

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“SIMULACIÓN DE AUTÓMATAS CELULARES 1D Y SUS PARTÍCULAS EMERGENTES”

TOMÁS SANTIAGO NIETO GUERRERO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Andrés Moreira
Profesor Correferente: Roberto Asín

Diciembre - 2024

AGRADECIMIENTOS

Agradezco a mi profesor guía Andrés Moreira por ayudarme en diferentes desafíos a lo largo de todo lo que significó el desarrollo de la presente memoria.

Agradezco a mi familia y amigos por acompañarme durante tantos años y permitirme crecer como persona en tantas facetas diferentes.

RESUMEN

Resumen— Los autómatas celulares (CA) unidimensionales pueden exhibir estructuras emergentes llamados defectos. Las partículas (o "defectos") son la frontera entre dos dominios, los cuales son fondos regulares con periodicidad en espacio y tiempo. El trabajo de esta memoria propone algoritmos para encontrar y describir tanto los defectos como la manera en que interactúan entre sí, para cualquier CA unidimensional. Luego, se añade una capa de abstracción, donde se permite crear simulaciones de partículas a partir de los defectos encontrados. Todo esto se encapsula en una interfaz gráfica de usuario que le permite al investigador estudiar los CA con herramientas que no estaban a su alcance antes de este desarrollo. Los resultados principales muestran defectos e interacciones para la regla elemental 54, y variadas simulaciones donde la herramienta predice correctamente los resultados.

Palabras Clave— Autómatas Celulares; Defectos; Simulación de Partículas; Algoritmos; Sistemas Dinámicos

ABSTRACT

Abstract— One-dimensional cellular automata (CA) can exhibit emergent structures known as defects. Particles (or "defects") are the frontier of two domains, which are regular backgrounds with space and time periodicity. The present work proposes algorithms to find and describe the defects of any CA rule and how they interact. Then, a layer of abstraction is further introduced, which allows the creation of particle simulations, based on the defects found. All this is encapsulated in a graphical user interface, that allows the researcher to study CA with tools that were not available to them previous to this work. Main results show defects and interactions for elementary rule 54, as well as numerous simulations that showcase the tool's correct predictions.

Keywords— Cellular automata; Defects; Particle Simulation; Algorithms; Dynamical Systems

ÍNDICE DE CONTENIDOS

RESUMEN	III
ABSTRACT	III
ÍNDICE DE FIGURAS	v
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Contexto General del Problema	3
1.2 Objetivos de la Presente Memoria	6
CAPÍTULO 2: MARCO CONCEPTUAL	7
2.1 Autómatas Celulares Unidimensionales	7
2.2 Reglas Elementales	8
2.3 Clases de Wolfram	9
2.4 Reversibilidad y jardines del edén	10
2.5 Otras reglas	11
2.6 Partículas emergentes	12
2.6.1 Dominios	14
2.6.2 Partículas	14
2.6.3 Interacciones	16
2.7 Herramientas	18
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	19
3.1 Encontrando Dominios	19
3.2 Encontrando Partículas	21
3.3 Encontrando Interacciones	24
3.4 Filtrando Dominios	25
3.5 Dinámica de partículas	27
3.6 La herramienta	29
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	30
4.1 Simulando autómatas celulares	30
4.2 Encontrando las estructuras	32
4.3 Abstrayendo hacia la simulación de partículas	33
4.4 Resultados en algunas reglas elementales	34
CAPÍTULO 5: CONCLUSIONES	36
REFERENCIAS BIBLIOGRÁFICAS	43

ÍNDICE DE FIGURAS

1	Soluciones para el firing squad synchronization problem	4
2	Árbol del problema.	5
3	Ejemplo de autómata celular 1D de dos estados	7
4	Ejemplos de función de transición	8
5	Evolución de autómata celular	8
6	Ejemplos de autómatas celulares 1D	9
7	Clases de Wolfram	10
8	Autómatas reversibles	11
9	Autómata celular no inyectivo	12
10	Tabla de dominios e interacción entre partículas emergentes	13
11	Ejemplo de Dominio	14
12	Ejemplo de Partículas	15
13	Ejemplo de Velocidad	16
14	Ejemplo de Interacciones	17
15	Ejemplo de Interacción interrumpida	18
16	Ejemplo de autómata celular con los dominios filtrados	26
17	Ejemplo Visual de Dinámica de Partículas	29
18	Primera Vista de la Herramienta	31
19	Instancia de simulación y validación	32
20	Segunda Vista de la Herramienta	33
21	Instancia de detección de estructuras y validación	34
22	Segunda Vista de la Herramienta - Simulación de Partículas	35
23	Segunda Vista de la Herramienta - Simulación de Partículas	36
24	Ejemplo de discrepancia por fase	37
25	Resultados de la regla 60	38
26	Resultados de la regla 122	39
27	Colisión de partículas de la regla 54: A	40
28	Colisión de partículas de la regla 54: B	41
29	Colisión de partículas de la regla 54: C	42
30	Colisión de partículas de la regla 54: D	43
31	Colisión de partículas de la regla 54: E	44
32	Flujo de creación de configuración de partículas para regla 54	45
33	Dominios filtrados para la regla 54	45

INTRODUCCIÓN

Los autómatas celulares son un tipo de redes dinámicas, donde la topología es constante, y a cada paso en el tiempo los estados de los nodos cambian de acuerdo a lo dictado por una función de transición local ϕ . Esta función es la misma para todos los nodos de la red, por lo que se dice que la función es homogénea. $\phi(x)$ retorna un valor que depende de los vecinos de x . Adicionalmente, la estructura de la red también es homogénea, lo que significa que todos los nodos tienen el mismo grado. En el caso de autómatas celulares de una dimensión, los nodos son dispuestos en una línea y los vecinos de cada nodo son los $2 \cdot r$ nodos más cercanos, donde $r \in \mathbb{N}$ corresponde al radio de vecindad. En $t = 0$ cada nodo es inicializado con algún valor $0 \leq n < k$ donde k es la cantidad de estados en que los nodos pueden estar. En cada unidad de tiempo todos los nodos tienen solo y solo un estado de los posibles, y este estado se actualiza usando la función ϕ mencionada hace un momento.

Esta memoria se centra en el análisis de los resultados que se encuentran haciendo uso de las propiedades emergentes de los autómatas celulares unidimensionales. Para cualquier sistema dinámico, las propiedades emergentes son comportamientos que se crean a partir de la interacción de los elementos unitarios de dicho sistema a partir de las reglas que dictan el comportamiento del elemento unitario. En el caso de los autómatas celulares, la propiedad emergente que se verá es la responsable de crear estructuras conocidas como “defectos” o “partículas”. Estas estructuras pueden llegar a lograr una comunicación global entre los elementos que forman el autómata celular.

Una de las principales dificultades relacionadas a estos defectos es detectarlas. En la literatura no existen algoritmos que las describan, dado un autómata celular, por lo que se recurre a una inspección por parte del investigador. Esto puede resultar en una discrepancia entre lo que se logra ver a simple vista y lo que realmente está presente. Esto puede resultar en una dificultad a la hora de estudiar los autómatas celulares, puesto que, adicionalmente, no existen herramientas abiertas al público que permitan un extenso análisis de estos sistemas.

Para resolver esta problemática, en esta memoria se expone el desarrollo de una herramienta con la que el usuario puede interactuar mediante una interfaz diseñada en torno al estudio de los autómatas celulares y sus partículas emergentes. Adicionalmente, en los capítulos siguientes se explica el proceso para obtener los algoritmos que describen estas partículas y que hacen uso de algunas suposiciones.

Esto se inicia con el algoritmo responsable de la detección de los “dominios”, patrones que se repiten en el tiempo y en el espacio, las cuales son las estructuras más básicas. A partir de esto se crea el algoritmo que describe las partículas, que no son más que fronteras entre dos dominios. El último algoritmo describe las intersecciones posibles entre dos partículas, que incluye su colisión y resultado. Una vez que están todos los algoritmos creados, se desarrolla la herramienta que hace uso de estos. Todo el código es en lenguaje *python*, el cual dispone de librerías muy útiles a la hora de presentar visualmente los autómatas celulares, y para

crear interfaces de usuario.

La estructura del resto de la memoria inicia con la definición del problema, donde se da un contexto general del problema tratado, pero con una mayor profundidad a la vista en la introducción. La definición del problema aborda además los objetivos de la presente memoria. El capítulo dos expone el marco conceptual que se necesita para entender todos los elementos que están involucrados en el área de investigación de esta memoria. Los contenidos vistos en ese capítulo definen el autómata celular unidimensional, las funciones de transición global y local, los dominios, partículas, y se revisan las reglas elementales. Finalmente, en este capítulo se mencionan las herramientas tecnológicas usadas en la creación de la memoria.

El capítulo tres presenta la propuesta de solución. Aquí se muestran los algoritmos creados y se revisan calculando los costos computacionales de cada uno. Primero se describe la manera de detectar los dominios, los cuales tienen una periodicidad temporal y espacial máxima. Luego se describe la manera de detectar las partículas, que presentan un periodo al igual que los dominios. A continuación se presenta el algoritmo que describe las interacciones, las cuales suceden cuando dos partículas colisionan, produciendo variados resultados. El resto del tercer capítulo se enfoca en el filtro de dominios, lo cual aplica un filtro visual a cierta configuración para un mejor análisis por parte del usuario, y se presenta además la forma de simular configuraciones de partículas como objetos físicos, separados pero relacionados con el contexto de autómata celular.

El capítulo cuatro hace uso de los algoritmos creados y la herramienta desarrollada para demostrar el cumplimiento de los objetivos específicos de esta memoria. Para esto se comparan los resultados de los algoritmos con resultados conocidos. Además, en este capítulo se muestra el correcto desempeño de todas las funcionalidades de la herramienta que forma la propuesta de solución. Para esto, el capítulo se estructura primero presentando la correcta simulación de autómatas celulares, siguiendo la detección de las estructuras emergentes, finalizando con todo lo relacionado a la dinámica de partículas y su conexión con los autómatas celulares.

El último capítulo contiene la conclusión, donde se validan los resultados en referencia a los objetivos planteados, se explicitan los alcances de la solución propuesta, y a partir de ahí se motivan varias líneas de trabajo para profundizar o ampliar lo visto en esta memoria.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto General del Problema

Los autómatas celulares son estructuras que primero fueron estudiadas hace ya más de 60 años por von Neumann, en un intento de formular de manera algorítmica el proceso en el cual una máquina puede generar auto-reproducción [von Neumann, 1966]. Desde entonces han sido objeto de investigación y estudio en múltiples áreas como la informática, matemática, biología, economía, física y sociología por mencionar algunos [Mazoyer, 1987] [Deutsch y Dormann, 2005] [Axelrod y Hamilton, 1981] [Packard, 1985] [Dewdney, 1984].

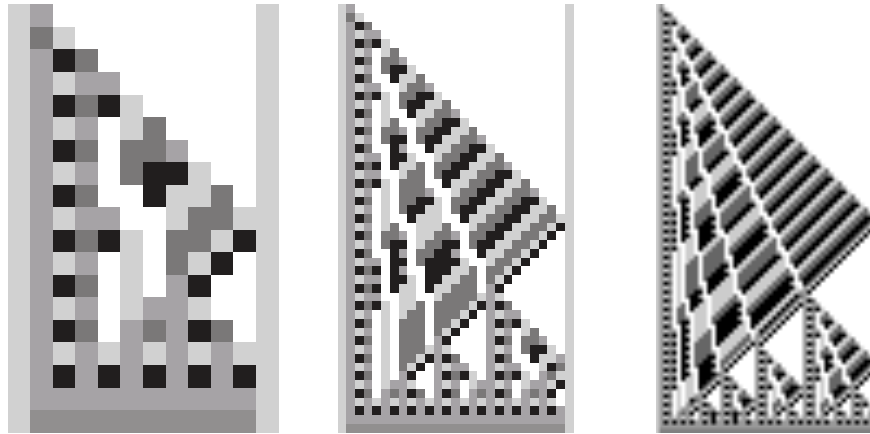
Un autómata celular es un arreglo de celdas idénticas, comúnmente de 1 o 2 dimensiones. Para definir un autómata celular necesitamos un conjunto de estados que pueden tomar estas celdas, la topología de la malla, un radio de vecindad, y una función local de transición, la cual se aplica en cada celda de la configuración actual. En el capítulo 2 entraremos en mayor detalle en las definiciones y se verán múltiples ejemplos y variantes.

Hoy en día importantes figuras siguen aportando a la investigación de autómatas celulares, como por ejemplo Stephen Wolfram, uno de los mayores referentes de esta área [Wolfram, 2002]. En 1984 Wolfram estudia los 256 posibles autómatas celulares que se pueden crear con dos estados y radio de vecindad igual a uno, y crea lo que se van a conocer como las “clases de Wolfram” [Wolfram, 1984]. Al respecto, existen 4 clases de Wolfram, las cuales se detallan a continuación:

- clase I: El autómata es atraído rápidamente a una configuración fija.
- clase II: El autómata evoluciona a estructuras periódicas o continuamente repetidas.
- clase III: El autómata exhibe un comportamiento aleatorio, altamente sensible a la configuración inicial de las celdas.
- clase IV: El autómata presenta comportamiento complejo. Es útil visualizarla como la clase que está entre la II y la III. Un autómata de esta clase puede generar estructuras emergentes que interactúan entre sí, pero para reconocerlas no existen criterios formales, mas bien se pueden apreciar a medida que el autómata va evolucionando, viendo no solo la configuración del tiempo presente, sino también las pasadas.

Dentro de la clase IV se pueden generar lo que se conocen como “partículas emergentes”, que son patrones periódicos locales que se desplazan sobre algún fondo regular. Estas partículas emergentes pueden colisionar e interactuar entre sí, por lo que a veces es útil visualizar estas estructuras y su comportamiento como un análisis de partículas interactuando. Una

de las aplicaciones más conocidas de este comportamiento es el “firing squad synchronization problem” (o “problema de sincronización”). Este problema pide encontrar la función de transición que permite, dada una configuración inicial definida que siempre es la misma, que todas las n celdas lleguen por primera vez a cierto estado, en el mismo tiempo t . Para esto debemos hacer uso de reglas locales para alcanzar un estado global particular.



adapted from Wolfram, S. *A New Kind of Science*.
Wolfram Media, p. 1035, 2002.

Figura 1: Solución de mínimos estados conocida vista en 3 distintos largos. El tiempo avanza de arriba a abajo.

Fuente: Weisstein, Eric W. "Firing Squad Problem." de MathWorld—A Wolfram Web Resource. <https://mathworld.wolfram.com/FiringSquadProblem.html>.

Actualmente no existen herramientas capaces de simular partículas emergentes y que puedan abstraer el autómata a una dinámica de partículas. Esto quiere decir que no existen herramientas que nos permitan visualizar el autómata celular como una simulación de partículas en movimiento, o que nos permitan generar nuevos autómatas celulares a partir de una configuración inicial de estas partículas. Esto tiene como consecuencia que resulta difícil estudiar estos comportamientos emergentes en los autómatas celulares. Además debemos considerar que no existen herramientas de simulación con múltiples funcionalidades para configurar los componentes que definen el autómata celular, lo cual dificulta aún más su investigación.

Para lograr simular y poder generar un análisis de estas partículas emergentes, necesitamos de una herramienta que pueda identificarlas. Una herramienta como esta, además de ser capaz de simular una gran variedad de autómatas celulares 1D, podría servir para encontrar cotas inferiores de complejidad comunicacional [Goles *et al.*, 2011], o ayudar en la construcción de circuitos lógicos para demostrar la universalidad computacional de ciertos autómatas. Lo anterior es debido a que podríamos usar los patrones emergentes y tratarlos como señales booleanas, cuyas interacciones generarían los circuitos deseados. Esto último ayudaría en las demostraciones de que ciertas reglas elementales son computacionalmente universales, como lo es la regla 110 [Cook, 2009].

A continuación se presenta una figura que muestra el árbol del problema. En él se puede apreciar el problema principal, junto con las causas de este y los efectos que pueden originarse. Adjunto además va el objetivo general y los objetivos específicos de la presente memoria.

