Primera práctica (Programación Lógica Pura)

Fecha de entrega: ~~19~~23 de abril de 2023

Autómatas Celulares Reversibles

Antes de empezar la práctica es **muy importante** empezar por leer las **Instrucciones generales para la realización y entrega las prácticas**, así como la **Preguntas frecuentes sobre Ciao, Deliverit, LPdoc, etc.**, disponibles en Moodle.

Instrucciones específicas

En esta práctica ponemos en práctica algunos de los conceptos estudiados en el bloque temático de programación lógica pura. Por ello, además de las instrucciones generales, como instrucciones específicas para esta práctica se deben usar sólo **cláusulas** (reglas y hechos), **términos** (constantes, variables, estructuras), y la **unificación** (términos en los argumentos, o llamadas a `=/2`), y la declaración inicial de módulo y el hecho `author_data/4` de las instrucciones generales para prácticas. **No está permitido** utilizar los predicados predefinidos de ISO Prolog (p.ej., aritmética, predicados metalógicos, predicados de inspección de estructuras, corte, negación por fallo, etc.). Tampoco está permitido utilizar las librerías del sistema (p.ej., las de predicados sobre listas, árboles, etc.). Si fuese necesario utilizar alguno de estos predicados, debéis programarlo vosotros mismos. Por ejemplo, si se necesitase definir la pertenencia a una lista mediante el predicado `member/3`, debéis programar una versión propia del predicado `member/3` a la que deberéis dar un nombre diferente (p.ej., `my_member/3`) para evitar el uso por defecto del predicado predefinido `member/3`. Se deberá tener especial cuidado con esto, dado que hay predicados predefinidos como `length/2` (que calcula la longitud de una lista) que utilizan aritmética de ISO Prolog en lugar de aritmética de Peano (que es la que se puede y debe utilizar en este ejercicio).

1. Autómata celular (8.5 puntos, fichero `code.pl`)

Un autómata celular esta formado por una cinta infinita (por ambos extremos) que contiene células, las cuales tienen siempre un color dado, y un conjunto de reglas que determinan cómo evoluciona el estado de la cinta: cómo cambian de color sus células. La evolución del autómata consiste en aplicar las reglas a un estado de la cinta para llegar a un nuevo estado, moidficando los colores de las células. Por simplicidad, utilizaremos solo el blanco y el negro, representados por `o` y `x`, respectivamente.

El nuevo color de una célula depende solo de los colores (originales) de sus dos células vecinas, a izquierda y derecha. Gracias a esto las reglas se pueden definir a partir de solo tres células. La siguiente tabla muestra como *ejemplo* el conjunto de reglas de un autómata celular concreto:

ooo	xoo	oxo	oox	xox	xxo	oxx	xxx
o	x	o	x	x	x	x	o

La tabla representa ocho reglas (en vertical). La primera establece que una célula blanca con ambas vecinas blancas permanece blanca, la segunda que una célula blanca con la vecina izquierda negra y la derecha blanca se vuelve negra, la tercera que una célula de color negro con ambas vecinas de color blanco se vuelve blanca, y así sucesivamente.

La primera regla se conoce como la regla nula y se da por supuesto que pertenece a todo autómata celular. Además, el número de células negras en cualquier estado de una cinta es siempre finito. Gracias a estas dos propiedades, el estado de una cinta se puede representar de forma finita, sin más que mostrar la mínima secuencia de células que empieza y acaba en una célula blanca y contiene a todas las que son negras, como en:

oxxxoooxo

donde al aplicar las reglas anteriores se obtiene el nuevo estado:

oxxoxxoxoxo

Las (infinitas) células que no se muestran, a izquierda y derecha de la cinta, son todas blancas. Nótese además que (la representación de) un nuevo estado siempre contiene dos células más que el anterior y que las reglas se han aplicado a todas las células del estado inicial, incluidas las dos blancas de los extremos (y a todas las demás que no aparecen, que eran blancas y lo siguen siendo).

