Autómatas Celulares Reversibles

(Programación lógica pura)

Naval Rodríguez, Ernesto

ernesto.naval@alumnos.upm.es

c200343

Índice

1. Predicados

1.1. cells/3

Avanza el estado de una cadena de células en base a un conjunto de reglas dado.

Inic -> Estado inicial (debe empezar y terminar por celulas blancas).

Rule -> Conjunto de reglas que se aplican sobre cada triada de células

Cells -> Estado resultante tras ser aplicadas las reglas sobre Inic (tener dos células más que Inic).

cells_aux/3

Lists -> Lista de listas de 3 elementos.

Rule -> Conjunto de reglas que se aplican sobre cada triada de células.

Cells1 -> Lista devuelta como resultado con las reglas aplicadas (evolución del autómata).

```
%cells/3
%Computes the state of the cells after applying the rules in a given rule set.
cells(Inic, Rule, Cells) :-
    first_white(Inic),
    last_white(Inic),
    join([o], Inic, White1),
    add_last(o, White1, White2),
    list_of_3(White2, Lists),
    cells_aux(Lists, Rule, Cells1),
    join([o],Cells1, Cells2),
    add_last(o, Cells2, Cells).

cells_aux([], _, []).
cells_aux([H|T], Rule, [Color|Cells]) :-
    take_3(H, A, B, C),
    rule(A, B, C, Rule, Color),
    cells_aux(T, Rule, Cells).
```

Cells, en primer lugar, comprueba que el estado inicial es correcto. Esto lo hace mediante los predicados auxiliares first_white y list_white. A continuación, haciendo uso de los predicados auxiliares first_white y last_white, se añaden al estado inicial dos células blancas al principio y al final. De esta forma al aplicar list_of_3 da un formato cómodo para trabajar con el autómata. Con cells_aux, entra en una llamada recursiva en la que irá aplicando las reglas (el predicado rule proporcionado en el enunciado) a cada una de las sublistas de la lista, hasta que esta lista quede vacía. Mientras se va vaciando la lista y aplicando las reglas se van añadiendo las células nuevas a una lista resultado. Finalmente, cuando finaliza la llamada recursiva y el resultado es devuelto en Cells1, se añaden las dos células blancas al comienzo y al final de la lista celular.

Tests:

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

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

no
?-
```

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

no
?-
```

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

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

no
?-
```

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

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

yes
?-
```

1.2. evol/3

Evoluciona el estado inicial [o,x,o] el numero de veces indicado con el conjunto de reglas deseado.

N -> numero de pasos de evolución del autómata

Ruleset -> Conjunto de reglas que se aplican sobre cada triada de células

Cells -> Estado Resultante

evol_aux/4

N -> numero de pasos de evolución del autómata

RuleSet -> Conjunto de reglas que se aplican sobre cada triada de células

PrevCells -> Estado previo al resultante (siendo [o,x,o] el primero).

Cells -> Estado Resultante

```
%evol/3
%Evolves the cells for a given number of steps using a given rule set.
evol(N, RuleSet, Cells) :-
    evol_aux(N, RuleSet, [o,x,o], Cells).

evol_aux(0, _, Cells, Cells).
evol_aux(s(N), RuleSet, PrevCells, Cells) :-
    cells(PrevCells, RuleSet, NextCells),
    evol_aux(N, RuleSet, NextCells, Cells).
```

Evol usa a evol_aux, llamándole con el estado inicial [o,x,o], para que el predicado recursivo llame a cells hasta que N sea 0 (caso base de la llamada recursiva).

Tests:

```
?- 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
?-
```

```
?- evol(s(s(s(0))), r(x,x,x,o,o,x,o), Cells).

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

no
?-
```

1.3. steps/2

Calcula el numero de pasos que se requieren para llegar a cierta cadena celular.

Cells -> Cadena celular

N -> Numero de pasos para llegar a esa cadena celular

```
%step/2
%Computes the number of steps required to reduce the cells to a size of 3 cells.
steps([_,_,_], 0).
steps(Cells, s(N)) :-
    del_2(Cells, ReducedCells),
    steps(ReducedCells, N).
```

steps va sacando elementos de la lista Cells de forma recursiva mediante la función auxiliar del_2 hasta que queden únicamente 3 celulas en la cadena (caso base). Cada vez que saca dos elementos añade un +1 (en peano).

Tests:

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

N = s(0) ? ;

no
?-
```

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

no
?-
```

1.4. ruleset/2

Permite saber el juego de reglas usado para un estado de una cadena celular ([o,x,o]).

