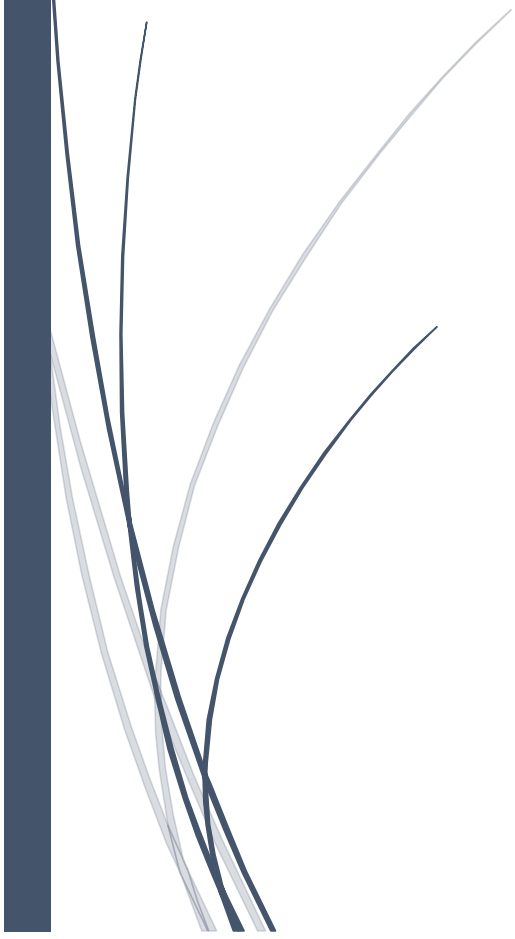
19-4-2023

MEMORIA

Code.pl

SERGIO HERAS ÁLVAREZ (C20M025)
GRADO MATEMÁTICAS E INFORMÁTICA (UPM)



Código

```
:- module(_,_,[assertions,regtypes]).
author_data('Heras', 'Alvarez', 'Sergio', 'C20M025').

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).

my_append([], List, List).
my_append([Head | Tail1], List, [Head | Tail2]) :-
    my_append(Tail1, List, Tail2).


evolve([A, B, C | Rest], Rules, [NewColor | NewState]) :-
    rule(A, B, C, Rules, NewColor),
    evolve([B, C | Rest], Rules, NewState).
evolve([A, B], Rules, [NewColor]) :-
    rule(A, B, o, Rules, NewColor).

valid_state([o | Rest]) :-
    my_append(_, [o], Rest).

cells(State, Rules, NewState) :-
    valid_state(State),
    my_append([o, o | State], [o], ExtendedState),
    evolve(ExtendedState, Rules, NewState).

evol(0, _, [o, x, o]).
evol(s(N), RuleSet, Cells) :-
    evol(N, RuleSet, PrevCells),
    cells(PrevCells, RuleSet, Cells).

steps([o,x,o],0).
steps(Cells,s(N)):-
    evol(N,_,PrevCells),
    cells(PrevCells,_,Cells).

ruleset(_, [o, x, o]).
ruleset(RuleSet, Cells) :-
    steps(Cells,s(N)),
    evol(N, RuleSet, PrevCells),
    cells(PrevCells, RuleSet, Cells).
```

Explicaciones / razonamientos

- **my_append/3:**

Es un predicado implementado para simular el funcionamiento de append/3 en Prolog, que se utiliza para concatenar dos listas.

La primera regla dice que si la primera lista es una lista vacía ([]), entonces la concatenación de esa lista con cualquier otra lista es simplemente la otra lista en sí misma. Por lo tanto, la regla devuelve la segunda lista como resultado.

La segunda regla consiste en que si la primera lista no es una lista vacía, entonces la concatenación de esa lista con la segunda lista se puede obtener concatenando la cola de la primera lista (Tail1) con la segunda lista (List) y luego añadiendo la cabeza de la primera lista (Head) al principio del resultado. Esto se hace llamando recursivamente al predicado my_append/3 con la cola de la primera lista (Tail1) y la segunda lista (List), y se almacena el resultado en Tail2. Luego, la lista resultante se construye añadiendo la cabeza de la primera lista (Head) al principio de Tail2.

