
    use_rule([B,C|Rest], Rule, NewState). %volvemos a llamar pero sin
el primer elemento (A)

%evol/3
evol(0, _, [o, x, o]). %caso base
evol(s(N), RuleSet, Cells) :-
    cells(NewState, RuleSet, Cells), %obtenemos nuevo estado
    evol(N, RuleSet, NewState). %llamada recursive con el nuevo estado
```

```
%steps/2
steps(Cells, N) :-
    evol(N, _, Cells). %sabiendo que el caso base es [o,x,o] le
pasamos la evolucion final y nos devuelve N

%ruleset/2
ruleset(Rule, Cells) :-
    evol(N, Rule, Cells).
```

Consultas realizadas con el programa y respuestas obtenidas para dichas consultas

A continuación, podemos ver las distintas pruebas realizadas para asegurar un mínimo funcionamiento del código.

```
?- cells([o,x,o], r(x,x,x,o,o,x,o), Cells).

Cells = [o,x,x,x,o] ? ;

no
?- cells([o,x,x,x,o], r(x,x,x,o,o,x,o), Cells).

Cells = [o,x,x,o,o,x,o] ? ;

no
?- cells([o,x,x,o,o,x,o], r(x,x,x,o,o,x,o), Cells).

Cells = [o,x,x,o,x,x,x,x,o] ? ;

no
?- cells([o,x,x,o,x,x,x,x,o], r(x,x,x,o,o,x,o), Cells).

Cells = [o,x,x,o,o,x,o,o,o,x,o] ? ;

no
?- cells([o,x,o], r(o,x,o,x,o,x,o), Cells).
```

```
Cells = [o,o,x,o,o] ? ;

no
?- cells([o,x,o,o,o,o,x,x,x,o,o,x,o,x,o], r(o,x,o,x,o,x,o), Cells).

Cells = [o,o,x,o,o,o,o,x,o,o,o,o,x,x,x,x,o] ? ;

no
?- evol(N, r(x,x,x,o,o,x,o), Cells).

Cells = [o,x,o],
N = 0 ? ;

Cells = [o,x,x,x,o],
N = s(0) ? ;

Cells = [o,x,x,o,o,x,o],
N = s(s(0)) ? ;

Cells = [o,x,x,o,x,x,x,x,o],
N = s(s(s(0))) ? ;

Cells = [o,x,x,o,o,x,o,o,o,x,o],
N = s(s(s(s(0)))) ? ;

Cells = [o,x,x,o,x,x,x,x,o,x,x,x,o],
N = s(s(s(s(s(0))))) ?

yes
?- steps([o,x,o], N).

N = 0 ?

yes
?- steps([o,x,x,x,o], N).

N = s(0) ?

yes
```

```
?- steps([o,x,x,o,o,x,o], N).

N = s(s(0)) ?

yes
?- ruleset(RuleSet, [o,x,x,o,o,x,o,o,o,o,x,o,o,x,o]).

RuleSet = r(x,x,x,o,o,x,o) ?

yes
```