PRÁCTICA 2
CMC

Ángel Igualada Moraga

Índice:

Ejercicio 1 (Sesión 1)

```
AC[Inicial_List, Regla_Integer, t_Integer] := Module[{listaDeListas, l, res, listaRes, cont, ant, pos, act, nuevoEstado, aux},
  listaDeListas = MapThread[Append, {Tuples[{0, 1}, 3], Reverse[IntegerDigits[Regla, 2, 8]]}];
  res = Inicial;
  listaRes = {{Inicial}};
  For[x = 1, x ≤ t, x++,
   l = Join[{Last[res]}, res, {First[res]}];
   nuevoEstado = {};
   For[i = 1, i <= Length[Inicial], i++,
    ant = l[[i]];
    act = l[[i + 1]];
    pos = l[[i + 2]];
    AppendTo[nuevoEstado, FirstCase[listaDeListas, {ant, act, pos, estado_} → estado]];
   ];
   res = nuevoEstado;
   aux = Last[listaRes];
   AppendTo[listaRes, Join[{res}, aux]];
  ];
  listaRes
 ]
```

Ejercicio 1 (Sesión 2)

```
Ejercicio1[AFD_List] := Module[{vecindario, estados},
   vecindario = Join[AFD[[3]], Cases[Tuples[AFD[[1]], 3], {e1_, e2_, e2_}], Cases[Tuples[AFD[[2]], 3], {e1_, e2_, e2_}]];
   estados = Union[AFD[[1]], AFD[[2]]];
   {estados, vecindario, AFD[[5]]}
  ];
```

Ejercicio 2

```
CompruebaCadena[cadena_List, AC_List, frontera_] := Module[{res, i, j, aux},
   res = cadena;
   res[[1]] = FirstCase[AC[[2]], {frontera, cadena[[1]], elem_} → elem];
   For[i = 2, i ≤ Length[res], i++,
    res[[i]] = FirstCase[AC[[2]], {res[[i - 1]], res[[i]], elem_} → elem];
   ];
   Return[MemberQ[AC[[3]], res[[Length[cadena]]]]];
  ];
```

Ejercicio 3

```
Ejercicio3[AFD_List] := Module[{vecindario, estados, res},
  estados = Union[AFD[[1]], AFD[[2]]];
  vecindario = Cases[Tuples[estados, 4], {e1_, e2_, e3_, e4_} /; MemberQ[AFD[[3]], {e1, e2, e4}]];
  vecindario = Join[vecindario, Cases[Tuples[AFD[[2]], 4], {_, e2_, _, e2_}]];
  vecindario = Join[vecindario, Cases[Tuples[AFD[[1]], 4], {_, e2_, _, e2_}]];
  {estados, vecindario, AFD[[5]]}
 ]
```

Ejercicio 4

```
DibujaDiagramaTemporal[cadena_List, AC_List, frontera_] :=
  Module[{res, i, j, aux, diagramaTemporal, colorRules, l},
   colorRules = GetColorRulesForAutomata[AC];
   res = cadena;
   l = {};
   diagramaTemporal = {{cadena}};
   res[[1]] = FirstCase[AC[[2]], {frontera, cadena[[1]], elem_} → elem];
   AppendTo[diagramaTemporal, Append[Last[diagramaTemporal], res]];
   For[i = 2, i ≤ Length[res], i++,
    res[[i]] = FirstCase[AC[[2]], {res[[i - 1]], res[[i]], elem_} → elem];
    AppendTo[diagramaTemporal, Append[Last[diagramaTemporal], res]];
   ];
   For[i = 1, i ≤ Length[diagramaTemporal], i++,
    AppendTo[l,
     ArrayPlot[diagramaTemporal[[i]], PlotLegends → All, Frame → True, FrameTicks → All,
         DataReversed → {True, False},
         Epilog → MapIndexed[Text[Framed[#1, Background → Green], #2 - 1 / 2] &, Transpose@diagramaTemporal[[i]],{2}],
         ColorRules → colorRules, Mesh → All
        ]
      ]
     ];
     Return[ListAnimate[l]];
    ];
```

