Programación Lógica Pura Práctica 1:

Autómatas Celulares Reversibles

Miguel Reviriego Gine
Nº matrícula: 190180

Código y explicaciones

A continuación, se ve una imagen del código completo, el cual iré explicando por partes más adelante. El código fue desarrollado utilizando Emacs en Ubuntu, igual que las consultas realizadas para la comprobación del funcionamiento del código.

```
:- module(_,_,[]).
author_data('Reviriego','Gine','Miguel','b190180').


% Predicados color y rule del enunciado
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).

% Predicado para concatenar una lista con otra (principalmente para concatenarlo con una 'o')
concatenar([], List, List).
concatenar([Head|Tail], List, [Head|Rest]) :-
    concatenar(Tail, List, Rest).


% Predicado para ver si el último elemento es una 'o'
ultimo(X,[X]).
ultimo(X,[_|Z]) :- ultimo(X,Z).

% Predicado para ver si el primer elemento es una 'o'
primero(X,[X|_]).


% Comprueba si un estado es válido (empieza y termina con células blancas)
empieza_y_termina_con(X, Lista) :-
    primero(X, Lista), % Comprueba si X es igual al primer elemento de la lista
    ultimo(X, Lista). % Comprueba si X es igual al último elemento de la lista

% Predicado para añadir un 'o' al principio y al final de la lista dada
aniadir_o(Lista, Resultado) :-
    concatenar([o], Lista, Temporal), % Agrega 'o' al principio de la lista
    concatenar(Temporal, [o], Resultado). % Agrega 'o' al final de la lista

% Condiciom de parada, si quedan solo dos celulas que seran blancas (no aplica nada, sale de recursion)
aplicar_reglas(_, [_Last1, _Last2], []).

aplicar_reglas(Reglas, [A,B,C|Resto], [CabezaNueva|RestoNuevo]) :-
    rule(A, B, C, Reglas, CabezaNueva),
    aplicar_reglas(Reglas, [B,C|Resto], RestoNuevo).
```

```
% Predicado cells/3
cells(EstadoInicial, Reglas, EstadoFinal) :-
    empieza_y_termina_con(o, EstadoInicial),
    aniadir_o(EstadoInicial, EstadoTemporal),
    aplicar_reglas(Reglas, EstadoTemporal, FinalTemporal),
    aniadir_o(FinalTemporal, EstadoFinal).

% Definición de números de Peano, si X es un numero, s(X) tambien lo es, sirve para "contar" s(s(s(0))) = 3
peano(0).
peano(s(X)) :- peano(X).


% Predicado evol/3
evol(N, RuleSet, Cells) :-
    peano(N),
    evol_rec(N, RuleSet, [o,x,o], Cells).

% Recursivo de evol para conseguir los pasos deseados de evolucion
evol_rec(0, _, Cells, Cells).
evol_rec(s(N), Reglas, EstadoInicial, Cells) :-
    cells(EstadoInicial, Reglas, EstadoFinal),
    evol_rec(N, Reglas, EstadoFinal, Cells).

% Predicado steps/2
steps(Cells, N) :-
    empieza_y_termina_con(o, Cells),
    peano(N),
    evol(N, _, Cells).

% Predicado ruleset/2
ruleset(_,Cells):-
empieza_y_termina_con(o,Cells).
```

Predicados rule/5, r/7 y color/1: estos predicados son parte del enunciado, definen los elementos que ocupan las células, el posicionamiento de las 7 reglas (+ la nula) y el proceso de aplicación de las reglas, con el predicado rule/5 que tiene como parámetros 3 células, las reglas con formato r(A, B, C, D, E, F, G) y el estado final de la célula, que será un color.

Predicado concatenar/3: este predicado es una versión personal de append, puesto que no está permitido el uso de librerías ni de los predicados predefinidos de ISO Prolog. Su funcionamiento está pensado para el uso de la recursión para añadir un elemento X a otra lista para conseguir el objetivo final, aunque también podría utilizarse para concatenar listas y nos solo un único elemento.

Predicado aniadir_o/2: utilizando el predicado anterior concatenar/3, aniadir_o añadirá a ambos lados de una lista dada el elemento “o”, utilizando otra variable Resultado que “almacenará” la lista inicial con los elementos añadidos. El funcionamiento básicamente consiste en una aplicación de concatenar con un elemento específico “o”.

Predicados ultimo/2 y primero/2: estos dos predicados están pensados para comprobar la condición previa de que el estado inicial sea un estado válido, y una de las condiciones de que sea un estado válido es que tanto el primero como el último elemento deben ser “o”, así que eso comprueba respectivamente cada uno de los predicados, separando Head y Tail de la lista y comparando con el elemento X, únicamente usado con valor “o” para X.