- **evolve/3:**

Este es un predicado que se utiliza para evolucionar el autómata celular de tres colores a través de varias etapas.

El predicado evolve/3 toma una lista de estados ([A, B, C | Rest]), un conjunto de reglas (Rules) y devuelve una nueva lista de estados ([NewColor | NewState]) basada en la aplicación de las reglas a la lista de estados original.

La primera regla dice que si la lista de estados original tiene al menos tres elementos, se aplica la regla correspondiente a los primeros tres elementos de la lista (A, B y C) para determinar el nuevo estado. La regla se aplica llamando al predicado rule/5 (enunciado), que toma los primeros tres elementos (A, B y C), las reglas (Rules) y devuelve el nuevo estado (NewColor). Luego, el predicado evolve/3 se llama recursivamente con los elementos B, C y el resto de la lista (Rest), y el nuevo estado (NewState) se agrega al principio de la lista resultante ([NewColor | NewState]).

La segunda regla dice que si la lista de estados original tiene solo dos elementos (A y B), se aplica la regla correspondiente a los primeros dos elementos y el estado "vacío" (o) para determinar el nuevo estado. La regla se aplica llamando al predicado rule/5 con A, B, o, Rules y NewColor. Luego, el nuevo estado (NewColor) se devuelve como una lista con un solo elemento ([NewColor]).

- **valid_state/1:**

La regla valid_state/1 es un predicado que se utiliza para verificar si una lista es un estado válido para el autómata celular de tres colores.

La regla valid_state/1 toma una lista de estados ([o | Rest]) y tiene éxito si la lista cumple las siguientes dos condiciones:

1)El primer elemento de la lista es (o).

2)El último elemento de la lista también es (o), lo que significa que la lista está "acotada" por estados vacíos.

La segunda condición se verifica llamando al predicado my_append/3 con dos argumentos. El predicado my_append/3 es un predicado auxiliar que se utiliza para verificar si el segundo argumento es una lista que termina en el estado "vacío" (o). La sintaxis general del predicado my_append/3 es la siguiente:

my_append(List1, List2, List1AndList2).

El predicado my_append/3 tiene éxito si la lista List1AndList2 es la concatenación de las listas List1 y List2, y la lista List2 termina en el estado "vacío" (o).

En la regla valid_state/1, se utiliza una variable anónima (_) como primer argumento de my_append/3, lo que significa que no nos importa qué elementos están en la primera lista. El segundo argumento de my_append/3 es la lista de estados Rest, y el tercer argumento es la concatenación de la primera lista (que puede ser cualquier lista) y la lista de estados Rest. Si la lista Rest termina en el estado "vacío" (o), entonces my_append/3 tiene éxito, lo que significa que la segunda condición se cumple y la lista es un estado válido.

- **cells/3:**

La regla cells/3 se utiliza para aplicar las reglas del autómata celular de tres colores a una lista de estados y obtener una nueva lista de estados.

El predicado cells/3 toma tres argumentos: una lista de estados (State), un conjunto de reglas (Rules) y una lista de estados resultante (NewState). La lista de estados resultante se obtiene aplicando las reglas del autómata celular de tres colores a la lista de estados original.

La regla cells/3 utiliza dos predicados auxiliares: valid_state/1 y my_append/3. El predicado valid_state/1 se utiliza para verificar que la lista de estados State sea un estado válido para el autómata celular de tres colores. El predicado my_append/3 se utiliza para extender la lista de estados original con dos estados "vacíos" (o) al principio y un estado "vacío" al final.

El proceso de aplicación de las reglas se realiza mediante la llamada al predicado evolve/3, que toma la lista de estados extendida (ExtendedState), el conjunto de reglas (Rules) y devuelve una nueva lista de estados (NewState). El predicado evolve/3 aplica las reglas del autómata celular de tres colores a la lista de estados extendida, y devuelve la nueva lista de estados resultante.