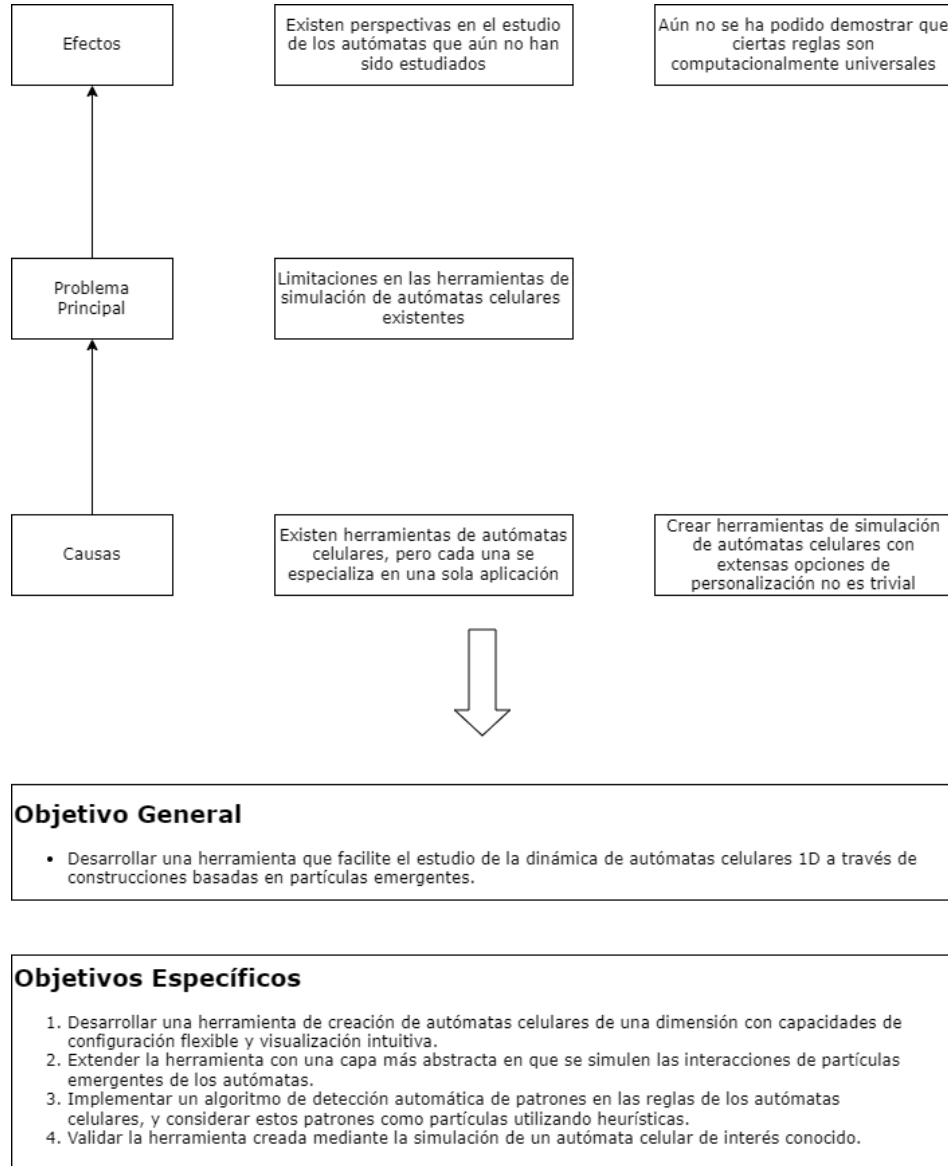


Figura 2: Árbol del problema.

Fuente: Elaboración Propia.

1.2. Objetivos de la Presente Memoria

■ Objetivos Generales

- Desarrollar una herramienta que facilite el estudio de la dinámica de autómatas celulares 1D a través de construcciones basadas en partículas emergentes.

■ Objetivos Específicos

1. Desarrollar una herramienta de creación de autómatas celulares de una dimensión con capacidades de configuración flexible y visualización intuitiva.
2. Extender la herramienta con una capa más abstracta en que se simulen las interacciones de partículas emergentes de los autómatas.
3. Implementar un algoritmo de detección automática de patrones en las reglas de los autómatas celulares, y considerar estos patrones como partículas utilizando heurísticas.
4. Validar la herramienta creada mediante la simulación de un autómata celular de interés conocido.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Autómatas Celulares Unidimensionales

Consideraremos los autómatas celulares unidimensionales como un arreglo o fila de celdas cuadradas. Cada celda existe con un valor asociado llamado el “estado” de la celda. El número de estados posibles se conoce como el alfabeto, o también como el conjunto de estados. Para un autómata pueden haber k estados, donde k es un número natural mayor o igual que 2. Cada celda del autómata puede tomar un y solo un estado en un determinado tiempo. El largo del autómata, es decir, la cantidad de celdas que contiene, es finito pero puede ser tan largo como lo pida el análisis. El autómata más simple es el de dos estados (blanco y negro, 0 y 1), que se puede ver en la siguiente figura.



Figura 3: Autómata celular de 9 celdas y 2 estados, en tiempo $t = 0$.

Fuente: Elaboración Propia.

En los autómatas celulares, el tiempo transcurre de forma discreta. En cada paso del tiempo, cada celda actualiza el valor de su estado. El valor que tomará el estado de cada celda en tiempo $t + 1$ dependerá de los estados de sus celdas vecinas en un radio igual a r en el tiempo t . Si $c_i(t)$ describe el estado de la celda i en tiempo t , y el radio de vecindad es $r = 1$ tenemos que

$$c_i(t + 1) = \phi[c_{i-1}(t), c_i(t), c_{i+1}(t)] \quad (1)$$

donde ϕ es la función de transición local. La función global de transición $\Phi : A^n \rightarrow A^n$ aplica ϕ para cada vecindad del autómata dado, donde A es un alfabeto y n el largo del autómata. En el ejemplo de la figura 3, para radio $r = 1$ vemos que las celdas de los bordes solo tienen un vecino. Es por esto que consideraremos los autómatas celulares como estructuras con condiciones de frontera periódica, donde las celdas de los bordes serán vecinas. Por ejemplo, para $r = 1, k = 2$ podemos considerar la siguiente función de transición:

$$\phi[c_{i-1}(t), c_i(t), c_{i+1}(t)] = c_{i-1}(t) \vee c_{i+1}(t) \quad (2)$$

Es decir, la celda i tomará el valor de 1 en tiempo $t + 1$ si al menos uno de sus vecinos estaba en ese estado en tiempo t . La cantidad de posibles configuraciones que puede tomar nuestra vecindad de $r = 1$ y $k = 2$ es de $k^{2r+1} = 8$. Visualmente podemos ver los valores posibles a partir de las ocho posibles configuraciones en la siguiente figura.

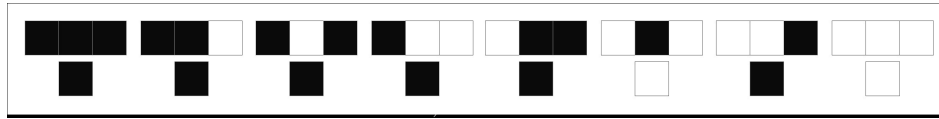


Figura 4: Función de transición descrita en la ecuación (2).

Fuente: Elaboración Propia.

Como podemos ver, cada configuración de 3 celdas está representada, donde la celda que se actualiza es la celda que está en medio. El resultado de la función de transición es la celda singular que está debajo, que representa el valor de la celda en el siguiente tiempo. A continuación se observa la configuración inicial de la figura 3 evolucionando con esta función de transición.

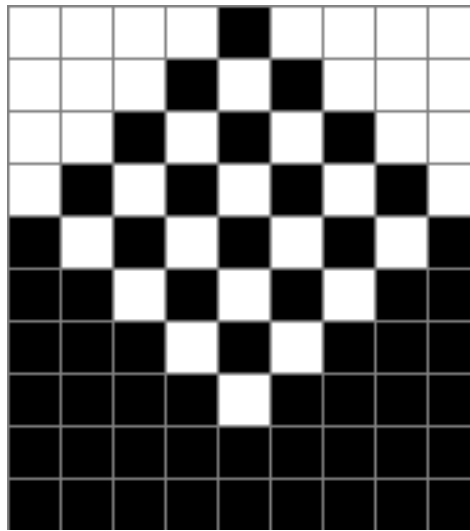


Figura 5: Primeras 10 iteraciones del autómata celular bajo la función de transición de (2), con el tiempo avanzando hacia abajo.

Fuente: Elaboración Propia.

Es fácil ver que, para un radio r y número de estados k , la cantidad de posibles funciones de transición es $k^{k^{2r+1}}$. Por ejemplo, para $r = 1$ y $k = 2$, la cantidad de reglas posibles es $2^{2^{2 \cdot 1 + 1}} = 2^8 = 256$. Estas reglas son conocidas como las reglas elementales. Intuitivamente, podemos ver en la figura 4 que todas las combinaciones posibles son resultado de elegir, para cada uno de los 8 posibles inputs, si queremos 0 o 1 como output. Entonces, podemos tratar la secuencia de outputs como un número binario de 8 bits, dando como resultado las 256 maneras posibles en que esta secuencia puede existir.

2.2. Reglas Elementales

Como habíamos dicho, existen 256 reglas elementales, que corresponden a todas las funciones de transición que podemos tener para un radio de uno y dos estados. Cada una de

estas reglas va a ser identificada por un número, entre 0 y 255 (en total, 256) que se obtiene a partir de los outputs representados como un número binario de 8 bits. Por ejemplo, la regla correspondiente a la figura 4 es la regla 250, ya que sus outputs corresponden al binario $11111010_2 = 250$. En la siguiente figura se pueden ver cuatro ejemplos de las reglas elementales. Las configuraciones iniciales de las cuatro son la misma, 50 celdas todas blancas excepto la del medio que es negra.

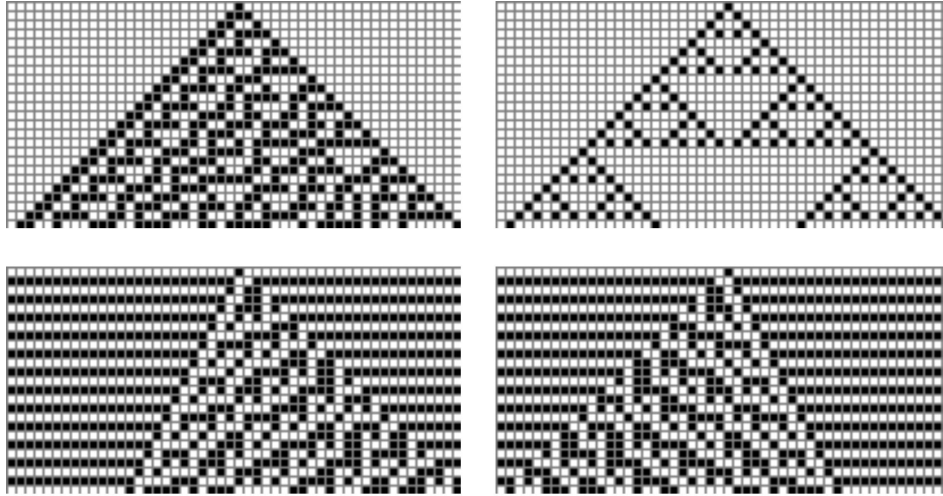


Figura 6: Ejemplos de autómatas celulares 1D. En orden, las reglas elementales 30, 90, 75, y 89. Se puede notar a simple vista que la regla 89 es la versión reflejada de la regla 75.

Fuente: Elaboración Propia.

Como se puede observar en la figura 6 las reglas 75 y 89 son la versión reflejada de la otra. En general, las reglas elementales van a tener otras que se parecen en comportamiento, ya sea por ser la versión reflejada, la versión “negativa”, o ambas variantes al mismo tiempo.

2.3. Clases de Wolfram

En 1984 Stephen Wolfram define las clases de complejidad, también conocidas también como las “clases de Wolfram” [Wolfram, 1984]. Con ellas podemos clasificar las reglas elementales que presentan comportamientos similares. Como se vio en el capítulo anterior, existen cuatro clases. Por ejemplo, en la figura 6 podemos ver la clase III, caótica (regla 30), y la clase 2, Ordenada y periódica (regla 90).

Como observación, podemos mencionar que si consideramos cualquier autómatata celular finito, tarde o temprano exhibirá un comportamiento periódico, técnicamente volviéndolo un autómatata celular de clase II. Esto es debido a que, dado que el autómatata se compone de una cantidad finita de celdas, existe una cantidad finitas de configuraciones en las que el autó-

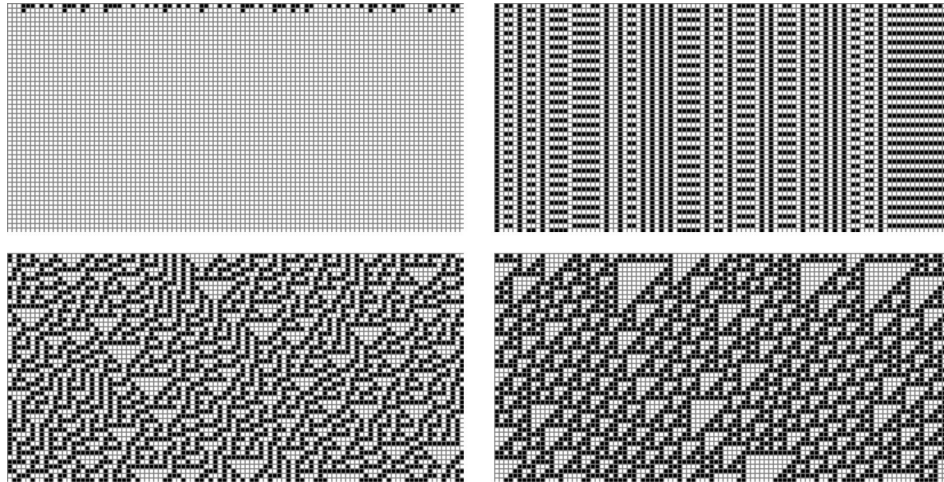


Figura 7: Ejemplos de autómatas de cada clase, en orden. La configuración inicial es aleatoria.

Fuente: Elaboración Propia.

mata puede estar. Para un autómata celular de k estados y n celdas, la cantidad de posibles configuraciones es k^n . Entonces, cualquier autómata celular que se ejecute por suficiente tiempo llegará a una configuración en la que ya estuvo, cayendo en un comportamiento periódico.

Dentro de la clase IV están los autómatas celulares que son candidatos a ser computacionalmente universales. Un autómata celular computacionalmente universal es capaz de simular una máquina de Turing universal o modelo de cómputo equivalente. Como habíamos mencionado en el capítulo anterior, Cook demostró que la regla elemental 110 es computacionalmente universal [Cook, 2004]. Existen otras reglas de la clase IV que se cree que también tienen la propiedad de la regla 110.

2.4. Reversibilidad y jardines del edén

Existen autómatas celulares que tienen la propiedad de que sean reversibles en el tiempo. Esto nos permite tener una configuración cualquiera, y saber no solo sus configuraciones en el futuro, sino también todas las que vinieron antes. Para que un autómata sea reversible, su función de transición debe ser inyectiva, es decir, cada imagen tiene asociada a lo más una preimagen y cada elemento del dominio de la función tiene una imagen.

Si consideramos el conjunto de configuraciones posibles como el dominio y recorrido, entonces si un autómata es inyectivo, también será epiyectivo, lo que a su vez lo hace biyectivo. Si un autómata no cumple con que sea biyectivo, entonces, para una configuración, no podremos saber cual fue el camino pasado, debido a que puede haber más de una configuración

que tenga como imagen el mismo elemento.

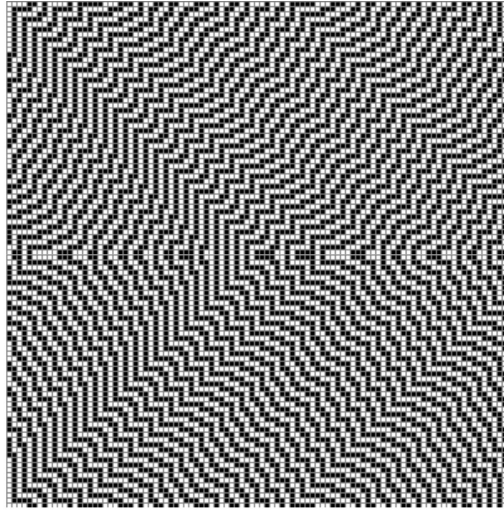


Figura 8: Ejemplo de autómatas celulares reversibles. La primera mitad corresponde a la regla 15, mientras que la segunda mitad corresponde a la regla 85.
Fuente: Elaboración Propia.

Si un autómata celular no es biyectivo, entonces dos o más configuraciones tendrán la misma imagen, y dado que el conjunto de ida y llegada es el mismo, tienen la misma cardinalidad (misma cantidad de elementos) lo que resulta en que haya configuraciones en el codominio a las que ninguna configuración puede llegar, es decir, que no tienen preimagen. Tales configuraciones se les conocen como “jardines del edén”.

2.5. Otras reglas

En esta sección se verá lo que son reglas de autómatas que van más allá del radio unitario para definir los vecinos. Particularmente, se mostrará la forma en que se expresa el código que define la regla, puesto que más adelante veremos estos códigos.

Los códigos pueden estar en hexadecimal, lo cual ayuda a que no sean tan largos. Esto es especialmente útil cuando el radio es mayor a uno, puesto que la cantidad de transiciones que se tienen que expresar aumenta como $2^{(2*r+1)}$ donde r es el radio y asumimos en este caso que la cantidad de estados es 2.