Función Auxiliar GetColorRulesForAutomata

```
GetColorRulesForAutomata[automata_List] := Module[{numEstados, incremento, l, rgb},
   numEstados = Length[automata[[1]]];
   incremento = 1 / Round[(numEstados / 3)];
   rgb = {0, 0, 0};
   l = {};
   For[i = 1, i ≤ numEstados, i++,
    rgb[[Mod[i, 3] + 1]] += incremento;
    AppendTo[l, automata[[1, i]] → RGBColor[rgb] ];
   ];
   Return[l];
  ];
```

Explicación Código Ejercicio 1 (Sesión 1)

```
AC[Inicial_List, Regla_Integer, t_Integer] := Module[{listaDeListas, l, res, listaRes, cont, ant, pos, act, nuevoEstado, aux},
  listaDeListas = MapThread[Append, {Tuples[{0, 1}, 3], Reverse[IntegerDigits[Regla, 2, 8]]}];
  res = Inicial;
  listaRes = {{Inicial}};
```

En este módulo, buscamos conseguir la representación de un autómata celular tomando como parámetros la configuración inicial, el numero de la regla de wólfram y el número de configuraciones a calcular:

En la línea 2 , convertimos el número de regla en una lista de listas :

- `Tuples[{0, 1}, 3]` Calcula todas las posibles combinaciones de ceros y unos con longitud 2.
- `Reverse[IntegerDigits[Regla, 2, 8]]` Calcula el valor en binario (de longitud 8) del número de regla introducido y le realizamos el reverso para conseguir el orden necesario (Ej: IntegerDigits proporciona 100 y necesitamos 001).

Tras esto, tendríamos por una parte la lita ordenada de las combinaciones de ceros y unos y por otro una lista del valor (0 o 1) de cada regla en orden.

Ej: {{0,0,0},{0,0,1},{0,1,0}…} y {0,1,1…} (Regla 54)

Tras esto, se aplica MapThread que aplica Append a los elementos de la misma posición de la primera y la segunda lista.

listaDeListas sería de la forma :

{{0,0,0,0}, {0,0,1,1}, {0,1,0,1}, {0,1,1,0}, {1,0,0,1}, {1,0,1,1}, {1,1,0,0}, {1,1,1,0}}

En la línea 3 asignamos el valor inicial a la variable res.

En la línea 4 inicializamos listares como una lista de listas de listas, cuya primera posición contendrá la configuración inicial.

El propósito de esta organización, como veremos más adelante, será no tener una lista de configuraciones, sino una lista que muestre la evolución completa, es decir cada elemento contendrá todas las configuraciones anteriores

Explicación Código Ejercicio 1 (Sesión 1)

```
For[x = 1, x ≤ t, x++,
 l = Join[{Last[res]}, res, {First[res]}];
 nuevoEstado = {};
 For[i = 1, i <= Length[Inicial], i++,
  ant = l[[i]];
  act = l[[i + 1]];
  pos = l[[i + 2]];
  AppendTo[nuevoEstado, FirstCase[listaDeListas, {ant, act, pos, estado_} → estado]];
 ];
```

En esta parte, tenemos un primer bucle externo que realizará tantas iteraciones como configuraciones haya que calcular.

En la **línea 2**, a le damos el valor de res añadiéndole su ultimo elemento en el principio y el primero en el final, para simular así el vecindario periódico.

En la **línea 3** fijamos nuevo estado a lista vacía.

En la **línea 4** comienza el bucle for interior que lo que hará será calcular el nuevo estado de cada estado de la configuración inicial.

Para ello, necesitaremos el vecino izquierdo, el estado actual y el vecino derecho.

```
AppendTo[nuevoEstado, FirstCase[listaDeListas, {ant, act, pos, estado_} → estado]];
```

Mediante FirstCase, recuperamos dentro de la lista de lista calculadas la lista que cumple el patrón vecino izquierdo , actual, vecino derecho y devolvemos el 4º elemento que será el nuevo estado, este, se añade en última posición a nuevo estado, preservando el orden, al finalizar todas las iteraciones nuevoEstado contendrá la nueva configuración.

Explicación Código Ejercicio 1 (Sesión 1)

```
    res = nuevoEstado;
    aux = Last[listaRes];
    AppendTo[listaRes, Join[{res}, aux]];
   ];
   listaRes
  ]
```

En la línea 1 , asignamos la nueva configuración a res.

