Autómatas Celulares Reversibles

Julia Serrano Arrese. C200119

Table of Contents

code

Usage and interface

- **Library usage:**
  use_module('code.pl')
- **Exports:**
  - *Predicates:*
    `rule/5`, `my_append/3`, `add_os/2`, `last_o/2`, `first_o/2`, `check_first_and_last/1`, `at_least_3_elems/1`, `sum/3`, `len/2`, `length_difference/2`, `cells_aux/3`, `cells/3`, `evol/3`, `steps/2`, `ruleset/2`.
  - *Properties:*
    `author_data/4`, `color/1`, `my_list/1`, `natural/1`, `is_state/1`.

Documentation on exports

author_data/4: PROPERTY

Usage:

Nombre y matrícula del autor de la práctica.

```
author_data('Serrano','Arrese','Julia','200119').
```

color/1: PROPERTY

Usage:

Los colores válidos para las células son:

- blanco -> o.
- negro -> x

```
color(o).
color(x).
```

rule/5: PREDICATE

Usage: `rule(X1,X2,X3,R,Y)`

`Y` es la célula resultante de aplicar el conjunto de reglas `R` de tipo r/7 a las células: `X1`, `X2` y `X3`.

```
rule(o,o,o,_1,o).
rule(x,o,o,r(A,_1,_2,_3,_4,_5,_6),A) :-
    color(A).
rule(o,x,o,r(_1,B,_2,_3,_4,_5,_6),B) :-
    color(B).
rule(o,o,x,r(_1,_2,C,_3,_4,_5,_6),C) :-
    color(C).
rule(x,o,x,r(_1,_2,_3,D,_4,_5,_6),D) :-
```

```
    color(D).
rule(x,x,o,r(_1,_2,_3,_4,E,_5,_6),E) :-
    color(E).
rule(o,x,x,r(_1,_2,_3,_4,_5,F,_6),F) :-
    color(F).
rule(x,x,x,r(_1,_2,_3,_4,_5,_6,G),G) :-
    color(G).
```

Other properties:

Test: `rule(X1,X2,X3,R,Y)`

– *If the following properties hold at call time:*

`X1=o` (`= /2`)

`X2=x` (`= /2`)

`X3=o` (`= /2`)

`R=r(x,o,x,x,x,x,o)` (`= /2`)

then the following properties should hold upon exit:

`Y=o` (`= /2`)

then the following properties should hold globally:

All the calls of the form `rule(X1,X2,X3,R,Y)` do not fail. (`not_fails/1`)

my_list/1: PROPERTY

Usage: `my_list(L)`

Predicado que comprueba si el argumento `L` es una lista.

```
my_list([]).
my_list([_1|Y]) :-
    my_list(Y).
```

Other properties:

Test: `my_list(L)`

– *If the following properties hold at call time:*

`L=[o,x,o]` (`= /2`)

then the following properties should hold upon exit:

`R=yes` (`= /2`)

then the following properties should hold globally:

All the calls of the form `my_list(L)` do not fail. (`not_fails/1`)

Test: `my_list(L)`

– *If the following properties hold at call time:*

`L=o` (`= /2`)

then the following properties should hold upon exit:

`R=no` (`= /2`)

then the following properties should hold globally:

All the calls of the form `my_list(L)` do not fail. (`not_fails/1`)

my_append/3: PREDICATE

Usage: `my_append(Xs,Ys,Zs)`

Concatena las listas `Xs` e `Ys` y devuelve el resultado en `Zs`. Se espera que `Xs` y `Ys` sean listas.

```
my_append([],Ys,Ys) :-
    my_list(Ys).
my_append([X|Xs],Ys,[X|Zs]) :-
    my_append(Xs,Ys,Zs).
```

– *Call and exit should be compatible with:*

Predicado que comprueba si el argumento `Xs` es una lista.

```
my_list([]).
my_list([_1|Y]) :-
    my_list(Y).
```

(`my_list/1`)

Predicado que comprueba si el argumento `Ys` es una lista.

```
my_list([]).
my_list([_1|Y]) :-
    my_list(Y).
```