Por ejemplo, para un radio de 3 la cantidad de transiciones es de 128. Los códigos, entonces, serán un *string* de símbolos que representan números en hexadecimal. Un ejemplo es la regla bajo estudio en el paper de Mitchell [Mitchell *et al.*, 1996].

0504058705000f77037755837bfbb77f

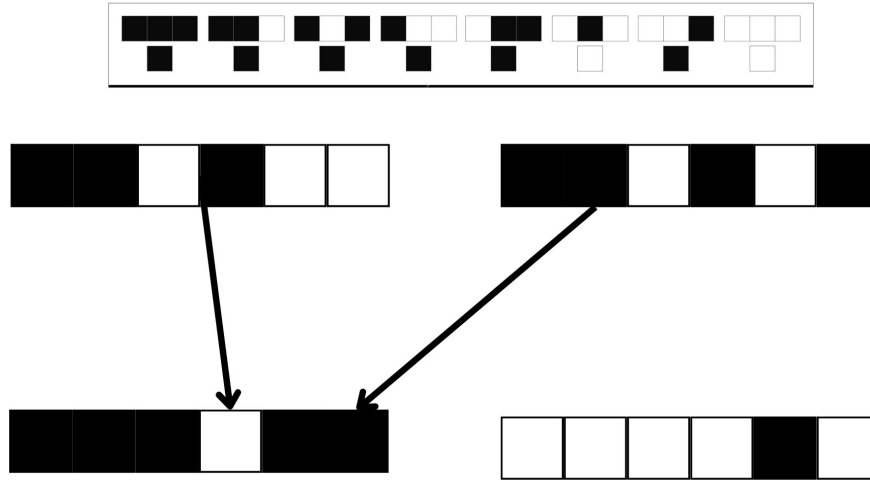


Figura 9: La regla elemental 250 no es biyectiva. Las dos configuraciones presentes producen la misma imagen. La configuración mostrada abajo a la derecha corresponde a un jardín del edén.

Fuente: Elaboración Propia.

Para decodificar la secuencia se convierte en binario, usando exactamente 4 bits para representar cada símbolo hexadecimal. El resultado en este caso es:

```
0000010100000100000001011000011100000101000000000000111101110111
0000001101110111010101011000001101111011111111111101101110111111
```

Este *string* resultante es de largo 128, lo cual coincide con la cantidad de transiciones posibles para una regla de autómatas celulares de radio 3. Esto es debido a que cada *bit* corresponde a un *output*, para cada uno de los 128 posibles. El primer *bit*, 0 es el resultado de aplicar la función de transición local al vecindario 0000000, y así para el resto. El *bit* número 42, cuyo valor es 0, será el resultado de aplicar la función de transición local al vecindario 010101 (42 en binario).

2.6. Partículas emergentes

En 1993, Hanson y Crutchfield definen un modelo computacional basado en interacciones de partículas dentro de los autómatas celulares [Hanson y Crutchfield, 1997]. Se les llaman “defectos” o partículas a las estructuras que se generan como una propiedad emergente. Para un autómata cualquiera, se pueden definir los dominios como estructuras emergentes más básicas, una región homogénea donde un mismo patrón se repite [Das, 1998]. Podemos

filtrar los dominios en el autómata celular, esencialmente ignorándolos para destacar los defectos en los dominios, también conocidos como partículas.

Múltiples dominios y defectos pueden generarse en un mismo autómata celular. Asociada a cada partícula viene su velocidad, que representa cuántas celdas se desplaza (+ para movimientos a la derecha y - para movimientos a la izquierda) por iteración. Por ejemplo, un defecto α con velocidad $+\frac{2}{3}$ se moverá 2 celdas hacia la derecha por cada 3 tiempos. Las velocidades fraccionarias se deben a que los defectos a menudo tienen un comportamiento periódico en el tiempo, no solo en el espacio. En el ejemplo anterior, α puede tener un desplazamiento espacial de 2 celdas y una periodicidad temporal de 3 tiempos.

Como es de suponer, estas partículas pueden interactuar entre sí. Dos partículas pueden ser borradas en contacto, o pueden reaccionar para formar otras partículas. Por último, existen interacciones donde las partículas se cruzan espacialmente, pero no interactúan de ninguna forma, desplazándose sin alterar el movimiento entre ellas. Este modelo de computación es muy útil ya que nos permite ver cómo la información se transmite en los autómatas celulares.

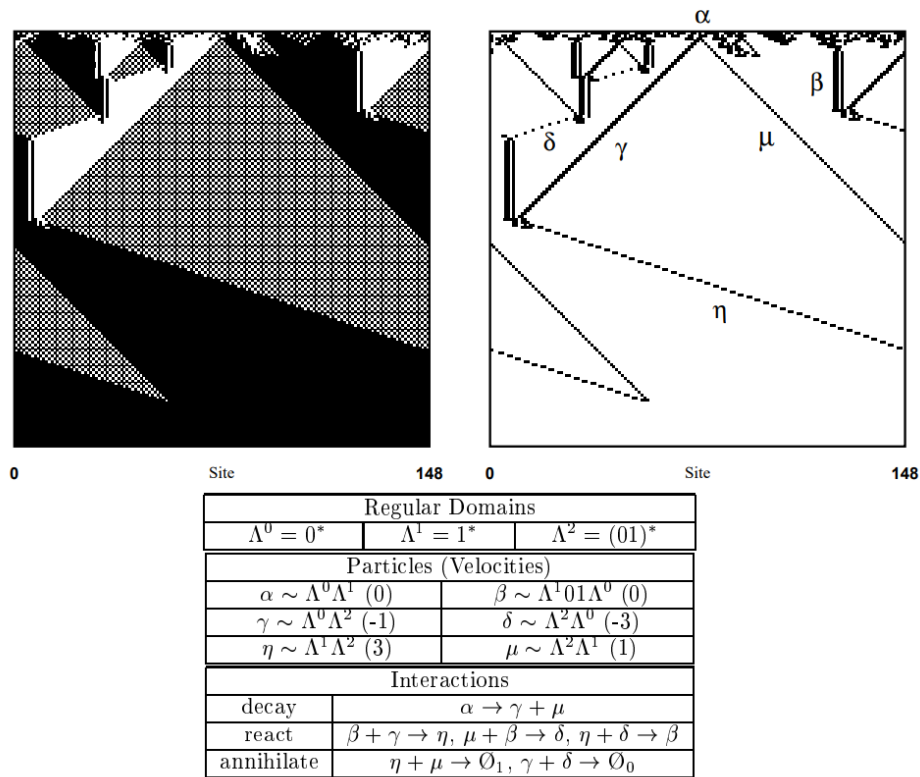


Figura 10: El presente autómata ha sido filtrado para ocultar sus dominios, dejando los defectos ($\alpha, \beta, \gamma, \delta, \eta, \mu$). La tabla adjunta detalla como las partículas interactúan entre sí.

Fuente: [Mitchell *et al.*, 1996]

2.6.1. Dominios

Dada un autómata celular con condiciones de frontera periódica, con función de transición global Φ y alfabeto A , una secuencia de k símbolos $\Lambda \in A^k$ será un dominio de período n si:

$$\Phi^n(\Lambda^*) = \Lambda^* \quad (3)$$

La figura 11 muestra un fondo regular formado por el dominio $(10)^*$ cuyo periodo es 1.

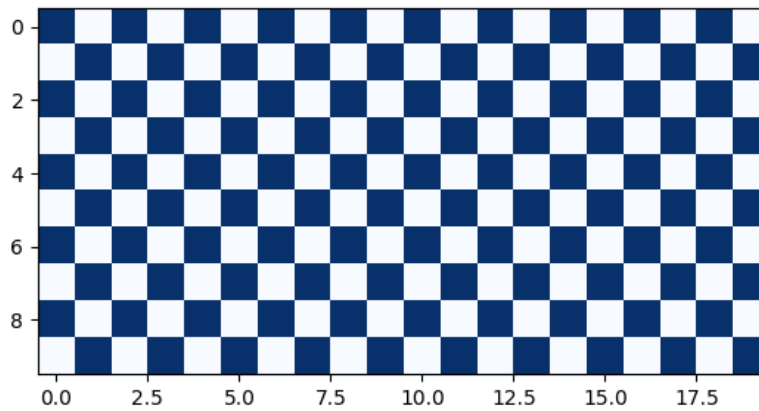


Figura 11: Ejemplo de dominio descrito como $(10)^*$ o $(01)^*$. Presenta un periodo de 1 y proviene de la regla $0504058705000f77037755837bf7b77f$ en su código hexadecimal, encontrado en el paper de Mitchell [Mitchell *et al.*, 1996]

Fuente: Elaboración propia.

2.6.2. Partículas

Las partículas emergen en la interacción de dos dominios. Esto quiere decir que se va a generar una frontera entre los dos dominios que va a tener comportamientos periódicos en el tiempo. También es válido tener un elemento medio entre ambos dominios, como veremos más adelante. En la figura 12 se ven ejemplos de partículas.

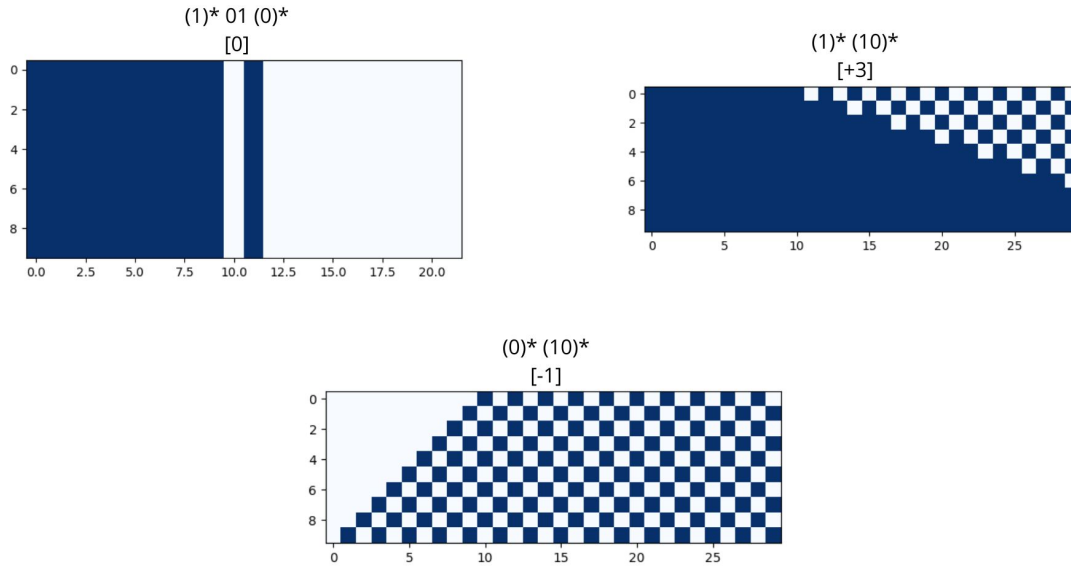


Figura 12: Tres partículas resultantes de las interacciones de los dominios de la regla 0504058705000f77037755837bf7b77f en su código hexadecimal, encontrado en el paper de Mitchell [Mitchell *et al.*, 1996]. En corchete las velocidades de cada partícula.

Fuente: Elaboración propia.

De forma análoga a la definición de un dominio, se puede definir una partícula como una secuencia compuesta por dos dominios (Λ^a y Λ^b) y una secuencia intermedia A^* (que puede ser de largo 0) en la que, si imaginamos un autómata celular sin fronteras, es decir que ambos extremos se expanden infinitamente, tenemos que

$$\Phi^n((\Lambda^a)^* A^* (\Lambda^b)^*) = (\Lambda^a)^* A^* (\Lambda^b)^* \quad (4)$$

donde n sería el valor de la periodicidad temporal de la partícula, y cumple con ser el mínimo común múltiplo de los periodos de los dominios involucrados. Podemos además definir la velocidad de la partícula; puede ser positiva, en donde la partícula se mueve a la derecha; puede ser negativa, en donde la partícula se mueve a la izquierda; y puede ser 0, en donde la partícula no se mueve en el espacio. En la figura 12 se puede ver un ejemplo de cada una de estas velocidades.

Es necesario definir el concepto de la velocidad de las partículas, puesto que identificarla podría no ser tan sencillo a simple vista. En los ejemplos de la figura 12 tuvimos suerte, puesto que el periodo temporal de estos era 1, lo que resulta en que su velocidad sea simplemente el cambio en su movimiento. Este no siempre será el caso, por lo que es necesario definir la velocidad como la razón entre el cambio de posición de nuestra partícula y su periodo

temporal.

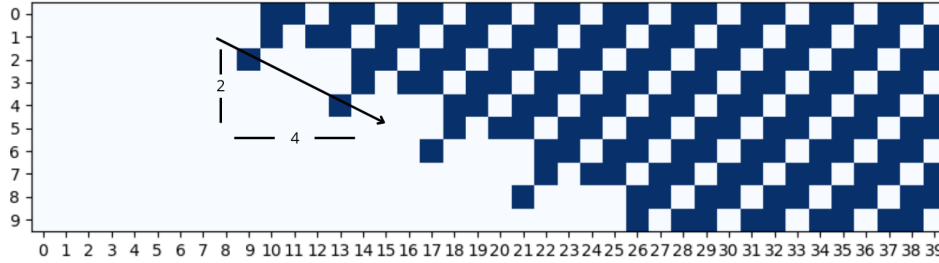


Figura 13: La velocidad de esta partícula $((0^*)(110)^*)$ es de $\frac{4}{2}$. Proveniente de la regla 0504058705000f77037755837bf7b77f en su código hexadecimal, encontrado en el paper de Mitchell [Mitchell *et al.*, 1996].

Fuente: Elaboración propia.

En el ejemplo de la figura 13 podemos ver que el periodo temporal es 2, y necesitamos tomarlo en cuenta para definir su velocidad final como $\frac{\pm 4}{2}$ (4 unidades hacia la derecha por cada 2 iteraciones).

2.6.3. Interacciones

Dos partículas pueden chocar e interactuar, lo que resulta en cero o más partículas. Dadas dos partículas $\alpha = \Lambda^a A^* \Lambda^b$ con velocidad v_α y posición p_α y $\beta = \Lambda^c A^* \Lambda^d$ con velocidad v_β y posición p_β con dominios y alfabeto de un autómata celular, dos partículas pueden interactúan si se cumple que:

$$\Lambda^b = \Lambda^c \wedge v_\beta < v_\alpha \wedge p_\alpha < p_\beta \quad (5)$$

En la figura 14 se observan unos cuantos ejemplos de interacciones entre partículas. Es muy importante señalar que no necesariamente dos partículas que cumplan estas condiciones interactuarán sí o sí en el caso de que estén presentes en la configuración del autómata, si no más bien significa que esta interacción podría suceder, y en el caso de que suceda podremos saber cuál es el resultado. Por ejemplo, dos pares de partículas pueden cumplir con estas

condiciones, pero al mismo tiempo las cumplen con otras partículas. En ese caso, podría ser que las partículas originales interactúen con estas otras antes de acercarse, suceso en el cual podrían ser reemplazadas por otras partículas debido a las interacciones. Se puede ver un ejemplo de esto en la figura 15. De forma análoga, si dos partículas no cumplen con lo anterior no podrán interactuar.

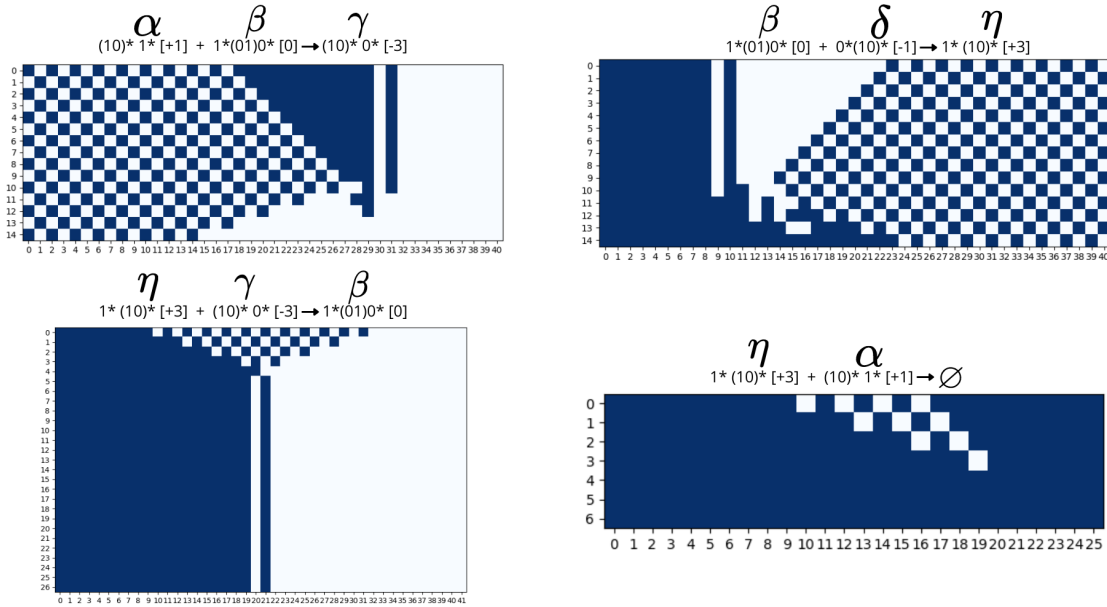


Figura 14: Ejemplos de interacciones entre partículas.

