UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

TESIS DOCTORAL:

**COMPUTACIÓN CELULAR CON
MEMBRANAS: UN ESTUDIO DE
MODELOS DE CÓMPUTO BIOINSPIRADOS**

Autor: David Pérez Pérez

Ingeniero Superior en Informática por la Universidad Pontificia
de Salamanca

Director: Alfonso Rodríguez-Patón Aradas

Doctor en Informática por la Universidad Politécnica de Madrid

A Emma

Agradecimientos

Tantos años invertidos en la elaboración de esta tesis dan para enumerar a un buen número de personas que me han ayudado y apoyado durante todo este tiempo. No ha sido un trabajo sencillo, con momentos en los que más de una vez la palabra abandonar estuvo en mi mente más que presente.

Mi más sincero agradecimiento a Alfonso, director de esta tesis, por haberme aceptado como alumno y haber confiado en mí para en general formarme como investigador en el Laboratorio de Inteligencia Artificial y en particular para la elaboración de esta tesis. Agradezco también la ayuda de Petr, sin cuyas explicaciones, colaboración, consejos y ayuda, la palabra abandonar hubiese ganado.

Agradezco a Chema y Pepe sus ánimos constantes para que esta tesis fuese una realidad, haciéndome ver que era más sencillo de lo que parecía, y sí, era cierto, teníais razón.

También me gustaría agradecer a Daniel y Juan que me aceptasen, recién llegado de otra universidad, a formar parte del por aquel entonces llamado Laboratorio de Redes de Neuronas. Por supuesto, no me olvido tampoco, de todos los compañeros y amigos con los que he compartido laboratorio e incontables horas de trabajo. Gracias por escucharme.

Todo esto nunca hubiera sido posible sin el amparo incondicional de mi familia, mis padres y abuelos, sé que han hecho un esfuerzo tremendo apoyándome. Gracias Elena, por tu comprensión y ayuda.

Índice general

Resumen	1
Abstract	3
Aportaciones de esta Tesis	5
Organización y estructura de la memoria	7
 I Estado de la Cuestión	 3
 1. Sistemas P	 5
1.1. Introducción a los sistemas P	6
1.2. Biología de los sistemas P	8
1.2.1. La célula	9
1.2.2. La neurona	10
1.2.3. La membrana celular	12
1.3. Sistemas P de Objetos	13
1.4. Formalización de los sistemas P	14
1.4.1. Funcionamiento	17
1.5. Ejemplo	19
1.6. Extensiones y variantes de los sistemas P	21

1.6.1.	Disolución de la membrana	22
1.6.2.	Prioridades entre las reglas de evolución	25
1.6.3.	Sistemas P con membranas activas	25
1.6.4.	Sistemas P con cadenas	26
1.6.5.	Sistemas P con proteínas en las membranas	27
1.6.6.	Sistemas P de tejidos y neuronas	30
2.	Sistemas P basados en Neuronas de Impulsos	33
2.1.	Introducción	33
2.2.	Sistemas P de neuronas: definición y ejemplo	36
2.2.1.	Reglas de disparo	41
2.2.2.	Reglas de olvido	41
2.2.3.	Definición de Computación	42
3.	Planteamiento del Problema	45
II	Soluciones Propuestas	49
4.	Nuevas Formas Normales de los SN P	51
4.1.	Definiciones previas	51
4.2.	Trabajos previos	54
4.3.	Formas Normales Fuertes	59
4.4.	FN sin retrasos y con expresiones regulares simples	60
4.5.	FN con expresiones regulares simples	70
4.6.	FN sin retrasos y sin reglas de olvido	74
5.	Eficiencia de los sistemas P con proteínas	83
5.1.	Introducción a los sistemas P con proteínas	83
5.2.	Familias de sistemas P	90

5.3. Resolución del problema QSAT en tiempo lineal	93
5.4. Simulación de sistemas P con proteínas en ESPACIO-P	100
III Conclusiones y Líneas Futuras de Investigación	103
6. Conclusiones	105
7. Líneas Futuras	109
7.1. Aplicación futura de los sistemas SN P	111

Índice de figuras

1.1. Diagrama de un sistema P básico	14
1.2. Diagrama de Venn del sistema en su configuracion inicial . . .	20
1.3. Cronograma de ejecución para el sistema	21
2.1. Ejemplo básico de SNP	43
4.1. Modulo Π_{d_1} neurona con retraso 1	62
4.2. Módulo Π_{d_2} neurona con retraso 2	62
4.3. Máquina de registros con circulación dinámica de <i>spikes</i> y que no usa demora/retrasos	64
4.4. Módulo que simula $l_i : (ADD(r), l_j, l_k)$	65
4.5. Módulo que simula $l_i : (SUB(r), l_j, l_k)$	67
4.6. Módulo FIN	69
4.7. Módulo FIN con expresiones regulares simplificadas y condi- ción fuerte de parada	71
4.8. Módulo <i>INPUT</i> con expresiones regulares simples	73
4.9. Módulo <i>ADD</i> que simula la instrucción $l_i : (ADD(r), l_j, l_k)$. .	76
4.10. Módulo <i>SUB</i> que simula la instrucción $l_i : (SUB(r), l_j, l_k)$ junto con el registro r	78
4.11. Módulo <i>AND</i>	79
4.12. Módulo <i>FIN</i>	80

5.1. Un ejemplo de sistema P con proteínas en las membranas . . .	90
5.2. Expansión de la membrana inicial a un árbol binario.	96

Resumen

Los resultados presentados en la memoria de esta tesis doctoral se enmarcan en la denominada *computación celular con membranas* una nueva rama de investigación dentro de la computación natural creada por Gh. Paun en 1998, de ahí que habitualmente reciba el nombre de *sistemas P*. Este nuevo modelo de cómputo distribuido está inspirado en la estructura y funcionamiento de la célula.