(`my_list/1`)

Predicado que comprueba si el argumento `Zs` es una lista.

```
my_list([]).
my_list([_1|Y]) :-
    my_list(Y).
```

(`my_list/1`)

Other properties:

Test: `my_append([Xs],[Ys],[Zs])`

Caso base:

– *If the following properties hold at call time:*

`Xs=[]` (`= /2`)

`Ys=[]` (`= /2`)

then the following properties should hold upon exit:

`Zs=[]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `my_append([Xs],[Ys],[Zs])` do not fail. (`not_fails/1`)

Test: `my_append([Xs],[Ys],[Zs])`

Caso base:

– *If the following properties hold at call time:*

`Xs=[]` (`= /2`)

`Ys=[o,x,o]` (`= /2`)

then the following properties should hold upon exit:

`Zs=[o,x,o]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `my_append([Xs],[Ys],[Zs])` do not fail. (`not_fails/1`)

Test: `my_append([Xs],[Ys],[Zs])`

– *If the following properties hold at call time:*
`Xs=[o,x,o]` (= /2)
`Ys=[]` (= /2)
then the following properties should hold upon exit:
`Zs=[o,x,o]` (= /2)
then the following properties should hold globally:
All the calls of the form `my_append([Xs],[Ys],[Zs])` do not fail. (`not_fails/1`)

Test: `my_append([Xs],[Ys],[Zs])`
– *If the following properties hold at call time:*
`Xs=[o,x,o]` (= /2)
`Ys=[x,x,o]` (= /2)
then the following properties should hold upon exit:
`Zs=[o,x,o,x,x,o]` (= /2)
then the following properties should hold globally:
All the calls of the form `my_append([Xs],[Ys],[Zs])` do not fail. (`not_fails/1`)

add_os/2: PREDICATE

Usage: `add_os(List,Result)`

Agrega el elemento 'o' al comienzo y al final de la lista `List`, produciendo la lista resultante `Result`.

```
add_os(List,[o|Result]) :-
    my_append(List,[o],Result).
```

Other properties:

Test: `add_os(L,R)`
– *If the following properties hold at call time:*
`L=[x,x,x]` (= /2)
then the following properties should hold upon exit:
`R=[o,x,x,x,o]` (= /2)
then the following properties should hold globally:
All the calls of the form `add_os(L,R)` do not fail. (`not_fails/1`)

last_o/2: PREDICATE

Usage: `last_o(X,L)`

Predicado que comprueba si el cúltimo elemento de la lista `L` es `X`.

```
last_o(X,[_1|Tail]) :-
    last_o(X,Tail).
last_o(X,[X]).
```

Other properties:

Test: `last_o(X,L)`

– *If the following properties hold at call time:*
`X=o` (= /2)
`L=[x,o,x,o]` (= /2)
then the following properties should hold upon exit:
`R=yes` (= /2)
then the following properties should hold globally:
All the calls of the form `last_o(X,L)` do not fail. (`not_fails/1`)

Test: `last_o(X,L)`
– *If the following properties hold at call time:*
`X=o` (= /2)
`L=[x,o,x]` (= /2)
then the following properties should hold upon exit:
`R=no` (= /2)
then the following properties should hold globally:
Calls of the form `last_o(X,L)` fail. (`fails/1`)

first_o/2: PREDICATE

Usage: `first_o(X,L)`

Predicado que comprueba si el primer elemento de la lista `L` es `X`.

```
first_o(X,[X|_1]).
```

Other properties:

Test: `first_o(X,L)`
– *If the following properties hold at call time:*
`X=o` (= /2)
`L=[o,o,x,o]` (= /2)
then the following properties should hold upon exit:
`R=yes` (= /2)
then the following properties should hold globally:
All the calls of the form `first_o(X,L)` do not fail. (`not_fails/1`)