Fuente: Elaboración propia.

Revisemos estas condiciones para que tengan más sentido y nos ayuden a entender mejor a las partículas, que son nuestro principal tema de estudio. Primero, ¿por qué dos partículas no pueden interactuar si los dominios que las unen no son el mismo? Bueno, si no fueran iguales, habría una frontera de dominios que separaría las dos partículas que queremos que interactúen.

Siguiente, ¿Por qué la partícula izquierda debe tener una velocidad mayor a la partícula derecha? Bueno, si no fuera así la partícula izquierda no podría alcanzar a la otra para interactuar. Cabe destacar que esta condición se interesa en las velocidades como valores numéricos. Por ejemplo, si la partícula de la izquierda, α , tiene una velocidad de $+1$ y la partícula de la derecha, β , tiene una velocidad -3 , efectivamente β es más rápida que α , pero nos interesa solamente el hecho de que $-3 < +1$.

Por último, ¿Por qué la partícula con mayor velocidad debe estar a la izquierda, y la partícula con menor velocidad a la derecha? Bueno, en realidad esta última condición se desprende de las otras dos, por lo que no es tan necesario mencionarla. Aún así, es relevante mencionar que, si se cumple la primera condición, no queda otra opción más que se cumpla esta tercera.

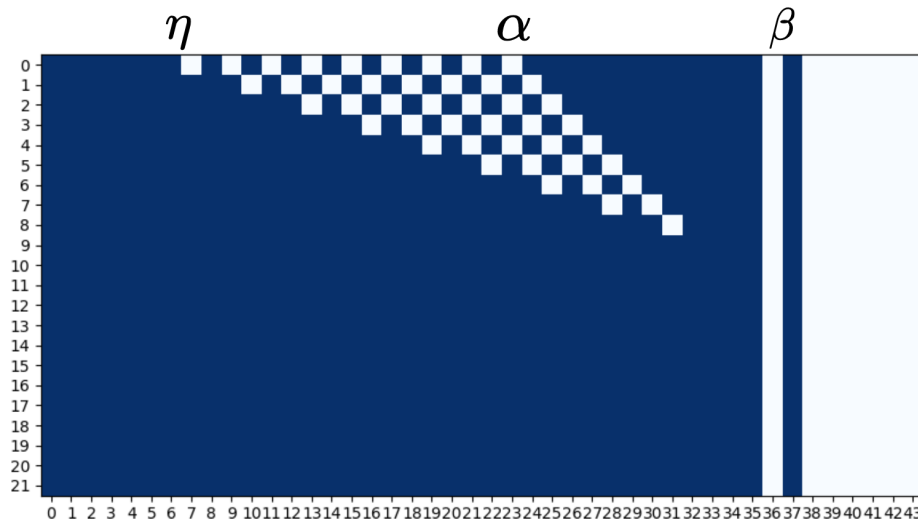


Figura 15: Las partículas α y β cumplen las condiciones necesarias para que se pueda producir una interacción, sin embargo la interacción de α con η sucede primero.

Fuente: Elaboración propia.

Con estas definiciones y conceptos concluye el marco teórico relacionado a los autómatas celulares unidimensionales. La bibliografía relacionada a estas estructuras es extensa, y abarca mucho más de lo visto en este capítulo, pero no es de interés para efectos de lo que se ve en este texto. Además, cabe destacar que en cuanto a detección de dominios, partículas, e interacciones, que sin duda son uno de los principales temas de esta memoria, la academia no ha mostrado ningún indicio de querer detectarlas de forma automática, como se plantea en esta ocasión, sino que estas estructuras emergentes son comúnmente halladas por “inspección” [Hanson y Crutchfield, 1997].

2.7. Herramientas

En cuanto a herramientas tecnológicas usadas para la creación de la propuesta de solución, se presenta *python* como la más importante. El lenguaje *python* ofrece una gran variedad de librerías para casi todas las necesidades, lo cual lo hace muy conveniente. En específico, *python* presenta librerías creadas para el manejo de interfaces de usuario, como *TKinter* y *PyQt*. Estas librerías abstraen la creación de interfaces, lo cual resulta muy útil. Además, *python* es una muy buena opción para mostrar gráficos, gracias a librerías como *matplotlib*, los cuales se usan en esta propuesta de solución para mostrar las simulaciones de los autómatas celulares.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

Como se ha mencionado anteriormente, la propuesta de solución se trata de una herramienta informática capaz de simular autómatas celulares, y encontrar los dominios, partículas, e interacciones para una regla dada por el usuario. Además, con estas estructuras se puede abstraer más allá de los autómatas celulares, siendo el usuario capaz de simular una dinámica de partículas y ver el resultado. Esta herramienta presenta una interfaz diseñada en *python* usando librerías como *TKinter*, *PyQt*, y *matplotlib*, esta última para mostrar los resultados de las simulaciones, tanto de los autómatas celulares como las dinámicas de partículas.

El resto del capítulo se divide en secciones que abordan cada problema o tópico. Cada sección explica el problema que se intenta resolver, y se muestra y se describe el pseudocódigo que lo resuelve, aunque se verá que en algunas secciones puede haber más de un algoritmo, que juntos resuelven el problema. Además, cada sección presenta los costos computacionales temporales de cada algoritmo que se incluye. La primera sección trata el problema de la detección de dominios. Este es el primer problema a resolver que se ve en este capítulo, puesto que, contrario a los demás problemas, no dependen de ningún otro algoritmo que se verá en este capítulo. Por ejemplo, el algoritmo que detecta las partículas, en la segunda sección, supone una lista de dominios, la cual es creada por el algoritmo que lo antecede. Lo mismo sucede con la tercera sección, la cual detecta las interacciones válidas de las partículas, que asume una lista de partículas, la cual es creada por el algoritmo de la segunda sección. De esta forma el capítulo toma una intención en su orden, de manera que facilita la comprensión al lector lo más posible.

3.1. Encontrando Dominios

Recordemos que los dominios son patrones regulares en el tiempo y en el espacio. Debido a que este tipo de dinámicas pueden ser usadas para simular un modelo de cómputo equivalente a la máquina de Turing, tiene como consecuencia que ciertos aspectos relacionados a los dominios y partículas son indecidibles [Pivato, 2007]. Es por esto que, si queremos diseñar un algoritmo capaz de detectar los dominios de una regla dada, se deberá recurrir al uso de heurísticas. En específico, se asume que los patrones que se deben encontrar no superan las n celdas de largo y que tienen una periodicidad de p , dado un n y p variable (en la herramienta se podrán modificar estas variables). Ahora bien, para encontrar estos dominios, podemos usar un algoritmo basado en fuerza bruta.

El algoritmo 1 muestra el procedimiento en pseudocódigo que se encarga de encontrar los dominios. Para generar todos los posibles dominios se crea la función *generate_all_unique_strings*(n, A) (cuya descripción en pseudocódigo se muestra en 2), la cual retorna una lista de todas las secuencias de largo menor o igual a n con símbolos del

Algorithm 1: Detección de Dominios

Input: Función de transición global Φ , alfabeto A , radio r , máxima periodicidad espacial n , máxima periodicidad temporal p

```
1  $domains \leftarrow []$ ;  
2  $candidates \leftarrow generate\_all\_unique\_strings(n, A)$ ;  
3 for  $candidate$  in  $candidates$  do  
4    $CA\_Original \leftarrow candidate \cdot \lceil \frac{2 \cdot r + 1}{|candidate|} \rceil$ ;  
5    $CA\_Copy \leftarrow CA\_Original$ ;  
6   for  $iteration \leftarrow 1$  to  $p$  do  
7      $CA\_Copy \leftarrow \Phi(CA\_Copy)$ ;  
8     if  $shifted\ CA\_Copy = CA\_Original$  then  
9        $domains.append(candidate)$ ;  
10 Return  $domains$ 
```

Algorithm 2: Generación de arreglos únicos invariante por *shift*

Input: Longitud máxima n , tamaño del alfabeto K

```
1  $unique\_arrays \leftarrow \{\}$ ;  
2 for  $length \leftarrow 1$  to  $n$  do  
3    $candidates \leftarrow \text{all } k^{length} \text{ possible strings}$ ;  
4   for  $array$  in  $candidates$  do  
5     if  $is\_shift\_unique(array, unique\_arrays)$  then  
6        $skip \leftarrow \text{False}$ ;  
7       for  $elem$  in  $unique\_arrays$  do  
8         if  $array \cdot |elem| = elem \cdot |array|$  then  
9            $skip \leftarrow \text{True}$ ;  
10           $break$ ;  
11       if not  $skip$  then  
12          $unique\_arrays.add(array)$ ;  
13 Return lista de arreglos en  $unique\_arrays$ ;
```

alfabeto A . Estas secuencias aseguran la minimalidad de los dominios encontrados, puesto que se omiten elementos que son múltiplos de otro, y además se omiten elementos iguales bajo operación de *shift*. Cada candidato se expande en la línea 4 para que el largo sea suficiente y existan vecindarios válidos, de largo $2r + 1$ para un r dado. Aunque el candidato se haya modificado, en la línea 9 se guarda el original. Cada dominio es iterado hasta p veces, y se guarda si en alguna iteración vuelve al estado original.

La complejidad del algoritmo 1 depende de la cantidad de candidatos que se evalúan, y la cantidad de iteraciones por las que se iteran usando la función de transición Φ . La complejidad de la función de transición Φ es más fácil de calcular, puesto que solo depende del largo del autómata, es decir que tiene una complejidad $\mathcal{O}(r)$. La función *generate_all_unique_strings*(n, A) tiene una complejidad de orden $\mathcal{O}(|A|^n \cdot n^2)$, ya que de acuerdo a la ecuación 6 existen $\sim |A|^n$ secuencias sin aplicar las restricciones de minimalidad, y el n^2 es el costo de aplicar las restricciones de *shift* y múltiplos. Aplicar la función de transición Φ por p veces a cada candidato entrega una complejidad total de $\mathcal{O}(|A|^n(n^2 + p \cdot r))$.

En cuanto a la complejidad espacial de este algoritmo, el uso extra de memoria es usado para guardar los candidatos generados, los cuales son en cantidad orden $|A|^n$ y además se genera el autómata celular, pero solo existen en memoria dos de estos a la vez, el original y la copia, lo que resulta en una complejidad espacial total de orden $\mathcal{O}(|A|^n + r)$.

3.2. Encontrando Partículas

En la sección anterior se presenta el algoritmo 1, el cual nos provee de todos los dominios hasta cierta longitud, la cual es un input que el usuario puede modificar. El siguiente paso es encontrar las partículas que se pueden formar con estos dominios. Recordemos que una partícula es una frontera entre dos dominios que se mantiene en el tiempo. Esta frontera puede contener un *string* intermedio, que podemos generar usando la función *generate_all_possible_strings*(n, A) que crea todos los posibles *strings* de largo n con símbolos del alfabeto A . Para esto se hace uso del módulo *product* de la librería *itertools* de *python*. Por ejemplo, para un $n = 3$ y un alfabeto $A = \{0, 1, 2\}$ se tiene que:

$$\begin{aligned} \text{generate_all_possible_strings}(n, A) = \\ [0, 1, 2, 00, 01, 02, 10, 11, 12, 20, 21, 22, 000, 001, 002, \\ 010, 011, 012, 020, 021, 022, 100, 101, 102, 110, 111, 112, \\ 120, 121, 122, 200, 201, 202, 210, 211, 212, 220, 221, 222] \end{aligned}$$

La cantidad de elementos que retorna esta función está dada por la siguiente ecuación:

$$\begin{aligned}
|generate_all_possible_candidates(n, A)| &= |A|^1 + |A|^2 + \dots + |A|^n \\
&= |A| \cdot (1 + |A| + \dots + |A|^{n-1}) \\
&= |A| \cdot (1 + |A| + \dots + |A|^{n-1}) \\
&= |A| \cdot \frac{|A|^n - 1}{|A| - 1} \sim |A|^n
\end{aligned} \tag{6}$$

Algorithm 3: Detección de Partículas

Input: Función de transición global Φ , alfabeto A , radio r , máximo largo del elemento intermedio n , máxima periodicidad temporal p , lista de dominios D

```

1 particles  $\leftarrow []$ ;
2 optional_middle_element_array  $\leftarrow [] + generate\_all\_possible\_strings(n, A)$ ;
3 for domaini, domainj in  $D$  do
4   for middle_element in optional_middle_element_array do
5     if middle_element =  $a \cdot domain_i + b \cdot domain_j$  then
6       /*  $a, b \in \mathbb{N}_0, 1 \leq a + b$  */
7       continue with next middle_element;
8     multiplier  $\leftarrow 2 \cdot r \cdot p$ ;
9      $CA \leftarrow domain_i \cdot \lceil \frac{multiplier}{len(domain_i)} \rceil + middle\_element + domain_j \cdot \lceil \frac{multiplier}{len(domain_j)} \rceil$ ;
10    string_check  $\leftarrow CA[r \cdot p : -(r \cdot p)]$ ;
11    for iteration  $\leftarrow 1$  to  $p$  do
12       $CA \leftarrow \Phi(CA)$ ;
13      if string_check  $\in CA$  then
14        velocity  $\leftarrow \frac{shift(CA\_Copy, CA\_Original)}{iteration}$ ;
15        particles.append(domaini, middle_element, domainj, velocity);
16        continue with next pair of domains;
17 Return particles

```

El algoritmo 3 muestra el procedimiento en pseudocódigo que se encarga de encontrar las partículas. El primer **for** itera sobre todos los posibles pares de dominios en D . en la línea 5 se asegura que el elemento medio no es una concatenación de prolongaciones de los dominios que la rodean, para que de esta forma las partículas que se encuentren estén en su forma más simple posible, donde $a, b \in \mathbb{N}_0$ (también se cumple que $a + b \leq len(middle_element)$).

En la línea 9 se crea una configuración donde los dominios se expanden hacia sus respectivos extremos por una cantidad específica. Esta cantidad asegura que habrá suficiente espacio en el autómata celular para iterar sin problemas por p veces.

Es necesario que los extremos del autómata celular no interactúen para que no interfiera el resultado de esa interacción con la detección que ejecuta el algoritmo. Lo que se requiere para esto es que el autómata sea infinito, en ambos extremos. En la práctica esto no es posible, pero es posible simularlo. Simplemente se toma la configuración y se añaden copias de los extremos ($\lceil \frac{r}{\text{len}(\text{domain}_i)} \rceil$ copias de dominio_i y $\lceil \frac{r}{\text{len}(\text{domain}_j)} \rceil$ copias de dominio_j respectivamente, con un radio r). Luego de aplicar la función de transición, eliminamos estas celdas añadidas y obtendremos nuestra nueva configuración. De esta forma, los dos extremos del autómata no producen contacto, de lo contrario podría interferir con la partícula que se está intentando detectar.

Luego de aplicar la función Φ , se revisa la existencia de *string_check*, el cual es el patrón que se busca para asegurar que la partícula tiene un comportamiento periódico en el espacio y en el tiempo. En caso de que así sea, la partícula se guarda con la información necesaria para describirla. La velocidad de la partícula se calcula como la razón entre la diferencia de localidades donde se encuentra este *string_check* y donde se encontraba en el inicio, y la iteración actual. Esta fórmula es análoga al cálculo de la velocidad física, diferencia de distancia dividido en tiempo. Es importante notar que esta velocidad puede ser negativa, lo que quiere decir que *string_check* fue encontrado más hacia la izquierda de donde estaba inicialmente.

La complejidad del algoritmo 3 depende de la cantidad de dominios en D , la cantidad de elementos medios, y la máxima periodicidad temporal permitida p . La cantidad de elementos medios es igual a $|\text{generate_all_possible_candidates}(n, A)| = |A| \cdot \frac{|A|^n - 1}{|A| - 1} \sim |A|^n$ de acuerdo a lo encontrado en la ecuación 6, donde n es el largo máximo permitido del elemento medio. Entonces se tiene que, para cada configuración dominio-elemento_medio-dominio, de las cuales hay aproximadamente $|D|^2 \cdot |A|^n$, esta se iterará en un autómata celular de largo $2 \cdot r \cdot p$ por p número de iteraciones, y en cada iteración se aplica tanto la función de transición global Φ , la cual tiene complejidad $\mathcal{O}(r \cdot p)$, como la sección del algoritmo que se encarga de revisar que la partícula está presente en la configuración, la cual tiene una complejidad de $(r \cdot p)^2$, ya que se requiere evaluar acaso *string_check* está contenida en CA , y ambas tienen un largo de orden $r \cdot p$. En total, la complejidad del algoritmo 3 es $\mathcal{O}(|D|^2 \cdot |A|^n \cdot p \cdot (rp + (rp)^2))$.