En la línea 2, asignamos el valor del último elemento de listares a aux (este será una lista de listas con todas las configuraciones anteriores).

En la línea 3, añadimos a listares un nuevo elemento que será una lista que contendrá las configuraciones anteriores y la nueva.

En la línea 5 , devolvemos listaRes que podría ser de la forma :

{

{ {0,1,1,1,1,0,1} },

{ {1,0,0,0,0,1,1} , {0,1,1,1,1,0,1} },

{ {0,1,0,0,1,0,0} ,{1,0,0,0,0,1,1} , {0,1,1,1,1,0,1} },

}

Así,con `ListAnimate[Map[ArrayPlot, l]];`

Podríamos obtener una lista animada que mostrase la evolución del autómata (en el código hay varios ejemplos).

Explicación Código Ejercicio 1 (Sesión 2)

```
Ejercicio1[AFD_List] := Module[{vecindario, estados},
   vecindario = Join[AFD[[3]], Cases[Tuples[AFD[[1]], 3], {e1_, e2_, e2_}], Cases[Tuples[AFD[[2]], 3], {e1_, e2_, e2_}]];
   estados = Union[AFD[[1]], AFD[[2]]];
   {estados, vecindario, AFD[[5]]}
  ];
```

En la **línea 2**, unimos:

- `AFD[[3]]` Las transiciones del AFD proporcionado
- `Cases[Tuples[AFD[[1]], 3], {e1_, e2_, e2_}]` De todas las combinaciones posibles de estados del AFD de longitud 3, aquellas en las que el segundo y el tercer elemento coinciden.
- `Cases[Tuples[AFD[[2]], 3], {e1_, e2_, e2_}]` De todas las combinaciones posibles de símbolos del alfabeto del AFD con longitud 3, aquellas en las que el segundo y el tercer elemento coinciden.

En la **línea 4** devolvemos una lista con la unión de los estados y del alfabeto del AFD, el vecindario calculado y los estados finales del AFD.

Explicación Código Ejercicio 2 (Sesión 2)

```
CompruebaCadena[cadena_List, AC_List, frontera_] := Module[{res, i, j, aux},
   res = cadena;
   res[[1]] = FirstCase[AC[[2]], {frontera, cadena[[1]], elem_} → elem];
```

En la **línea 1**, asignamos el valor de la cadena de entrada a res.

En la **línea 2** , a res[[1]] le asignamos el estado de la regla que sigue el patrón { est frontera, actual , algo}

```
    For[i = 2, i ≤ Length[res], i++,
     res[[i]] = FirstCase[AC[[2]], {res[[i - 1]], res[[i]], elem_} → elem];
    ];
    Return[MemberQ[AC[[3]], res[[Length[cadena]]]]];
   ];
```

Mediante el bucle, comenzamos analizando el segundo elemento, hacemos exactamente lo mismo que en la instrucción anterior, recuperamos el nuevo estado, y lo asignamos a su posición.

Por último, devolvemos el resultado de si el último estado es miembro de los finales.

Explicación Código Ejercicio 3 (Sesión 2)

```
vecindario = Cases[Tuples[estados, 4], {e1_, e2_, e3_, e4_} /; MemberQ[AFD[[3]], {e1, e2, e4}]];
```

Esta línea, devuelve todas las combinaciones posibles de longitud 4 de los estados tales que sus dos primeros elementos y su cuarto establecían una transición en el AFD.

```
vecindario = Join[vecindario, Cases[Tuples[AFD[[2]], 4], {_, e2_, _, e2_}]];
vecindario = Join[vecindario, Cases[Tuples[AFD[[1]], 4], {_, e2_, _, e2_}]];
```

Estas dos, añaden las combinaciones de lo que antes eran estados tal que el segundo y el cuarto es el mismo y las combinaciones de los que antes eran símbolos de la misma manera.