Test: `first_o(X,L)`
– *If the following properties hold at call time:*
`X=o` (= /2)
`L=[x,o,x]` (= /2)
then the following properties should hold upon exit:
`R=no` (= /2)
then the following properties should hold globally:
Calls of the form `first_o(X,L)` fail. (`fails/1`)

check_first_and_last/1: PREDICATE

Usage: `check_first_and_last(L)`

Predicado que verifica si el primer y último elemento de la lista `L` son 'o'.

```
check_first_and_last(I) :-
    last_o(o,I),
    first_o(o,I).
```

Other properties:

Test: `check_first_and_last(L)`

– *If the following properties hold at call time:*

`L=[o,x,o]` (= /2)

then the following properties should hold upon exit:

`R=yes` (= /2)

then the following properties should hold globally:

All the calls of the form `check_first_and_last(L)` do not fail. (`not_fails/1`)

Test: `check_first_and_last(L)`

– *If the following properties hold at call time:*

`L=[x,o,x]` (= /2)

then the following properties should hold upon exit:

`R=no` (= /2)

then the following properties should hold globally:

All the calls of the form `check_first_and_last(L)` do not fail. (`not_fails/1`)

at_least_3_elems/1: PREDICATE

Usage: `at_least_3_elems(L)`

Predicado que comprueba si una lista tiene al menos tres elementos.

```
at_least_3_elems([_1,_2,_3|_4]).
```

Other properties:

Test: `at_least_3_elems(L)`

– *If the following properties hold at call time:*

`L=[o,x,o]` (= /2)

then the following properties should hold upon exit:

`R=yes` (= /2)

then the following properties should hold globally:

All the calls of the form `at_least_3_elems(L)` do not fail. (`not_fails/1`)

Test: `at_least_3_elems(L)`

– *If the following properties hold at call time:*

`L=[o,x]` (= /2)

then the following properties should hold upon exit:

`R=no` (= /2)

then the following properties should hold globally:

Calls of the form `at_least_3_elems(L)` fail. (`fails/1`)

natural/1: PROPERTY

Usage:

Número natural.

```
natural(0).
natural(s(X)) :-
    natural(X).
```

sum/3: PREDICATE

Usage: `sum(A,B,C)`

`C` es la suma de `A` y `B` en Peano.

```
sum(0,Y,Y) :-
    natural(Y).
sum(s(X),Y,s(Z)) :-
    sum(X,Y,Z).
```

Other properties:

Test: `sum(A,B,C)`

Caso base:

- *If the following properties hold at call time:*

  `A=0` (`= /2`)

  `B=s(0)` (`= /2`)

  then the following properties should hold upon exit:

  `C=s(0)` (`= /2`)

  then the following properties should hold globally:

  All the calls of the form `sum(A,B,C)` do not fail. (`not_fails/1`)

Test: `sum(A,B,C)`

- *If the following properties hold at call time:*

  `A=s(0)` (`= /2`)

  `B=s(s(0))` (`= /2`)

  then the following properties should hold upon exit:

  `C=s(s(s(0)))` (`= /2`)

  then the following properties should hold globally:

  All the calls of the form `sum(A,B,C)` do not fail. (`not_fails/1`)

len/2: PREDICATE

Usage: `len(L,N)`

`N` es la logitud de la lista `L`.

```
len([],0).
len([_1|T],s(LT)) :-
    len(T,LT).
```

Other properties:

Test: `len(L,N)`

Caso base:

– *If the following properties hold at call time:*
`L=[]` (= /2)
then the following properties should hold upon exit:
`N=0` (= /2)
then the following properties should hold globally:
All the calls of the form `len(L,N)` do not fail. (`not_fails/1`)

Test: `len(L,N)`
– *If the following properties hold at call time:*
`L=[o,x,o,o]` (= /2)
then the following properties should hold upon exit:
`N=s(s(s(s(0))))` (= /2)
then the following properties should hold globally:
All the calls of the form `len(L,N)` do not fail. (`not_fails/1`)

length_difference/2: PREDICATE

Usage: `length_difference(I,F)`

Verifica si la longitud de la lista `F` es dos más que la longitud de la lista `I`.

```
length_difference(I,F) :-
    len(I,LI),
    len(F,LF),
    sum(LI,s(s(0)),LF).
```