El uso extra de memoria para este algoritmo va dirigido a guardar la lista con las posibles secuencias intermedia que pueden aportar a la estructura de las partículas. Además, se debe crear la lista de partículas que la función retorna, la cual tiene un tamaño de orden $\mathcal{O}(|D|^2 \cdot |A|^n)$. Por último, se debe guardar en memoria el autómata celular y *string_check*, ambos de orden $r \cdot p$. En total la memoria extra para este algoritmo es de orden $\mathcal{O}((|D|^2 + 1) \cdot |A|^n + r \cdot p)$.

3.3. Encontrando Interacciones

En la sección anterior se presentaron los algoritmos 3 y 2 que describen la manera de encontrar partículas para una función de transición global Φ y una lista de dominios. Esta lista de dominios se puede conseguir usando el algoritmo 1. En esta sección se detectan las interacciones que se pueden formar con las partículas encontradas.

Algorithm 4: Detección de Interacciones

Input: Función de transición global Φ , alfabeto A , radio r , lista de dominios D , lista de partículas P

```

1  $interactions \leftarrow \{ \}$ ;
2 for  $particle_i, particle_j$  in  $P$  do
3   if  $particle_i.domain_j \neq particle_j.domain_i$ 
4      $\vee particle_i.velocity \leq particle_j.velocity$  then
5      $\quad$  continue with next particle pair;
6    $multiplier \leftarrow (2r + 1) \cdot max\_particle\_temporal\_period;$ 
7    $p_i \leftarrow particle_i.domain_i + particle_i.middle\_element + particle_i.domain_j;$ 
8    $p_j \leftarrow particle_j.domain_i + particle_j.middle\_element + particle_j.domain_j;$ 
9    $CA \leftarrow particle_i.domain_i \cdot \frac{multiplier}{len(domain_i)} + p_i + p_j + particle_j.domain_j \cdot \frac{multiplier}{len(domain_j)};$ 
10   $collision\_time \leftarrow \lceil \frac{particle_i.domain_j + particle_j.domain_i}{particle_i.velocity - particle_j.velocity} \rceil$ ; for
11     $iteration \leftarrow 1$  to  $collision\_time$  do
12     $\quad CA \leftarrow \Phi(CA);$ 
13   $interactions[(particle_i, particle_j)] \leftarrow [ ];$ 
14  for  $iteration \leftarrow 1$  to  $max\_particle\_temporal\_period$  do
15     $CA \leftarrow \Phi(CA);$ 
16    for  $particle$  in  $P$  do
17       $particle\_conf \leftarrow$ 
18         $particle.domain_i + particle.middle + particle.domain_j;$ 
19      if  $particle\_conf$  in  $CA$  then
20         $\quad interactions[(particle_i, particle_j)].append(particle);$ 
21 Return  $interactions$ 

```

El algoritmo 4 muestra el procedimiento en pseudocódigo que se encarga de encontrar las interacciones entre las partículas. De forma análoga al algoritmo 3, el primer **for** itera sobre todos los posibles pares ordenados de partículas pertenecientes a la lista entregada como *input*. Se verifica que el par de partículas de la iteración cumplan con los requisitos vistos en la ecuación 5. Este filtro temprano es bastante útil ya que ahorra mucho trabajo computacional, debido a que solo una fracción de los pares posibles de partículas podrán pasar este filtro. En la línea 8 se forma el autómata celular, que consiste en la unión de las dos partículas, donde los extremos no interactúan (similar al caso que se vio para el algoritmo 3).

Luego se aplica la función de transición $\Phi_{collision_time}$ veces para que las partículas, inicialmente separadas, puedan colisionar y producir el resultado de esta interacción. El valor de $collision_time$ es el resultado de resolver el siguiente sistema:

$$\begin{aligned}
 particle_i(t) &= 0 + particle_i.velocity \cdot t \\
 particle_j(t) &= (particle_i.domain_j + particle_j.domain_i) + particle_j.velocity \cdot t \\
 particle_i(t) &= particle_j(t) \\
 0 + particle_i.velocity \cdot t &= (particle_i.domain_j + particle_j.domain_i) + particle_j.velocity \cdot t \\
 t &= \frac{(particle_i.domain_j + particle_j.domain_i)}{particle_i.velocity - particle_j.velocity} \tag{7}
 \end{aligned}$$

Se puede ver que la ecuación 7 determina el tiempo en que las partículas colisionan. Solo se debe calcular el techo de este valor, ya que estamos trabajando con espacio discreto, para saber la cantidad de iteraciones necesarias. Luego de esto se aplica la función de transición Φ al autómata celular y se hace registro de cada partícula que se encuentra en la configuración. Finalmente, asociamos en el diccionario la tupla de partículas interactuando a su resultado, el cual es una lista de partículas que fueron encontradas. Para esto, simplemente se revisa acaso cada partícula de P se encuentra en la configuración presente.

El costo computacional del algoritmo 4 visto en esta sección depende mayoritariamente de la cantidad de partículas en P . Afortunadamente no se debe iterar cada uno de los $|P|^2$ posibles pares que se pueden formar, puesto que primero filtramos por los que efectivamente cumplen con los requisitos mostrados en la ecuación 5. Si consideramos el peor caso, cada par posible de partículas deberá ser iterado p veces, donde p corresponde al máximo periodo temporal en la lista P . En cada iteración se debe aplicar la función de transición Φ , la cual tiene un costo lineal con respecto al largo del autómata celular, en este caso depende linealmente de r y p . Además, en cada iteración se revisa acaso cada partícula de P se encuentra en la configuración presente, lo cual tiene un costo de $|P| \cdot (r \cdot p)^2$, ya que se requiere evaluar acaso la partícula está contenida en CA , y ambas tienen un largo de orden $r \cdot p$. El costo computacional de este algoritmo es entonces $\mathcal{O}(|P|^2 \cdot p \cdot (r \cdot p + |P| \cdot (r \cdot p)^2))$.

La complejidad espacial de este algoritmo solo depende del autómata celular creado para revisar el resultado de la interacción. Aunque sean múltiples interacciones las que se deben revisar, solo un autómata celular es guardado en memoria a la vez. Es por esto que el uso extra de memoria es simplemente $\mathcal{O}(r \cdot p)$.

3.4. Filtrando Dominios

Las últimas tres secciones se enfocaron en los algoritmos que detectan los dominios, partículas, e interacciones. En esta sección se presenta una manera de “filtrar” los dominios para cierta configuración de autómata celular. Esto es muy importante para el análisis que pueda

hacer el usuario de la herramienta que se desarrolla como parte de la propuesta de solución, puesto que le facilita la visualización de las partículas que podrían estar presentes. Un ejemplo de este filtro de dominios se puede apreciar en la figura 16.

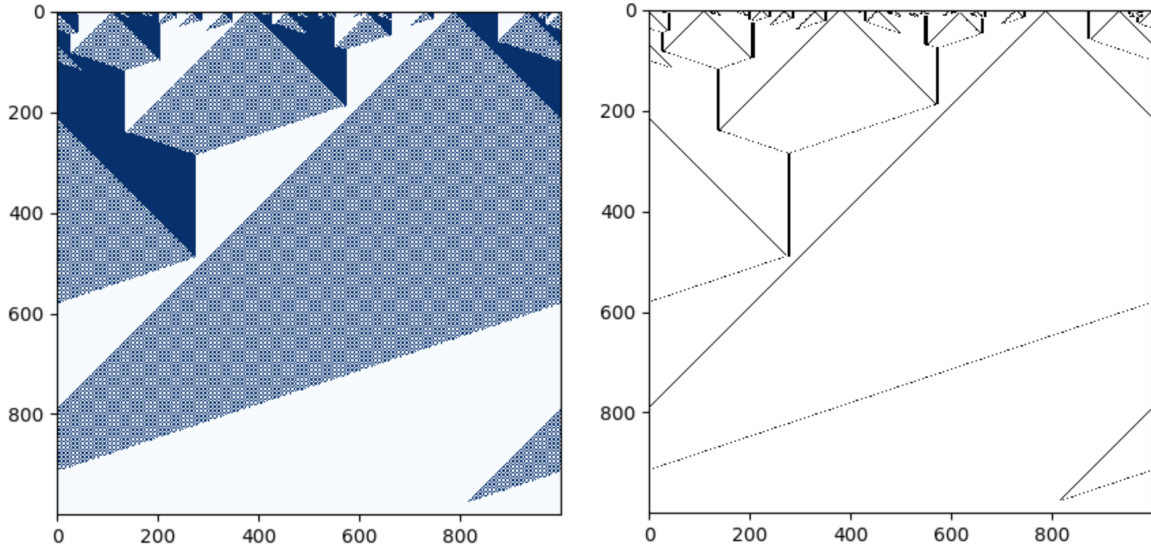


Figura 16: La configuración de la izquierda muestra la simulación de un autómata celular, mientras que la configuración de la derecha filtra los dominios detectados, dejando a simple vista las fronteras.

Fuente: Elaboración Propia.

Algorithm 5: Filtro de Dominios

Input: lista de Dominios D , Autómata celular CA

```

1  $CA\_filtered \leftarrow []$ ;
2 for  $configuration$  in  $CA$  do
3    $configuration\_filtered \leftarrow configuration$ ;
4   for  $domain$  in  $D$  do
5      $configuration\_filtered.replace\_all(domain, [0] \cdot len(domain))$ ;
6    $CA\_filtered.append(configuration\_filtered)$ ;
7 Return  $CA\_filtered$ 

```

El algoritmo 5 toma como argumento la simulación de un autómata celular, y una lista de sus dominios detectados. Como la simulación del autómata celular es una lista de configuraciones (las cuales son listas de símbolos $\in$ alfabeto A) podemos aplicar el filtro a cada configuración por separado. Las posiciones en donde se encuentran dominios se pueden marcar con un 0 y 1 si no hay. Finalmente, se junta todas estas nuevas configuraciones filtradas y se retorna $CA_filtered$.

El costo computacional de este algoritmo depende de las dimensiones del autómata celular, es decir su ancho y su largo, CA_width y CA_height respectivamente. Además la complejidad del algoritmo depende de la cantidad de dominios en $|D|$. En total se tiene que, para cada una de las CA_height configuraciones, se debe reemplazar cada ocurrencia de $|D|$ dominios, cuyo costo es proporcional al largo de la configuración, y todas son de largo CA_width . Entonces el costo del algoritmo 5 es de orden $\mathcal{O}(CA_width \cdot |D| \cdot CA_height)$.

La memoria extra usada para este algoritmo es simple de calcular, ya que solo se guarda en memoria la matriz con dimensiones igual a la del autómata celular original, lo que resulta en que la complejidad espacial sea orden $\mathcal{O}(CA_width \cdot CA_height)$.

3.5. Dinámica de partículas

Esta sección comprende uno de los temas fundamentales de esta memoria, la simulación de dinámica de partículas. Para esto es necesario olvidar momentáneamente a los autómatas celulares, puesto que aunque se parecen en estructura, la forma en que se construyen e interactúan a través de las iteraciones es un poco distinta. De la misma forma en que se han llevado a cabo las secciones anteriores, se presenta un algoritmo que describe el funcionamiento detrás de este tipo de simulaciones.

Algorithm 6: Simulador de Dinámica de Partículas

Input: lista de partículas P , diccionario de interacciones D , número de iteraciones n , largo w

```
1 for  $iteration \leftarrow 1$  to  $n$  do
2    $new\_particles \leftarrow []$ ;
3   for  $p_1, p_2$  in  $P$  do
4     if  $are\_close(p_1, p_2) \wedge p_1.pos \leq p_2.pos \wedge (p_1.type, p_2.type) \in D.keys$  then
5        $new\_particles.append((D[(p_1.type, p_2.type)], p_1.position))$ ;
6        $P.remove(p_1)$ ;
7        $P.remove(p_2)$ ;
8   for  $particle$  in  $P$  do
9      $particle.position \leftarrow (particle.position + particle.velocity) \% w$ ;
10  for  $tuple$  in  $new\_particles$  do
11     $i \leftarrow 0$ ;
12    for  $particle$  in  $tuple[0]$  do
13       $P.append(Particle(particle, tuple[1] + i))$ ;
14       $i++$ ;
15 Return
```

El algoritmo 6 muestra el manejo de las partículas, su movimiento, y sus interacciones. En

esta implementación las partículas son objetos que tienen asociados atributos como tipo, posición, y velocidad. Además se tiene acceso a un diccionario que describe la forma en que las partículas interactúan. El atributo tipo es necesario para encontrar el resultado, como se muestra en la línea 4. El diccionario de interacciones se puede encontrar usando el algoritmo 4 visto unas secciones atrás. En la línea 3 se itera sobre cada par ordenado de partículas pertenecientes a P . Luego, para cada par, en la línea 4 se revisa si están cerca. La función $are_close(a, b)$ toma dos partículas y retorna un *bool* que es verdadero si se cumple que:

$$b.pos + b.velocity \leq a.pos + a.velocity$$

Es importante notar que para que tenga sentido usar la función $are_close(a, b)$ necesitamos que $a.pos < b.pos$, lo cual se revisa en la línea 4 del algoritmo. En caso de que ocurra una interacción, las partículas son eliminadas y reemplazadas por las resultantes en las mismas posiciones. Luego se actualiza la posición de cada partícula del arreglo, lo cual significa sumarle su velocidad a la posición actual. Finalmente, agregamos las nuevas partículas que fueron creadas en esta iteración. Para esto se tiene la clase *Particle* que recibe dos argumentos, el tipo de partícula, a la cual podemos asociar su velocidad, y la posición inicial. Para visualizar esta simulación solo hace falta hacer un registro de cada posición en cada iteración. La figura 17 muestra un ejemplo de simulación de partículas. La idea de crear este sistema de simulación es que luego será muy importante a la hora de extender la herramienta de simulación de autómatas celulares hacia una visualización más abstracta, inclinada al interés de dinámica de partículas, con el objetivo de que el usuario pueda trabajar con las partículas que fueron encontradas por el algoritmo 3 y que se comportan según lo descrito por el algoritmo 4. Se recuerda que lo anteriormente mencionado es uno de los objetivos específicos de esta memoria, definido por el segundo.

La complejidad del algoritmo 6 depende de la cantidad de iteraciones totales n , y la cantidad de partículas presentes al inicio de la simulación, específicamente la cantidad de pares posibles de partículas $|P|^2$. Durante la ejecución del algoritmo no se sabe cómo va a variar P exactamente, pero $n \cdot |P|^2$ entrega un costo proporcional al que se vería en una ejecución real del algoritmo. Además se debe considerar que en cada iteración se itera sobre cada partícula para actualizar su posición, lo cual tiene un costo de orden $\mathcal{O}(|P|)$. En total la complejidad de este algoritmo es $\mathcal{O}(n \cdot (|P|^2 + |P|))$.

Este algoritmo solo necesita memoria extra para guardar las nuevas partículas que se generan en cada iteración. Afortunadamente solo se debe guardar las partículas que se generan en la iteración actual, mientras que las iteraciones pasadas no tienen necesidad de permanecer en memoria. Esto permite que la complejidad espacial sea simplemente $\mathcal{O}(|P|)$.

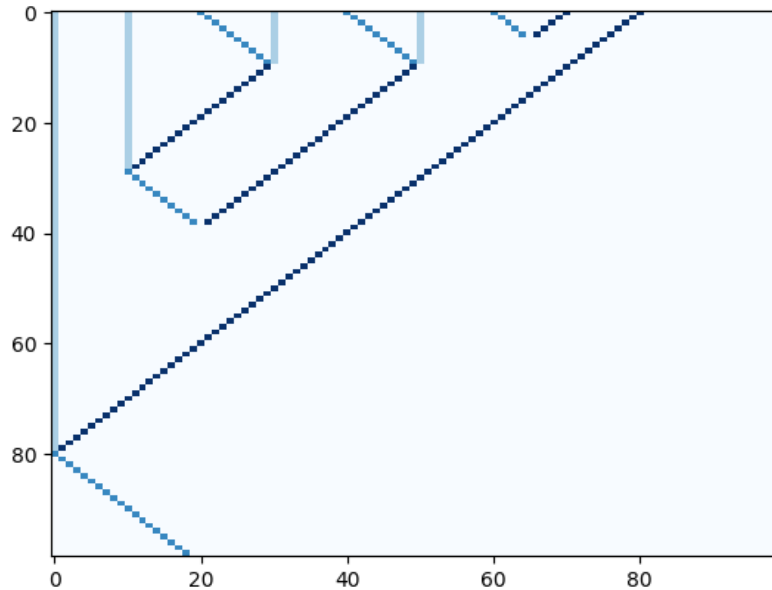


Figura 17: Ejemplo de partículas en dinámica. Se puede ver una interacción que resulta en destrucción. Color de la celda indica tipo de partícula, con blanco siendo el fondo.