Predicado empieza_y_termina_con/2: aplicación de ultimo/2 y primero/2 sustituyendo la X por “o” y comprobando para la lista. Al ser tan simple podría haber evitado usar este predicado y simplemente haber usado ultimo y primero, pero me ayudó a aclararme un poco con el código cuando me encontraba más perdido.

Predicado aplicar_reglas/3: predicado fundamental para la correcta implementación de cells/3, es básicamente la fórmula para aplicar la regla correspondiente a cada uno de los tríos y recursivamente avanzar en la lista original, rellenando una lista temporal con los estados finales después de aplicar las reglas.

Predicado cells/3: primer predicado pedido en el enunciado. Recibe los tres argumentos EsadoInicial, Reglas y EstadoFinal. Primero se comprueba que ambos estados sean válidos, el primero porque es necesario por el enunciado y el último para ahorrar todo el proceso recursivo a la memoria y descartarlo antes de hacerlo. Antes y después del proceso de aplicar las reglas, se debe añadir una “o” a los estados inicial y final; en el inicial para poder aplicar las reglas al primer elemento del estado y en el final para cerciorarse de que empiece y termine por “o”, ignorando la posibilidad de tener que reducir la lista como dice el enunciado para ahorrar dificultades. En este punto estuve bastante tiempo atrapado, ya que tenía un predicado llamado dividir_en_trios/2 codificado así:

```
dividir_en_trios([], []).
dividir_en_trios([_], []).
dividir_en_trios([_,_], []).
dividir_en_trios([X,Y,Z|Resto], [[X,Y,Z]|TramosResto]) :-
   dividir_en_trios([Y,Z|Resto], TramosResto).
```

Pero me di cuenta de que no era necesario el hecho de dividir en tríos, ya que en la propia llamada recursiva de aplicar_reglas podía llamar a los dos últimos elementos del trío actual junto con el resto de la lista, por lo que lo descarté.

Predicado peano/1: al estar prohibido el uso de aritmética de ISO Prolog, vi necesario crear este predicado que funciona con la lógica de que, si 0 es un número natural, s(0) también lo es y equivale a 1, s(s(0)) equivale a 2 etcétera.

Predicado evol/3: segundo predicado pedido en el enunciado. Recibe tres argumentos: N, RuleSet, Cells; siendo N el número de pasos de evolución desde el estado inicial [o, x, o], RuleSet las reglas que se usan y Cells el estado final al que se quiere llegar, que previamente se comprueba si empieza y termina por “o”. Evol funciona "contando"cada iteración que hace de la recursividad con el Peano de N, aplicando inicialmente las reglas de RuleSet a [o, x, o] y

después al EstadoFinal, en el que se almacena la lista después de aplicarle cells/3. Este predicado en comparación fue mucho más sencillo después de entender bien cells/3 y de tenerlo bien implementado. Parará si N es 0, devolviendo el estado final en Cells.

Predicado steps/2: tercer predicado que se pide en el enunciado. Se pide junto al predicado ruleset/2, y consiste en ver el número de pasos para llegar a un estado desde el inicial, es complementario para el predicado ruleset/2. Primero debemos comprobar si Cells es un estado válido y a continuación utilizar el predicado evol/3 para saber el número de pasos.

Predicado ruleset/2: último predicado del enunciado. Debe recibir como primer argumento un set de reglas y el segundo un estado al que se puede llegar desde el estado inicial [o, x, o]. Debe funcionar en todos los modos de llamada, es decir, debe indicar tanto los sets de reglas que hagan el autómata capaz de llegar al estado final desde [o, x, o] como los estados finales alcanzables con un set de reglas dado.

Para comprobar el correcto funcionamiento de cada uno de los predicados, realicé consultas al programa, además de las que hay en el enunciado como ejemplos probé otros valores:

Para comprobar si aniadir_o/2 funciona, hice algún test.

```
?- aniadir_o([o,x,x,o], Resultado).

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

no
```

Para comprobar si empieza_y_termina_con/2 funcionaba:

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

no
?- empieza_y_termina_con(o, [o,o,x]).

no
?- empieza_y_termina_con(o, [o,x,o]).

yes
```

Para comprobar el funcionamiento de cells/3:

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

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

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

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

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

no
```

Para comprobar el funcionamiento de evol/3:

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

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

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

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

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

```
?- evol(N, r(x,x,x,x,x,x,x), Cells).

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

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

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

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

No conseguí completar el predicado ruleset/2, por lo que no puedo añadir imágenes de pruebas del mismo.
Pero sí que conseguí que se generara un archivo PDF mediante el comando lpdoc, me surgieron bastantes problemas al intentarlo ya que no me dejaba elegir como tipo de archivo PDF y como HTML me daba un error. Al final, conseguí que se generara algo, lo adjunto a continuación, pero de todas formas está incompleto porque me seguía surgiendo el siguiente error:

```
Transcript written on code.log.
{ERROR (logged_process): process path(tex) returned code 1 (arguments: [-file-line-error-style,\nonstopmode\input code.texic], options: [show_logs(on_error),logbase(code.cachedoc/run_tex1),cwd(code.cachedoc/t
exinfo),status(_)]) }
This is TeXk, Version 3.141592653 (TeX Live 2022/dev/Debian) (preloaded format=tex)

(./code.texic (./texinfo.tex Loading texinfo package [Version 2.257]: Basics,
fonts, page headings, tables, indexing, sectioning, toc, environments, defuns,
macros, cross references,
(/usr/share/texlive/texmf-dist/tex/generic/epsf/epsf.tex
This is `epsf.tex' v2.7.4 <14 February 2011>
) paper sizes, and turning on texinfo input format.) (./code.aux)
./code.texic:7: Undefined control sequence.
l.7 @documentencoding
                     UTF-8
[1] [2] (./code.toc) [-1] [1] (./coderefs.texic (References) [2])
[./codeliindex.texic (Library/Module Index) [3] [4]) (./codepdindex.texic
(Predicate Index) [5] [6] (./code.pds)) (./codeprindex.texic (Property Index)
[7] [8]) (./codeteindex.texic (Regular Type Index) [9] [10])
(./codedeindex.texic (Declaration Index) [11] [12]) (./codecoindex.texic
(Concept Index) [13] [14]) (./codeauindex.texic (Author Index) [15] [16])
(./codeglindex.texic (Global Index) [17] [18] (./code.gls)) [19] [20] )
(see the transcript file for additional information)
Output written on code.dvi (23 pages, 28716 bytes).
Transcript written on code.log.
{ERROR (logged_process): process path(tex) returned code 1 (arguments: [-file-line-error-style,\nonstopmode\input code.texic], options: [show_logs(on_error),logbase(code.cachedoc/run_tex2),cwd(code.cachedoc/t
exinfo),status(_)]) }
```

UTF-8

code

Table of Contents

0.1 Usage and interface

- **Library usage:**
  `:- use_module(/home/miguel/code.pl).`
- **Exports:**
  - *Predicates:*
    `author_data/4`, `color/1`, `rule/5`, `concatenar/3`, `ultimo/2`, `primero/2`, `empieza_y_termina_con/2`, `aniadir_o/2`, `aplicar_reglas/3`, `cells/3`, `peano/1`, `evol/3`, `evol_rec/4`, `steps/2`, `ruleset/2`.

0.2 Documentation on exports

author_data/4: PREDICATE
No further documentation available for this predicate.

color/1: PREDICATE
No further documentation available for this predicate.

rule/5: PREDICATE
No further documentation available for this predicate.

concatenar/3: PREDICATE
No further documentation available for this predicate.

ultimo/2: PREDICATE
No further documentation available for this predicate.

primero/2: PREDICATE
No further documentation available for this predicate.

empieza_y_termina_con/2: PREDICATE
No further documentation available for this predicate.

aniadir_o/2: PREDICATE
No further documentation available for this predicate.

aplicar_reglas/3: PREDICATE
No further documentation available for this predicate.

cells/3: PREDICATE
No further documentation available for this predicate.

peano/1: PREDICATE
No further documentation available for this predicate.

evol/3: PREDICATE
No further documentation available for this predicate.

evol_rec/4: PREDICATE
No further documentation available for this predicate.

steps/2: PREDICATE
No further documentation available for this predicate.

ruleset/2: PREDICATE
No further documentation available for this predicate.

0.3 Documentation on imports

This module has the following direct dependencies:

– *Internal (engine) modules:*
`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`.

– *Packages:*
`prelude`, `initial`, `condcomp`.

References

(this section is empty)

Library/Module Index

(Index is nonexistent)

Predicate Index

Property Index

(Index is nonexistent)

Regular Type Index

(Index is nonexistent)

Declaration Index

(Index is nonexistent)

Concept Index

(Index is nonexistent)

Author Index

(Index is nonexistent)

Global Index

This is a global index containing pointers to places where concepts, predicates, modes, properties, types, applications, authors, etc., are referred to in the text of the document.