RuleSet -> Conjunto de reglas que se aplican sobre cada triada de células

Cells -> Cadena celular

```
%ruleset/2
%Computes a rule set for a given initial state of cells.
ruleset(RuleSet, Cells) :-
    steps(Cells, N),
    evol(N, RuleSet, Cells).
```

ruleset aplica los predicados creados anteriormente, usando steps para saber el numero de pasos y evol para conocer el set de reglas usado.

Tests:

```
?- 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) ? ;

no
?-
```

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

RuleSet = r(x,x,x,_,x,x,x) ? ;

no
?-
```

1.5. Predicados auxiliares

Estos son todos los predicados auxiliares usados (sin tener en cuenta rule, r y los predicados auxiliares de las funciones principales ya presentados).

```
%first element is white
first_white([o|_]).
```

Comprueba que el primer elemento de la lista sea blanco

```
%last element is white
last_white([o]).
last_white([_|T]) :-
    last_white(T).
```

Comprueba que el último elemento de la lista sea blanco vaciando la lista de forma recursiva hasta llegar al último elemento.

```
%add an element to the last position of a list
add_last(X, [], [X]).
add_last(X, [H|T], [H|T1]) :- add_last(X, T, T1).
```

Añade al final de una lista un elemento (X).

```
%Takes the 3 first elements of a list
take_3([A,B,C|_],A,B,C).
```

Extrae los 3 primeros elementos de una lista (A, B, C).

```
%Appends 2 lists
join([], X, X).
join([X | Y], Z, [X | W]) :-
    join(Y, Z, W).
```

Une el contenido de la lista pasada como segundo parámetro al final de la lista pasada como primer parámetro.

```
%Creates a list of sublists of 3 elememts each out of a list
list_of_3([X,Y,Z], [[X,Y,Z]]).
list_of_3([X,Y,Z|T], [[X,Y,Z]|R]) :-
    list_of_3([Y,Z|T], R).
```

Dada una lista en el primer parámetro, se creara una lista de listas de 3 elementos en el segundo parámetro.

```
%errases the 2 first elements of a list
del_2([_,_|T], T).
```

Elimina 2 elementos de una lista

2. Código (code.pl)

```
:- module(_,_,[classic,assertions,regtypes]).
author_data('Naval','Rodriguez','Ernesto','C200343').

%hechos
color(o).
color(x).

%reglas automata
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).



%reglas auxiliares

%first element is white
first_white([o|_]).

%last element is white
```

```
last_white([o]).
last_white([_|T]) :-
   last_white(T).

%add an element to the last position of a list
add_last(X, [], [X]).
add_last(X, [H|T], [H|T1]) :- add_last(X, T, T1).



%Takes the 3 first elements of a list
take_3([A,B,C|_],A,B,C).

%Appends 2 lists
join([], X, X).
join([X | Y], Z, [X | W]) :-
   join(Y, Z, W).

%Creates a list of sublists of 3 elememts each out of a list
list_of_3([X,Y,Z], [[X,Y,Z]]).
list_of_3([X,Y,Z|T], [[X,Y,Z]|R]) :-
   list_of_3([Y,Z|T], R).

%errases the 2 first elements of a list
del_2([_,_|T], T).

%fin reglas auxiliares

%cells/3
%Computes the state of the cells after applying the rules in a given rule set.
```

```
cells(Inic, Rule, Cells) :-
   first_white(Inic),
   last_white(Inic),
   join([o], Inic, White1),
   add_last(o, White1, White2),
   list_of_3(White2, Lists),
   cells_aux(Lists, Rule, Cells1),
   join([o],Cells1, Cells2),
   add_last(o, Cells2, Cells).

cells_aux([], _, []).
cells_aux([H|T], Rule, [Color|Cells]) :-
   take_3(H, A, B, C),
   rule(A, B, C, Rule, Color),
   cells_aux(T, Rule, Cells).



%evol/3
%Evolves the cells for a given number of steps using a given rule set.
evol(N, RuleSet, Cells) :-
   evol_aux(N, RuleSet, [o,x,o], Cells).

evol_aux(0, _, Cells, Cells).
evol_aux(s(N), RuleSet, PrevCells, Cells) :-
   cells(PrevCells, RuleSet, NextCells),
   evol_aux(N, RuleSet, NextCells, Cells).

%step/2
%Computes the number of steps required to reduce the cells to a size of 3 cells.
```

```
steps([_,_,_], 0).
steps(Cells, s(N)) :-
    del_2(Cells, ReducedCells),
    steps(ReducedCells, N).



%ruleset/2
%Computes a rule set for a given initial state of cells.
ruleset(RuleSet, Cells) :-
    steps(Cells, N),
    evol(N, RuleSet, Cells).
```