Fuente: Elaboración Propia

3.6. La herramienta

A lo largo de todo el capítulo se han visto los algoritmos necesarios para detectar las estructuras bajo análisis en esta memoria. Además, se explica el método de la simulación de la dinámica de partículas. Todos estos procesos son necesarios para cumplir los objetivos planteados en el capítulo de la definición del problema. Ahora bien, otro de los objetivos que se plantean hace referencia a la creación de una herramienta acompañada por una interfaz, la cual tiene como objetivo que el usuario pueda subir cualquier regla y analizarla haciendo uso de los algoritmos implementados en el lenguaje de programación seleccionado, en este caso *python*.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

La propuesta de solución de esta memoria es una herramienta, que consiste en una interfaz donde el usuario es capaz de simular autómatas celulares unidimensionales, pudiendo ingresar cualquier regla. La herramienta permite al usuario extender el estudio y análisis de la regla seleccionada, con la detección de las estructuras emergentes. Con esto, el usuario puede experimentar, simulando autómatas celulares de distintos tamaños, pudiendo además filtrar los dominios como fue visto en el algoritmo 5. Por último, el usuario puede trabajar solo con las partículas encontradas, y crear nuevas configuraciones en las que se observa la dinámica de estas partículas con sus interacciones.

Para la creación de esta interfaz, se hace uso de la librería *TKinter*, como se mencionó en el apartado de metodologías del segundo capítulo. *TKinter* ofrece casi todos los elementos necesarios para que se puedan realizar todas las acciones que se esperan de una herramienta relacionada a los autómatas celulares. El resto de los elementos necesarios los provee la librería *PyQt*, también mencionada en metodologías. Esta librería se encarga de la tarea que le permite al usuario crear configuraciones de dinámica a partir de las partículas encontradas. La forma en que estas configuraciones son creadas es con un sistema de “*drag and drop*”, lo cual le permite al usuario usar su “mouse” para seleccionar las partículas que desea en la simulación, y en la posición que requiera. Más detalle del funcionamiento y propósito de este *feature* se discutirá en las próximas secciones.

El resto del capítulo de validación de la solución se divide en secciones que revisan cada funcionalidad de la herramienta creada, indicando los algoritmos usados, describiendo los elementos que forman la interfaz, y demostrando su correcto funcionamiento mediante ejemplos explícitos. La primera sección revisa la correcta simulación de autómatas celulares unidimensionales y la interfaz que le permite al usuario crearlas. La segunda sección se adentra en la segunda parte de la herramienta, y revisa la detección de los dominios, partículas, e interacciones válidas. La última sección revisa la funcionalidad que le permite al usuario crear nuevas configuraciones a partir de las partículas encontradas, en donde se muestran instancias reales que demuestran los resultados esperados.

4.1. Simulando autómatas celulares

Recordemos que cualquier autómata celular unidimensional está determinado por la función de transición global Φ , su radio r , y su alfabeto A . Otras características que se podrían considerar a la hora de simular un CA son el largo del autómata y la cantidad de iteraciones. En la herramienta propuesta, la función de transición toma la forma de una función de *python* que el usuario selecciona desde la herramienta. Existen condiciones que debe tener este archivo *python* que contiene la regla. El archivo debe contener una función llamada *rule*

que tome como argumento una lista, y retorne un número $1 \leq x \leq |A|$ donde $|A|$ es la cantidad de estados. En la figura 18 se observan todas las configuraciones para simular el autómata celular.

Como se puede apreciar en la figura, el usuario tiene acceso a diferentes *inputs* para simular cualquier autómata celular que requiera. Las entrada *Width* y *Height* controlan las dimensiones del autómata celular (largo y número de iteraciones respectivamente). *Radius* y *N. of states* deben ser ajustados por el usuario de acuerdo a la regla ingresada. Esto tiene como objetivo ayudar a los algoritmos de este programa, puesto que no tienen manera de saber cuales son los valores correctos para la regla ingresada. El usuario tiene acceso mediante esta herramienta a ajustar la configuración inicial. Esto es posible mediante 3 opciones; la primera opción crea una configuración inicial aleatoria, donde cada celda toma un valor aleatorio entre 0 y $k - 1$, donde k es el tamaño del alfabeto; la segunda opción inicializa la configuración inicial con 0 en todas las celdas, excepto en la celda del medio, que se inicializa con un 1; la última opción le otorga al usuario decisión completa, permitiéndole ingresar cualquier configuración inicial que requiera. Esta última opción, debido a que da la mayor libertad al usuario, también es la más difícil, debido a que el usuario deberá ingresar el valor de cada celda.

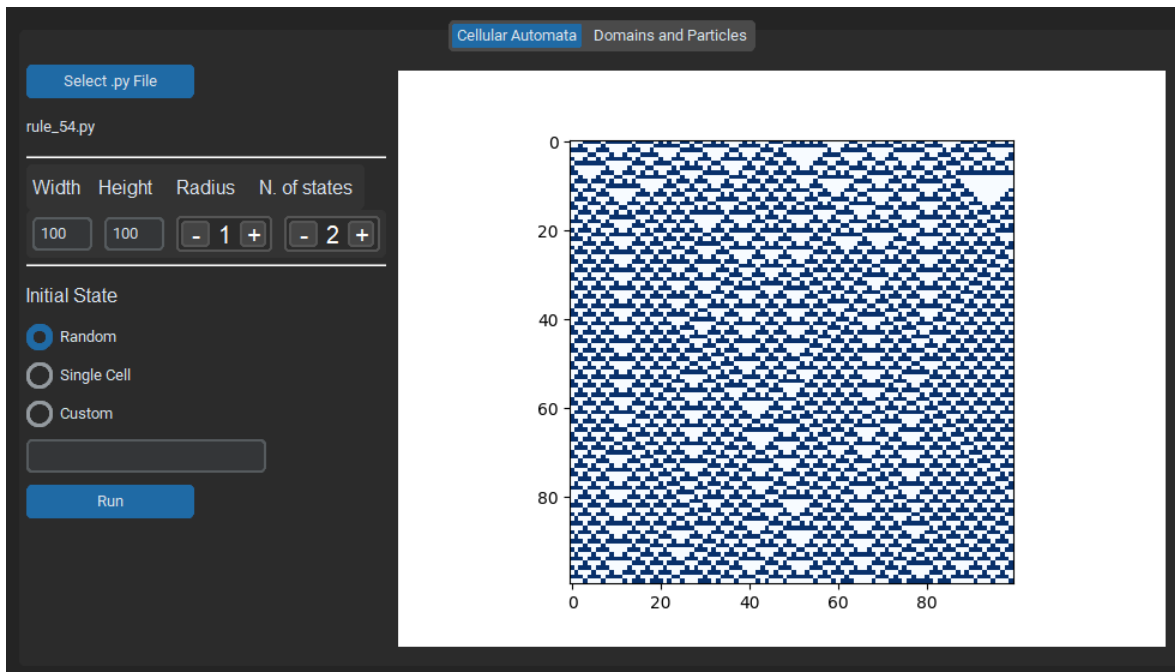


Figura 18: Sección de Autómatas Celulares de la herramienta creada.

Fuente: Elaboración Propia

Para validar el correcto funcionamiento de la herramienta a la hora de simular autómatas, es necesario comparar sus resultados con aquellos que ya se conocen con anterioridad. Afortunadamente, esto es muy sencillo ya que podemos usar la solución al problema de sincronización de Mazoyer [Mazoyer, 1987] y validar a simple vista que los resultados son los mismos.

ción tangible. La figura 20 muestra que los dominios, partículas, e interacciones encontradas por la propuesta de solución son los mismos al paper mencionado.

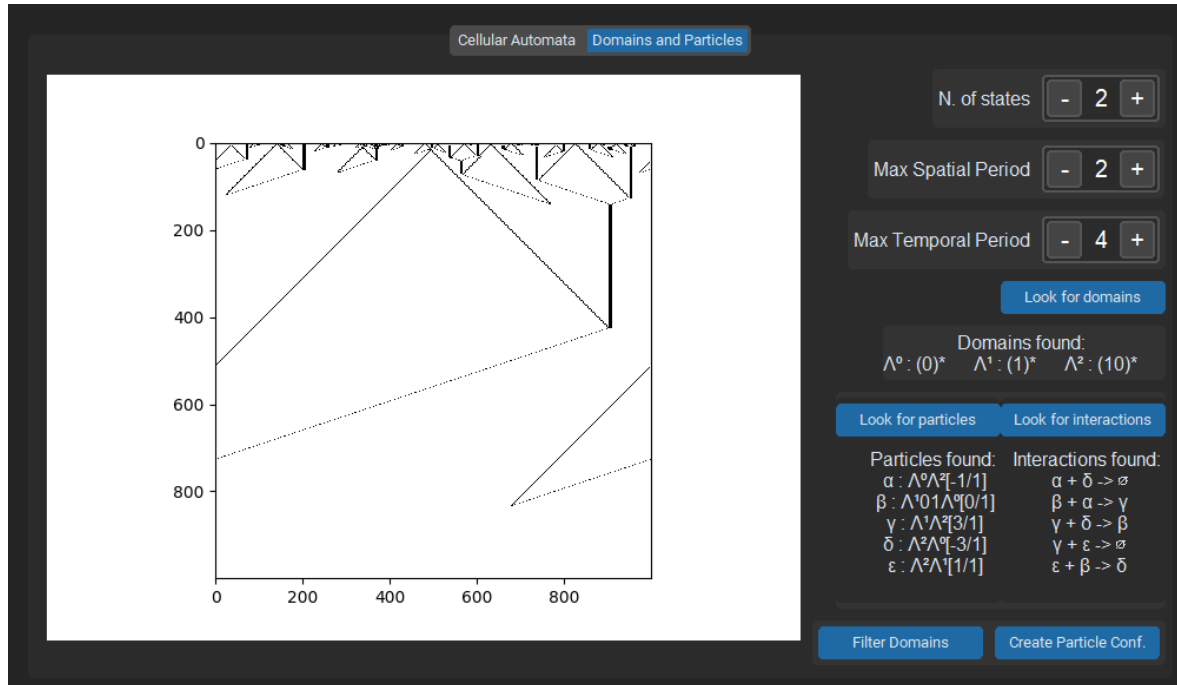


Figura 20: Sección de Dominios y Partículas de la herramienta creada. La imagen muestra una configuración con sus partículas filtradas.

Fuente: Elaboración Propia

4.3. Abstrayendo hacia la simulación de partículas

En cuanto a la segunda ventana de la herramienta, la única acción que queda por revisar es la relacionada a la tarea de crear una configuración a partir de las partículas encontradas. Es en esta tarea donde el *feature* de “*Drag and Drop*” es utilizado. Cuando el botón “*Create Particle Conf.*” es seleccionado, se abre una ventana creada con la librería PyQt. Esta ventana maneja el evento de la creación de la configuración. Aquí las partículas pueden ser arrastradas a cualquier posición (lo que importa es donde están puestas horizontalmente). Luego, cuando la configuración está lista, se selecciona el botón “*ready*” para cerrar la ventana y pasar la información de la configuración hacia la ventana principal; la figura 22 muestra una instancia de este *feature*.

La configuración creada es utilizada para simular la dinámica de las partículas escogidas en las posiciones donde el usuario las colocó. Luego, en caso de que la configuración sea válida para crear el autómata celular correspondiente, ésta se creará y se mostrará en la ventana de autómatas celulares dentro de la herramienta. La configuración es válida si cada par de partículas adyacentes comparten el mismo dominio interno (es decir, la partícula izquierda

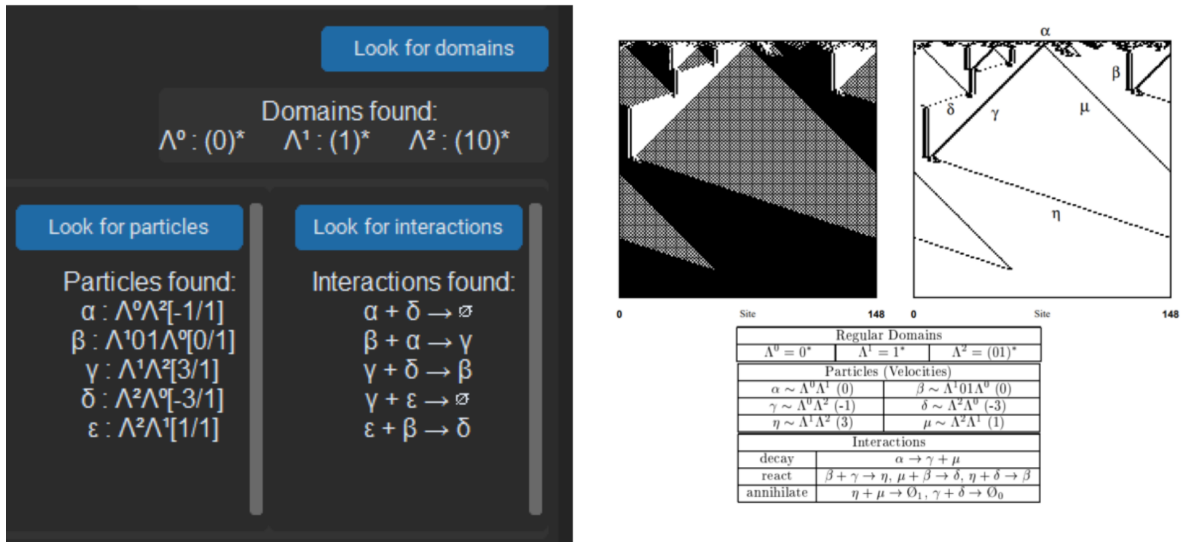


Figura 21: Los resultados de la propuesta de solución y los encontrados en [Mitchell *et al.*, 1996] son idénticos.

Fuente: Elaboración Propia (Imagen izquierda) y [Mitchell *et al.*, 1996] (Imagen derecha)

presenta un dominio derecho que es igual al dominio izquierdo de la partícula derecha). Estos pares de partículas incluyen las dos partículas que existen en cada borde. En la figura 23 se observa un ejemplo de los resultados de la secuencia de tareas que se lleva a cabo dentro de la herramienta, desde la creación de la dinámica de partículas, su simulación, hasta la recreación de esta en su forma de autómata celular.

4.4. Resultados en algunas reglas elementales

Las reglas elementales, según lo visto en el capítulo dos, son las 256 posibles reglas de transición local con 2 estados y radio de vecindad 1. Algunas de estas son estudiadas más que otras en la academia. En relación a los dominios y partículas, existen reglas que no exhiben esta propiedad emergente de manera interesante. Por ejemplo, la regla 60, como se aprecia en la figura 25, no presenta ninguna interacción entre sus partículas, sino más bien decaen en sí mismas. La regla número 122 también fue usada, donde la herramienta pudo encontrar las partículas existentes y la interacción que había. Aunque la herramienta haya encontrado los resultados correctos, la regla 122 no presenta comportamientos particularmente interesantes, como se puede apreciar en la figura 26. Como se puede ver, la herramienta pudo traducir la configuración de partículas al autómata celular que genera la misma interacción.

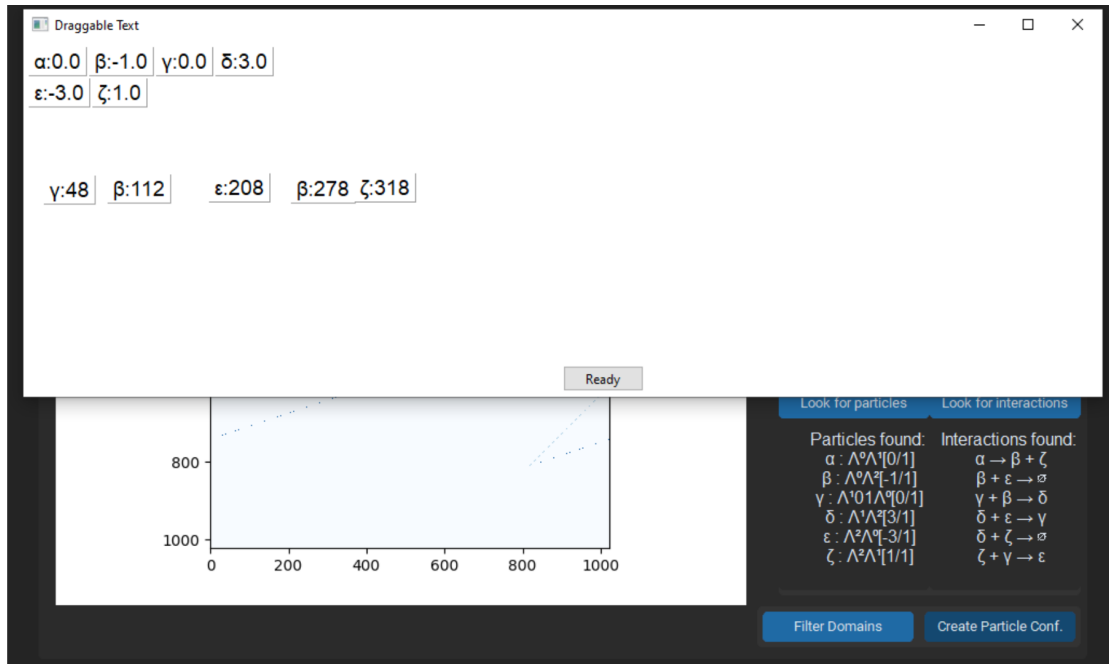


Figura 22: Creador de la configuración de partículas en dinámica. Las cajas de la esquina muestra las partículas disponibles, indicando su velocidad. Las demás cajas son instancias que el usuario ha creado, donde se indica su posición.