Codificación de las reglas del autómata. Las reglas de evolución se codifican mediante un término `r/7` en el que cada argumento contiene el color resultante para cada una de las posibles entradas (menos la regla nula que la fijamos a blanco). Es decir, codificaremos las reglas mediante el término `r(A,B,C,D,E,F,G)`, correspondiente al siguiente conjunto de reglas:

ooo	xoo	oxo	oox	xox	xxo	oxx	xxx
o	A	B	C	D	E	F	G

donde `A`, `B`, ..., deben ser colores, tal como se define en el predicado `color/1`:

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

Para consultar una regla concreta os proporcionamos el siguiente predicado que debeis incluir en vuestra solución:

```
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).
```

Por ejemplo, el conjunto de reglas del ejemplo inicial se representa como `r(x,o,x,x,x,x,o)` y la *consulta* `?- R=r(x,o,x,x,x,x,o), rule(o,x,o,R,Y)` tendrá éxito con `Y=o`.

Un paso de evolución: `cells/3` (5 puntos). Se debe programar un predicado cells/3 que acepte como primer argumento un estado inicial, como segundo argumento una codificación de reglas (término `r/7`) y como tercer argumento el estado resultante de aplicar las reglas del segundo argumento al estado del primer argumento. Ambos estados deben representarse por listas, como en `[o,x,o,x,o]` para el estado oxoxo). Nótese que un autómata no puede evolucionar si su cinta contiene tres células consecutivas a las que no se puede aplicar ninguna regla: en tal caso el predicado debe fallar (de forma finita).

Deben considerarse llamadas a `cells/3` en todos los posibles modos de llamada tal que:

- Los estados deben empezar y terminar por células blancas (`[o,x,o,x,o]` es un estado admitido mientras que `[x,o,x]` no lo debe ser).
- El estado resultante debe contener siempre dos células más que el estado inicial. Por simplicidad *no consideramos representaciones minimizadas* (es decir, si `[o,x,o]` evolucionara a `[o,o,x,o,o]` – que es equivalente – no debemos eliminar las celulas redundantes).
- El predicado debe fallar si no se cumple la especificación dada mostrada en los puntos anteriores.

Ejemplo:

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

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

no
?-
```

N pasos de evolución: `evol/3` (1 punto). Se debe implementar `evol(N, RuleSet, Cells)` que aplica `N` pasos de evolución desde el estado inicial `[o,x,o]` (usando `cells/3` con la codificación de reglas `RuleSet`) para llegar a `Cells`.

Ejemplo:

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

Descubrir autómatas: `ruleset/2` (2.5 puntos). Se debe implementar como predicado preliminar `steps(Cells,N)` (1 punto) donde `N` es el número de pasos necesarios para llegar un estado a partir de una configuración inicial de tres células (tal y como hemos definido `cells/3`)

Usando el predicado anterior, se debe definir `ruleset/2` (1.5 puntos), cuyo primer argumento debe ser una codificación de reglas y el segundo un estado al cual se puede llegar a partir del estado inicial (`[o,x,o]`). Debe funcionar en todos los modos posibles de ejecución. Nótese que a diferencia de `evol/3`, debemos fijar el espacio de búsqueda mediante `steps/2`, lo cual permitirá tanto descubrir los autómatas que llegan a una configuración dada como probar su no existencia:

```
?- 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,o,x,o,x,x,o]).

no
```

---

2. MEMORIA/MANUAL (1.5 puntos, fichero `code.pdf`)

(Ver las instrucciones generales para las prácticas.)

---

3. PUNTOS ADICIONALES (para subir nota):

- Ejercicio complementario en '*programación alfabetizada*' ('literate programming'):

  Se darán puntos adicionales a las prácticas que realicen la documentación de los predicados insertando en el código aserciones y comentarios del lenguaje Ciao Prolog, y entregando como memoria o parte de ella el manual generado automáticamente a partir de dicho código, usando la herramienta `lpdoc` (incluida con Ciao Prolog). En este caso se puede entregar este documento como memoria, y se recomienda por tanto escribir este manual de forma que incluye el contenido que se solicita para la memoria.

- Ejercicio complementario en *codificación de casos de prueba*:

  También se valorará el uso de aserciones `test` que definan casos de prueba para comprobar el funcionamiento de los predicados. Estos casos de test se deben incluir en el mismo fichero con el código.

Hemos dejado en Moodle instrucciones específicas sobre cómo hacer todo esto y un ejemplo de código que tiene ya comentarios y tests, para practicar corriendo `lpdoc` sobre él y ejecutando los tests.