Other properties:

Test: `length_difference(I,F)`
– *If the following properties hold at call time:*
`I=[]` (= /2)
`F=[o,x,x,o]` (= /2)
then the following properties should hold upon exit:
`R=no` (= /2)
then the following properties should hold globally:
Calls of the form `length_difference(I,F)` fail. (`fails/1`)

Test: `length_difference(I,F)`
– *If the following properties hold at call time:*
`I=[o,x,x,o]` (= /2)
`F=[x,x,o,o,x,o]` (= /2)
then the following properties should hold upon exit:
`R=yes` (= /2)
then the following properties should hold globally:
All the calls of the form `length_difference(I,F)` do not fail. (`not_fails/1`)

is_state/1: PROPERTY

Usage:

Comprueba que el argumento `L` es un estado válido:

- Estado es una lista
- Estado mínimo formado por 3 células -> [o,x,o]
- Estados deben empezar y terminar por células blancas

```
is_state(L) :-
    my_list(L),
    at_least_3_elems(L),
    check_first_and_last(L).
```

Other properties:

Test: `is_state(L)`

– *If the following properties hold at call time:*
`L=x` (`= /2`)
then the following properties should hold upon exit:
`R=no` (`= /2`)
then the following properties should hold globally:
Calls of the form `is_state(L)` fail. (`fails/1`)

Test: `is_state(L)`

– *If the following properties hold at call time:*
`L=[o,x]` (`= /2`)
then the following properties should hold upon exit:
`R=no` (`= /2`)
then the following properties should hold globally:
Calls of the form `is_state(L)` fail. (`fails/1`)

Test: `is_state(L)`

– *If the following properties hold at call time:*
`L=[o,x,x]` (`= /2`)
then the following properties should hold upon exit:
`R=no` (`= /2`)
then the following properties should hold globally:
Calls of the form `is_state(L)` fail. (`fails/1`)

Test: `is_state(L)`

– *If the following properties hold at call time:*
`L=[o,x,o]` (`= /2`)
then the following properties should hold upon exit:
`R=yes` (`= /2`)
then the following properties should hold globally:
All the calls of the form `is_state(L)` do not fail. (`not_fails/1`)

Test: `is_state(L)`

– *If the following properties hold at call time:*
`L=[o,x,x,o,x,x,x,x,o]` (`= /2`)
then the following properties should hold upon exit:
`R=yes` (`= /2`)
then the following properties should hold globally:
All the calls of the form `is_state(L)` do not fail. (`not_fails/1`)

cells_aux/3: PREDICATE

Usage: `cells_aux(InitialState,RuleSet,FinalState)`

Predicado auxiliar utilizado por 'cells/3', para realizar la recursividad.

En la implementación, se divide el estado inicial `InitialState` en tripletes de células consecutivas y se aplica la regla correspondiente (definida por `Ruleset`) para cada tripleta, lo que da lugar al siguiente estado (`FinalState`). Este proceso se repite hasta llegar al caso base, cuando quedan solo 2 células del estado inicial, finaliza el proceso recursivo

```
cells_aux([_2,_3],_1,Fs) :-
    Fs=[].
cells_aux([X1,X2,X3|Xs],RuleSet,[F1|Fs]) :-
    rule(X1,X2,X3,RuleSet,F1),
    cells_aux([X2,X3|Xs],RuleSet,Fs).
```

cells/3: PREDICATE

Usage: `cells(InitialState,RuleSet,FinalState)`

Realiza un paso de evolución desde `InitialState` a `FinalState` mediante la aplicación de la codificación de reglas `RuleSet`.

```
cells(I,RuleSet,F) :-
    is_state(I),
    add_os(I,I1),
    cells_aux(I1,RuleSet,F1),
    add_os(F1,F),
    is_state(I1),
    is_state(F),
    length_difference(I,F).
```

evol/3: 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.

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`, `basic_props`.

– *Packages:*

`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`.