Fuente: Elaboración Propia

La regla elemental número 54 fue probada adicionalmente, principalmente debido a que es una de las reglas con más interés, estudiada por Hanson y Crutchfield, por ejemplo [Hanson y Crutchfield, 1997]. Como se puede ver en las figuras 27, 28, 29, 30, 31, y 32, algunas de las partículas encontradas por la herramienta son las mismas encontradas en el artículo de Hanson y Crutchfield mencionado anteriormente. En relación a los resultados de esta memoria y los de ese artículo cabe mencionar que existen discrepancias en cuanto a los resultados de las diversas interacciones que están presentes, debido a que las partículas producen resultados distintos dependiendo de la distancia que las separa, ya que es esta distancia la que determina en que fase va a estar cada partícula a la hora de colisionar. La figura 24 muestra un ejemplo donde la distancia que separa las partículas influye en la creación de CA correspondiente a la interacción. Por último, se usa esta regla para probar el correcto funcionamiento del filtrado de dominios. Como se puede ver en la figura 33, algunas de las estructuras que se pueden ver a simple vista se mantienen al aplicar el filtro de dominios.

Encontrar alguna regla elemental que exhiba partículas e interacciones interesantes quizás sea una tarea ardua pero que puede valer la pena, sin embargo esta tarea se escapa de los objetivos propuestos en esta memoria, en cuanto esta sección tiene el propósito introducir los resultados de algunas reglas elementales para demostrar el correcto funcionamiento de la herramienta creada.

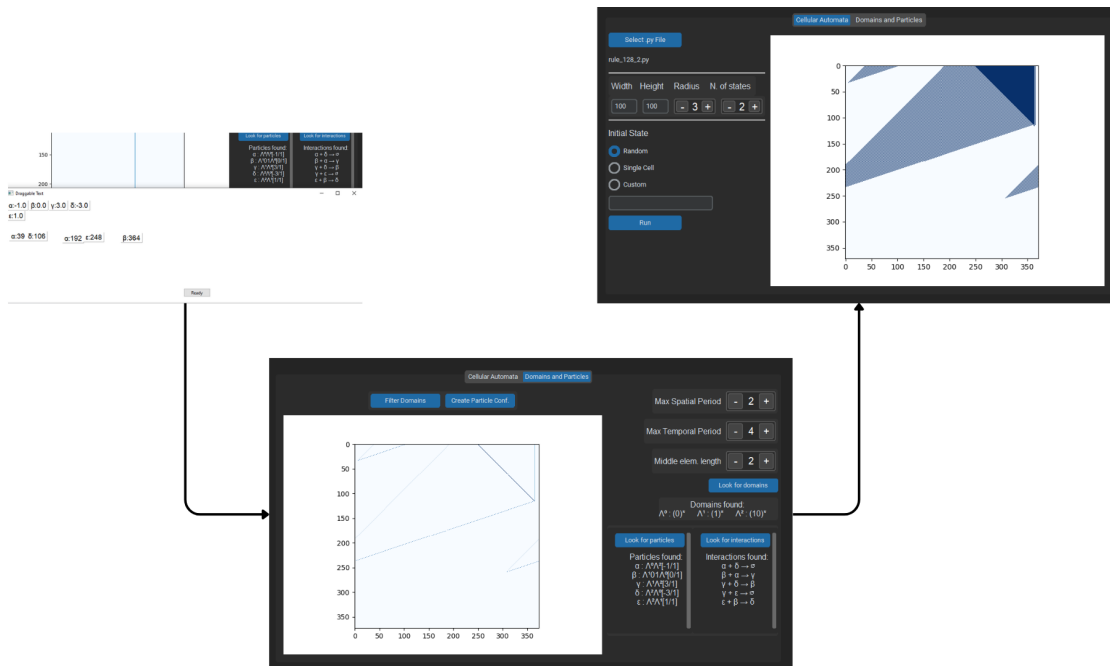


Figura 23: Creación del autómata celular a partir de la configuración de sus partículas encontradas.

Fuente: Elaboración Propia

CAPÍTULO 5

CONCLUSIONES

En esta sección, la última de la presente memoria, se intenta agrupar todo lo discutido a lo largo de las secciones anteriores, con el propósito de concluir sobre el trabajo que inspiró el desarrollo de esta memoria junto con la propuesta de solución. Además, es oportuno reflexionar, en esta sección, acerca de los límites de esta solución propuesta, con el objetivo de definir los puntos más importantes que deberían ser desarrollados aún más en trabajos futuros.

El trabajo del área de los autómatas celulares se remontan a los trabajos de von Neumann relacionado con las máquinas capaces de auto-producirse, hace casi 60 años. Desde entonces han sido objeto de interés en el área de sistemas dinámicos, para estudiar los modelos caóticos más simples. Los más interesantes de estos modelos son aquellos que manifiestan propiedades emergentes, las cuales no pueden ser explicadas mediante el análisis de los elementos por separado, sino más bien si se estudia el sistema como un todo.

En el estudio de los autómatas celulares, las partículas fueron de interés a partir de finales del siglo pasado, con trabajos de Hanson y Crutchfield, por ejemplo. Este tipo de propiedad emergente puede ser usado para explicar métodos en los cuales un autómata celular puede

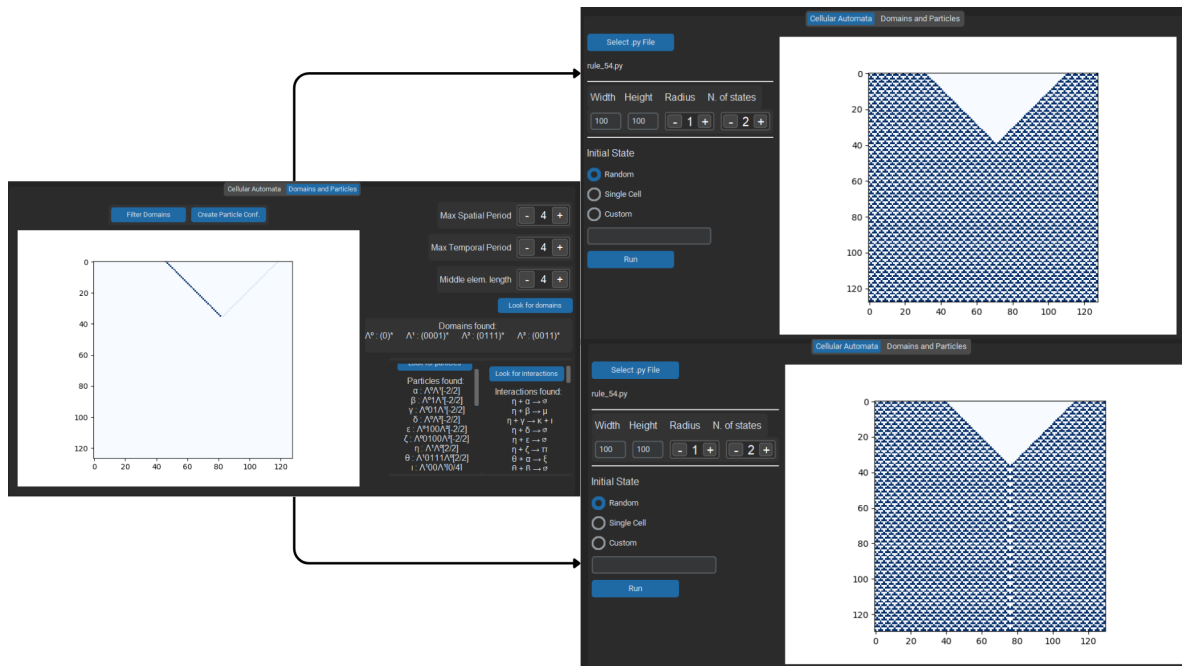


Figura 24: Ejemplo de discrepancia por fase para la regla 54.

Fuente: Elaboración Propia

lograr una comunicación global mediante numerosas comunicaciones locales. En esta comunicación global participan defectos, o partículas, cuya existencia es el resultado del choque de dos dominios, los cuales son patrones regulares en el espacio y en el tiempo.

En la literatura actual no hay intentos de detectar de forma automática estas estructuras, es decir, partículas y dominios, por lo que el trabajo realizado en esta memoria es un avance en un nuevo enfoque hacia el estudio de los autómatas celulares. Los algoritmos presentados en el capítulo tres muestran una manera directa de encontrar estos dominios y partículas, además de las interacciones que existen. Debido a que los dominios y partículas tienen una estructura definida, los algoritmos creados se aprovechan de esto para detectarlas, aplicando la función de transición Φ sobre una configuración, específica para cada estructura que se intenta detectar, y revisando si se mantiene la estructura en el tiempo. Debido a la naturaleza caótica de los autómatas celulares no es posible asumir mucho acerca del comportamiento de cualquier regla *a priori*.

Los algoritmos presentados en esta memoria son capaces de detectar los dominios y partículas de la regla entregada, que sin embargo presentan un costo computacional que demuestra la complejidad del problema. La herramienta creada, que forma la propuesta de solución, le permite al usuario desarrollar un análisis del autómata celular que tenga actualmente bajo su interés. En cuanto a la dinámica de partículas, los algoritmos que la ejecutan fueron desarrollados y expuestos en esta memoria para el correcto cumplimiento de los objetivos dispuestos a la solución del problema. Combinando la simulación de autómatas celulares, la detección de sus partículas emergentes y una manera de estudiar su dinámica bajo un

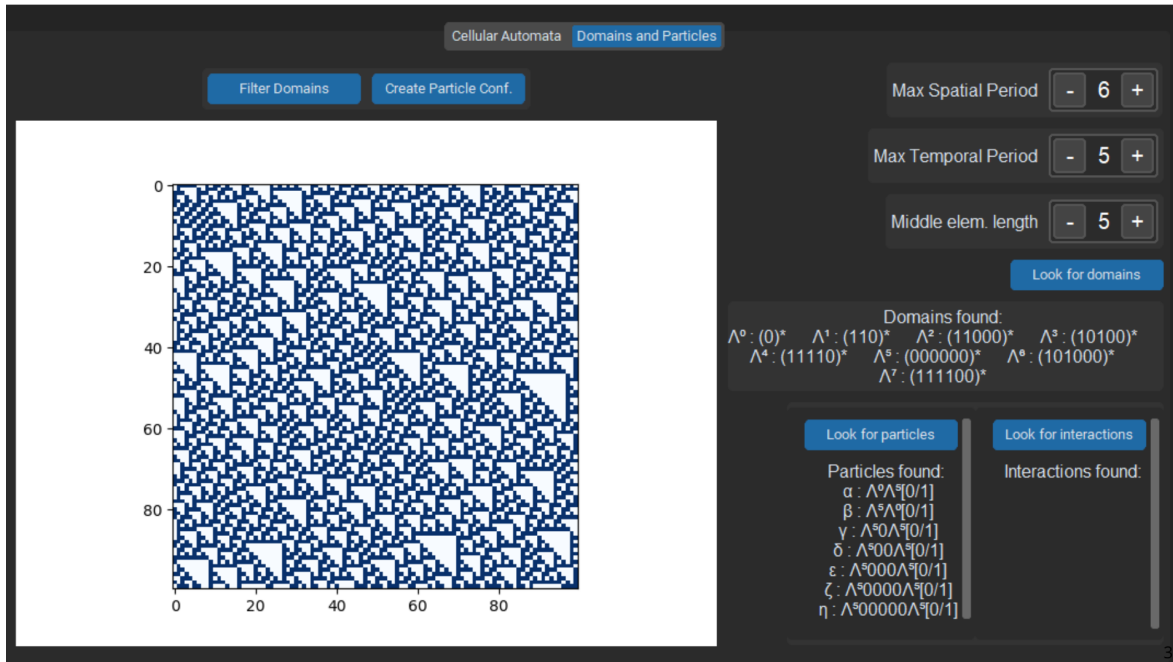


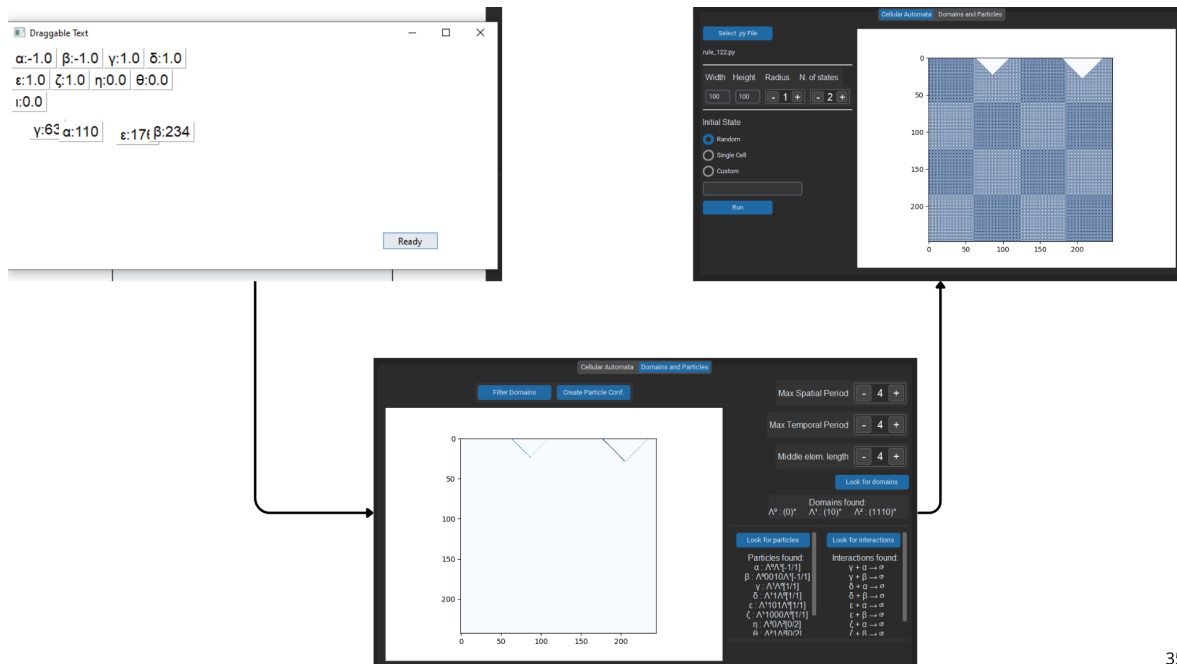
Figura 25: Resultados de la regla 60. No existe ninguna interacción que no resulte en destrucción.

Fuente: Elaboración Propia

plano distinto para su mejor análisis, se le otorga al usuario herramientas muy útiles que no estaban a su alcance anterior a este desarrollo.

La herramienta fue probada en algunas reglas elementales, con el objetivo de conseguir resultados que puedan validar el correcto funcionamiento de la propuesta de solución. Las reglas elementales usadas fueron la 60, la 122 y la 54. Para las primeras dos reglas los resultados fueron exitosos pero poco interesantes, en cuanto no se encontraron partículas que interactuasen de alguna forma que valga la pena analizar con mayor profundidad. La regla 54 entregó resultados particularmente interesantes, y estos se pudieron comparar con los resultados que existen en la academia respecto a esta regla elemental. Primero, la herramienta pudo encontrar las partículas descritas en el artículo de Hanson y Crutchfield [Hanson y Crutchfield, 1997] junto con otras. Luego, se demostró la correcta predicción de algunas interacciones, comparándolas con el resultado que muestra el artículo mencionado, donde se encuentran resultados diferentes en las interacciones. Esta discrepancia en las interacciones podría ser explicada si se toma en cuenta que dos partículas podrían colisionar en distintas fases de su periodo temporal, lo que podría causar distintos resultados en las interacciones.