- **evol/3:**

El predicado evol/3 sirve para implementar la evolución del autómata celular de tres colores.

El predicado evol/3 toma tres argumentos: el número de etapas (N), el conjunto de reglas (RuleSet) y la lista de estados resultante (Cells).

La primera regla de evol/3 dice que si N es igual a 0, entonces la lista de estados resultante Cells es simplemente la lista [o, x, o]. Esto indica el estado inicial del autómata celular.

La segunda regla es más compleja. Esta regla se utiliza para calcular el estado de las celdas después de N etapas. La regla se implementa mediante la llamada recursiva al predicado evol/3 con N - 1 como el nuevo valor para N, y la lista de estados resultante de la etapa anterior (PrevCells) como el nuevo valor para State. Luego, se llama al predicado cells/3 con PrevCells, RuleSet y Cells para obtener la nueva lista de estados resultante Cells.

- **steps/2:**

El predicado steps/2 se utiliza para calcular la lista de estados de una celda después de un número dado de etapas. Toma dos argumentos: la lista de estados de la celda (Cells) y el número de etapas (N). El predicado devuelve la lista de estados resultante después de N etapas.

La primera regla de steps/2 dice que si la lista de estados de la celda es [o, x, o], y el número de etapas es 0, entonces la lista de estados resultante es [o, x, o].

La segunda regla es más compleja. Esta regla se utiliza para calcular la lista de estados resultante después de N etapas utilizando la llamada recursiva al predicado evol/3 y el predicado cells/3. La regla comienza llamando al predicado evol/3 con N como el número de etapas y _ como el conjunto de reglas (RuleSet). La variable anónima _ se utiliza para indicar que no se especifica ningún conjunto de reglas en esta llamada. El resultado de la llamada a evol/3 es la lista de estados de la celda después de N etapas (PrevCells).

Luego, se llama al predicado cells/3 con PrevCells, _ como el conjunto de reglas (RuleSet), y Cells como la lista de estados de la celda original. La llamada a cells/3 devuelve la lista de estados resultante después de N etapas.

- **ruleset/2:**

El predicado ruleset/2 se utiliza para definir el conjunto de reglas del autómata celular de tres colores. Toma dos argumentos: el conjunto de reglas (RuleSet) y la lista de estados de la celda (Cells).

La primera regla de ruleset/2 dice que si el primer argumento es una variable anónima (_), entonces el conjunto de reglas es siempre [o, x, o]. Esto indica que el conjunto de reglas es el mismo para cualquier estado de la celda.

La segunda regla es más compleja. Esta regla se utiliza para calcular el conjunto de reglas específico para un estado de la celda particular. La regla comienza llamando al predicado steps/2 con Cells como la lista de estados de la celda y s(N) como el número de etapas. La variable s(N) se utiliza para indicar que el número de etapas se calcula a partir de la longitud de la lista de estados de la celda.

Luego, se llama al predicado evol/3 con N como el número de etapas, RuleSet como el conjunto de reglas y PrevCells como la lista de estados de la celda después de N etapas. Finalmente, se llama al predicado cells/3 con PrevCells, RuleSet y Cells para obtener la nueva lista de estados resultante después de N etapas.

Consultas / Respuestas

```
?- my_append([o, x], [x, o], Result).

Result = [o,x,x,o] ?

yes
```

```
?- my_append([], [x, o], Result).

Result = [x,o] ?

yes
?- my_append([o, x], [], Result).

Result = [o,x] ?

yes
```

```
?-  evolve([o, x],r(o,o,o,x,o,x,o), Result).

Result = [o] ?

yes
```

```
?- valid_state([o,x,x,o]).

yes
?- valid_state([o,x,x,x]).

no
```

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

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

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

Cells = [o,o,o,o,o] ?

yes
```

```
?- 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))))) ? ;

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

Yes
```

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

N = s(s(s(s(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
?- ruleset(RuleSet , [o,x,x,o,o,x,o,o,o,x,o]).

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

yes
```