Explicación Código Ejercicio 4 (Sesión 2)

```
DibujaDiagramaTemporal[cadena_List, AC_List, frontera_] :=
  Module[{res, i, j, aux, diagramaTemporal, colorRules, l},
   colorRules = GetColorRulesForAutomata[AC];
   res = cadena;
   l = {};
   diagramaTemporal = {{cadena}};
   res[[1]] = FirstCase[AC[[2]], {frontera, cadena[[1]], elem_} → elem];
   AppendTo[diagramaTemporal, Append[Last[diagramaTemporal], res]];
```

En la **línea 3**, se llama a una función auxiliar que se explicará con más detalle adelante, lo que hace básicamente, es devolver una lista de colorRules para cada estado de un Autómata celular dado intentando que los colores de cada estado sean bien diferenciables, esto, lo hacemos porque las Color Funtions proporcionadas por Mathematica dan problemas al usar variables y no números.

En la **línea 6**, inicializamos diagramaTemporal , que será una lista de listas de configuraciones, en la cuál la primera , será una lista con una única configuración , la inicial y la segunda contendrá la inicial y la recién calculada y así sucesivamente.

En la **línea 7**, cambiamos el estado del primer elemento.

En la **línea 8** añadimos la que será la segunda lista de configuraciones con la inicial y la nueva con el primer elemento en el nuevo estado.

```
For[i = 2, i ≤ Length[res], i++,
 res[[i]] = FirstCase[AC[[2]], {res[[i - 1]], res[[i]], elem_} → elem];
 AppendTo[diagramaTemporal, Append[Last[diagramaTemporal], res]];
];
```

En este bucle vamos calculando Estados y vamos añadiendo a diagramaTemporal todas las configuraciones anteriores y la recién calculada.

```
For[i = 1, i ≤ Length[diagramaTemporal], i++,
 AppendTo[l,
  ArrayPlot[diagramaTemporal[[i]], PlotLegends → All, Frame → True, FrameTicks → All,
      DataReversed → {True, False},
      Epilog → MapIndexed[Text[Framed[#1, Background → Green], #2 - 1 / 2] &, Transpose@diagramaTemporal[[i]],{2}],
      ColorRules → colorRules, Mesh → All
     ]
   ]
  ];
  Return[ListAnimate[l]];
 ];
```

Tras haber calculado diagramaTemporal, lo recorremos elemento a elemento y a l le añadimos el resultado de hacer el ArrayPlot de esa lista de configuraciones.

En este ArrayPlot hemos establecido algunas opciones personalizadas:

- `PlotLegends → All, Frame → True, FrameTicks → All` Mostramos la leyenda, y las marcas laterales.
- `DataReversed → {True, False}` Invertimos el orden de las filas (continuamos con el orden de clase el inferior es el instante 0)
- `Epilog → MapIndexed[Text[Framed[#1, Background → Green], #2 - 1 / 2] &, Transpose@diagramaTemporal[[i]],{2}]`

  Esta persigue únicamente mostrar un label en cada cuadricula con el estado que identifica.
- `ColorRules → colorRules, Mesh → All` Establecemos las reglas de color calculadas por la función y mostramos la cuadrícula.

Explicación Código Función GetColorRules

```
GetColorRulesForAutomata[automata_List] := Module[{numEstados, incremento, l, rgb},
  numEstados = Length[automata[[1]]];
  incremento = 1 / Round[(numEstados / 3)];
  rgb = {0, 0, 0};
  l = {};
  For[i = 1, i ≤ numEstados, i++,
   rgb[[Mod[i, 3] + 1]] += incremento;
   AppendTo[l, automata[[1, i]] → RGBColor[rgb] ];
  ];
  Return[l];
 ];
```

Esta función, en la línea 2 calcula el número de estados (necesario para saber que incremento en la paleta de colores dar a cada uno),

Tras esto, calcula el incremento, Mathematica acepta rangos de 0 a 1 para RGBColor por lo que dividimos 1 entre el numero de estados dividido por 3 (ya que el color tiene 3 componentes que iremos aumentar para intentar conseguir colores más distintos).

En la línea 4 creamos una lista con tres ceros que representa los valores de rojo verde y azul.

En la línea 5 inicializamos l que será la lista de ColorRules.

Tras esto, el bucle, recorre los estados y suma a la componente correspondiente de rgb (i % 3) el incremento calculado, tras esto creamos la ColorRule que sigue la forma signo -> Color y la añadimos a l (RGBColor devuelve el color para la lista rgb que hemos creado).

Por último devolvemos la lista con las ColorRules (tantas como estados).

Ejemplos