El objetivo general de esta memoria era el desarrollo de una herramienta que facilite el estudio de la dinámica de autómatas celulares 1D a través de construcciones basadas en partículas emergentes. De acuerdo a lo visto en la propuesta de solución su validación, el desarrollo y la innovación fueron destinados hacia el cumplimiento de este objetivo. Los ob-



35

Figura 26: Resultados de la regla 122. No existe ninguna interacción que no resulte destrucción.

Fuente: Elaboración Propia

jetivos específicos, de forma similar, fueron cubiertos con soluciones expuestas en el capítulo tres, y la construcción de la herramienta se demuestra en el capítulo cuatro.

Entrando más a detalle, el primer objetivo específico requería el desarrollo de una herramienta capaz de crear cualquier autómata celular de una dimensión. Dadas la cantidad de posibles reglas de transición, la herramienta se diseñó pensando en darle la mayor autonomía al usuario como fuera posible. El resultado de esto es que el usuario debe crear una regla, en forma de un archivo *python*, que contenga dicha función de transición local. La figura 18 muestra una instancia de esta herramienta creada.

Siguiendo con el segundo objetivo específico, la herramienta dispone de una funcionalidad que le permite al usuario crear configuraciones para simular colisiones de partículas. Estas partículas son detectadas de acuerdo al tercer objetivo específico, el cual se discutirá más adelante. De acuerdo a lo observado en la figura 22, se demuestra una interfaz que le permite al usuario arrastrar con el *mouse* cada partícula que requiera a la posición exacta que necesite.

El tercer objetivo específico es uno de los más innovadores en la propuesta de solución de esta memoria, y por lo tanto es también la más compleja y la que ocupa el mayor espacio del capítulo tres. Los dominios y partículas son secuencias de carácter periódico, tanto en el espacio como en el tiempo, por lo que su estructura se puede determinar y detectar. Esto implica, por lo tanto, detectar además aquello que no es un dominio o una partícula. Para

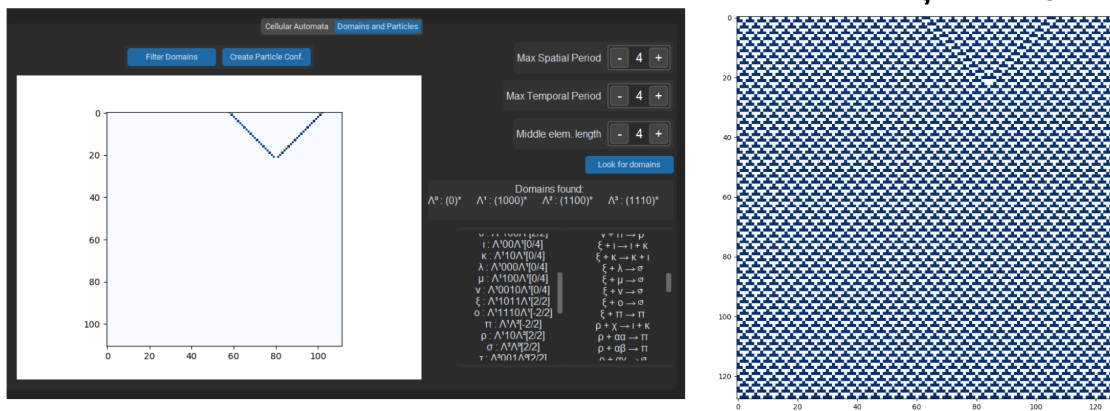


Figura 27: El resultado de la presente colisión es predicha correctamente por la herramienta.

Fuente: Elaboración Propia

esto se requiere, en el caso de los dominios, formar todas las secuencia posibles con los parámetros entregados, y filtrarlas, dejando solamente aquello que es un dominio, de acuerdo a lo descrito por la ecuación 3.

El último objetivo específico valida el correcto funcionamiento de la herramienta creada, pudiendo demostrar que efectivamente es capaz de lograr las tareas requeridas, como la simulación de un autómata celular. En el capítulo de validación de solución se muestran diferentes instancias de la herramienta, junto con una simulación de la cual se sabe con anterioridad el resultado final, y todo el proceso que hay en medio, como se ve en las figuras 19, 20, y 32.

Por último, al ser el objetivo general una generalización y agrupación de los objetivos específicos, se puede concluir que esta ha sido validado por los resultados mostrados. A partir de esto, se puede revisar el árbol del problema visto en el primer capítulo. Se recuerda que el problema principal era las limitaciones en las herramientas de simulación de autómatas celulares existentes. Las causas se referían a la existencia de tal herramienta, cuya construcción podría no ser trivial. Con los trabajos realizados en la creación de la herramienta expuesta en esta memoria, se puede confirmar con mayor seguridad esta afirmación. Los efectos del problema principal es la existencia de perspectivas de estudio sobre los autómatas celulares las cuales aún no han sido descubiertas, y reglas que aún no han podido ser probadas que sean computacionalmente universales. **Con la propuesta de solución se entregan herramientas para investigar y experimentar con cualquier regla**, y así describir sus estructuras que no pueden ser fácilmente calculadas por inspección. Además, la herramienta permite agrupar todas las opciones para estudiar autómatas celulares de una dimensión en un solo centro de *software*, lo cual facilita al investigador en sus experimentos, ya que no tiene que gastar su tiempo creando la herramienta que necesita para sus propósitos.

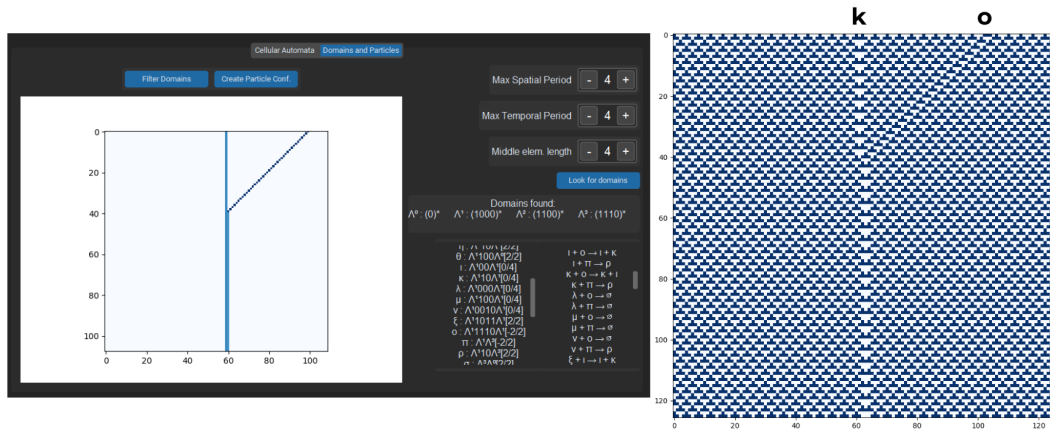


Figura 28: El resultado de la presente colisión es predicha correctamente por la herramienta.

Fuente: Elaboración Propia

Las principales contribuciones y aplicaciones de la herramienta creada van destinados hacia el mejoramiento del estudio y análisis de los autómatas celulares, permitiendo crear un puente que lo une con el estudio de la simulación de partículas en 1D. Con esto, se espera que sea más fácil que pueda usarse la dinámica de partículas y así demostrar que existe una comunicación global producto de estas interacciones entre las partículas y así intentar encontrar reglas que tengan un modelo de computo equivalente a la máquina de Turing. Por último, una de las mayores contribuciones que se esperan de una herramienta como la creada en el desarrollo de esta memoria, es la oportunidad de ser usada por la mayor cantidad de interesados posibles. Esto es particularmente cierto en el mundo de los autómatas celulares, ya que de acuerdo al físico Stephen Wolfram, en el futuro esta área de estudio será la base fundamental por la que se partirá la enseñanza a los estudiantes más jóvenes, incluso anterior al álgebra [Schiff, 2008].

Finalmente, se dedica un espacio en este último capítulo para motivar futuras líneas de trabajo a quienes se sientan interesados por el tema. Estos trabajos futuros pueden profundizar y mejorar lo visto en esta memoria, o más bien ampliar hacia otras áreas de estudio dentro de los autómatas celulares. En el primer caso, **se recomienda proponer alternativas a los algoritmos** que se han expuesto en esta memoria. Estas nuevas propuestas deberían apuntar a reducir la complejidad temporal, mediante usos de otras heurísticas, *threads*, o simple iteración sobre lo que ya existe. Alternativamente, **se recomienda profundizar en las funcionalidades de la herramienta**, tomando en cuenta la fase y el tamaño de las partículas para describir sus interacciones de forma más detallada, o encontrar otros resultados interesantes en reglas de autómatas celulares, relacionados a interacciones de las partículas encontradas que podrían simular circuitos lógicos, o encontrar, usando la herramienta, soluciones de menores estados para el problema de sincronización. En el segundo caso, relacionado a ampliar el trabajo de esta memoria hacia otras áreas de los autómatas celulares, se recomienda investigar la posibilidad de traducir los conceptos vistos en la presente memoria

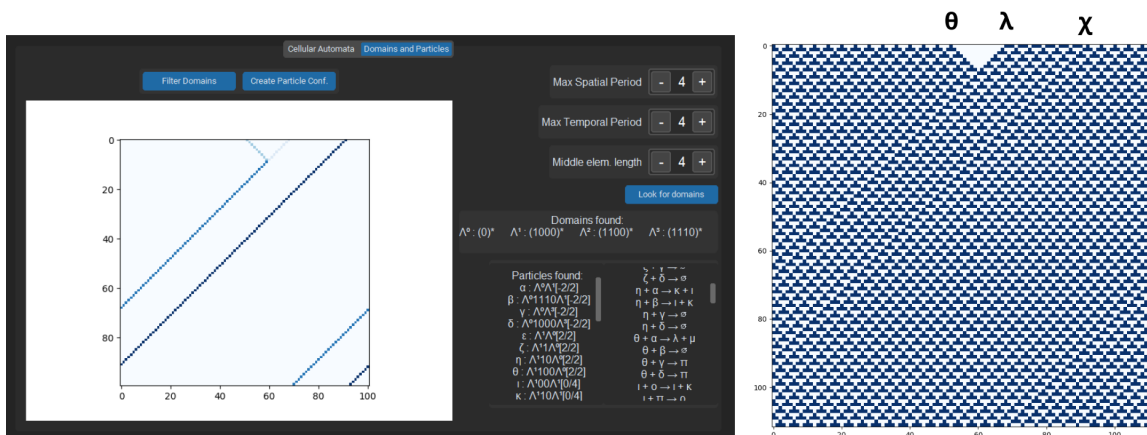


Figura 29: El resultado de la presente colisión es predicha correctamente por la herramienta.

Fuente: Elaboración Propia

hacia autómatas celulares de dos dimensiones. Esta línea de trabajo se enfocaría en encontrar maneras, en caso de que se pueda y tenga algún sentido en el área de investigación, de describir dominios y partículas en este tipo de autómatas celulares, y de manera automática, con el uso de algoritmos, similar a lo visto en este trabajo. Esta línea de trabajo podría ser en igual medida útil, interesante, pero también desafiante, puesto que efectivamente se está agregando una dimensión espacial más al problema.

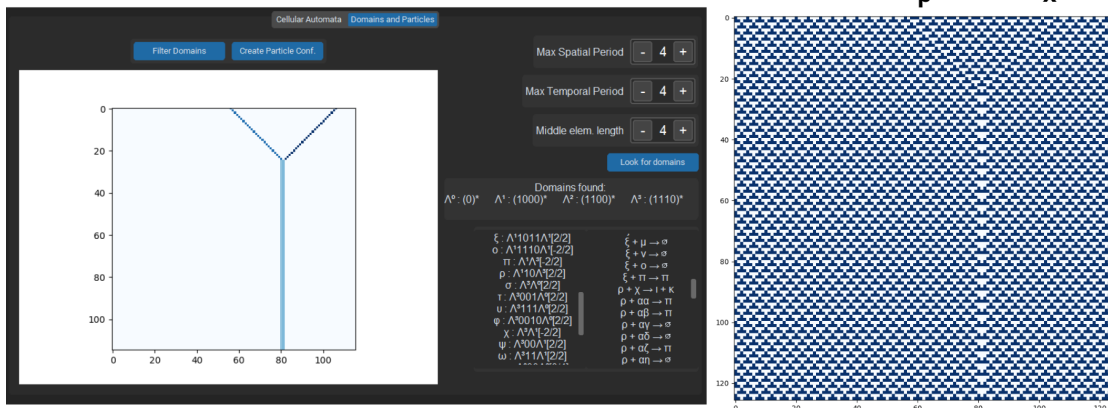


Figura 30: El resultado de la presente colisión es predicha correctamente por la herramienta.

Fuente: Elaboración Propia

REFERENCIAS BIBLIOGRÁFICAS

- [Axelrod y Hamilton, 1981] Axelrod, R. y Hamilton, W. D. (1981). The evolution of cooperation. *Science*, 211(4489):1390–1396.
- [Cook, 2004] Cook, M. (2004). Universality in elementary cellular automata.
- [Cook, 2009] Cook, M. (2009). A concrete view of rule 110 computation. En Neary, T., Woods, D., Seda, T., y Murphy, N., editores, *Proceedings International Workshop on The Complexity of Simple Programs*, Cork, Ireland, 6-7th December 2008, volumen 1 de *Electronic Proceedings in Theoretical Computer Science*, pp. 31–55. Open Publishing Association.
- [Das, 1998] Das, R. (1998). Emergent computation in cellular automata.
- [Deutsch y Dormann, 2005] Deutsch y Dormann (2005). *Cellular Automaton Modeling of Biological Pattern Formation*. Birkhäuser Boston.
- [Dewdney, 1984] Dewdney, A. K. (1984). Sharks and fish wage an ecological war on the toroidal planet wa-tor. *Scientific American*, 251(6):14–22.
- [Goles et al., 2011] Goles, E., Meunier, P-E., Rapaport, I., y Theyssier, G. (2011). Communication complexity and intrinsic universality in cellular automata. *Theoretical Computer Science*, 412(1):2–21. Complexity of Simple Programs.
- [Hanson y Crutchfield, 1997] Hanson, J. E. y Crutchfield, J. P. (1997). Computational mechanics of cellular automata: An example. *Physica D: Nonlinear Phenomena*, 103(1-4):169–189.

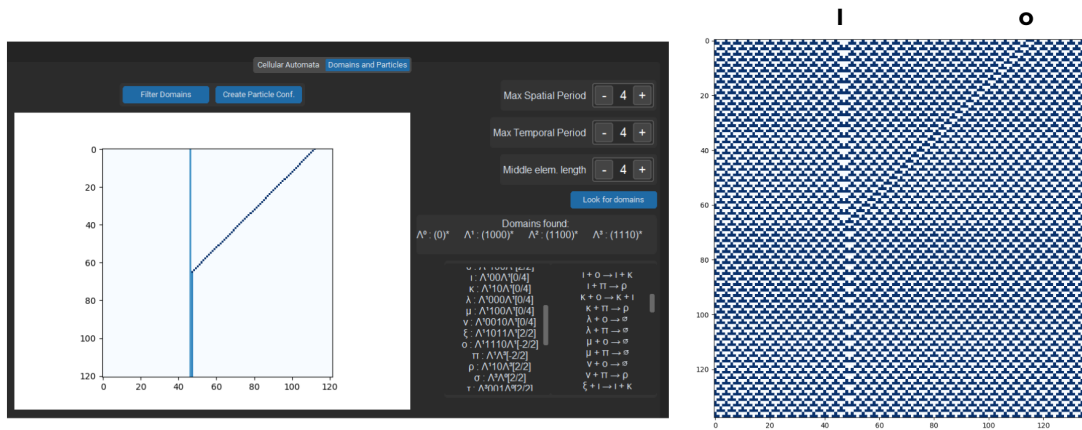


Figura 31: El resultado de la presente colisión es predicha correctamente por la herramienta.

Fuente: Elaboración Propia

[Mazoyer, 1987] Mazoyer, J. (1987). A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2):183–238.

[Mitchell et al., 1996] Mitchell, Melanie and Crutchfield, James P and Das, Rajarshi and others (1996). Evolving cellular automata with genetic algorithms: A review of recent work. En *Proceedings of the First international conference on evolutionary computation and its applications (EvCA'96)*, volumen 8. Moscow.

[Packard, 1985] Packard, N. H. (1985). Lattice models for solidification and aggregation. En *Proc. First International Symposium for Science on Form, Tsukuba University (Japan), Dec. 1985*, pp. 95–101.

[Pivato, 2007] Pivato, M. (2007). Defect particle kinematics in one-dimensional cellular automata. *Theoretical Computer Science*, 377(1):205–228.

[Schiff, 2008] Schiff, J. L. (2008). *Cellular Automata: A Discrete View of the World*. John Wiley & Sons, Hoboken, NJ.

[von Neumann, 1966] von Neumann, J. (1966). Theory of self-reproducing automata. *Edited by Arthur W. Burks*.

[Wolfram, 1984] Wolfram, S. (1984). Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):1–35.

[Wolfram, 2002] Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media, Champaign, IL.