El objetivo de esta tesis ha sido analizar el poder y la eficiencia computacional de estos sistemas de computación celular. En concreto, se han analizado dos tipos de sistemas P: por un lado los sistemas P de neuronas de impulsos, y por otro los sistemas P con proteínas en las membranas. Para el primer tipo, los resultados obtenidos demuestran que es posible que estos sistemas mantengan su universalidad aunque muchas de sus características se limiten o incluso se eliminen. Para el segundo tipo, se analiza la eficiencia computacional y se demuestra que son capaces de resolver problemas de la clase de complejidad ESPACIO-P (**PSPACE**) en tiempo polinómico.

ANÁLISIS DEL PODER COMPUTACIONAL: Los sistemas P de neuronas de impulsos (en adelante SN P, acrónimo procedente del inglés «Spiking Neural P Systems») son sistemas inspirados en el funcionamiento neuronal y en la forma en la que los impulsos se propagan por las redes sinápticas. Los SN P bio-inpirados poseen un numeroso abanico de características que ha-

cen que dichos sistemas sean universales y por tanto equivalentes, en poder computacional, a una máquina de Turing. Estos sistemas son potentes a nivel computacional, pero tal y como se definen incorporan numerosas características, quizás demasiadas. En (Ibarra et al. 2007) se demostró que en estos sistemas sus funcionalidades podrían ser limitadas sin comprometer su universalidad. Los resultados presentados en esta memoria son continuistas con la línea de trabajo de (Ibarra et al. 2007) y aportan nuevas *formas normales*. Esto es, nuevas variantes simplificadas de los sistemas SN P con un conjunto mínimo de funcionalidades pero que mantienen su poder computacional universal.

ANÁLISIS DE LA EFICIENCIA COMPUTACIONAL: En esta tesis se ha estudiado la eficiencia computacional de los denominados sistemas P con proteínas en las membranas. Se muestra que este modelo de cómputo es equivalente a las máquinas de acceso aleatorio paralelas (PRAM) o a las máquinas de Turing alterantes ya que se demuestra que un sistema P con proteínas, es capaz de resolver un problema ESPACIOP-Completo como el QSAT (problema de satisfacibilidad de fórmulas lógicas cuantificado) en tiempo polinómico. Esta variante de sistemas P con proteínas es muy eficiente gracias al poder de las proteínas a la hora de catalizar los procesos de comunicación intercelulares.

Abstract

The results presented at this thesis belong to *membrane computing* a new research branch inside of Natural computing. This new branch was created by Gh. Paun on 1998, hence usually receives the name of P Systems. This new distributed computing model is inspired on structure and functioning of cell.

The aim of this thesis is to analyze the efficiency and computational power of these computational cellular systems. Specifically there have been analyzed two different classes of P systems. On the one hand it has been analyzed the Neural Spiking P Systems, and on the other hand it has been analyzed the P systems with proteins on membranes. For the first class it is shown that it is possible to reduce or restrict the characteristics of these kind of systems without loss of computational power. For the second class it is analyzed the computational efficiency solving on polynomial time **PSACE** problems.

COMPUTATIONAL POWER ANALYSIS: The spiking neural P systems (SN P in short) are systems inspired by the way of neural cells operate sending spikes through the synaptic networks. The bio-inspired SN Ps possess a large range of features that make these systems to be universal and therefore equivalent in computational power to a Turing machine. Such systems are computationally powerful, but by definition they incorporate a lot of features, perhaps too much. In (Ibarra et al. in 2007) it was shown that their

functionality may be limited without compromising its universality. The results presented herein continue the (Ibarra et al. 2007) line of work providing new *formal forms*. That is, new SN P simplified variants with a minimum set of functionalities but keeping the universal computational power.

COMPUTATIONAL EFFICIENCY ANALYSIS: In this thesis we study the computational efficiency of P systems with proteins on membranes. We show that this computational model is equivalent to parallel random access machine (PRAM) or alternating Turing machine because, we show P Systems with proteins can solve a PSPACE-Complete problem as QSAT (Quantified Propositional Satisfiability Problem) on polynomial time. This variant of P Systems with proteins is very efficient thanks to computational power of proteins to catalyze inter-cellular communication processes.

Aportaciones de esta Tesis

A continuación se enumeran las cuatro aportaciones científicas de esta Tesis Doctoral en el área de la Computación Natural:

1. Marc García-Arnau, David Pérez, Alfonso Rodríguez-Patón, Petr Sosík. Spiking Neural P Systems: Stronger Normal Forms. *International Journal of Unconventional Computing* Volume 5(5): 411-425 (2009)
2. García-Arnau, M., Pérez, D., Rodríguez-Patón, A., & Sosík, P. (2008). On the power of elementary features in spiking neural P systems. *Natural Computing*, Volume 7(4), 471-483.
3. Sosík, P., Păun, A., Rodríguez-Patón, A., & Pérez, D. (2010). On the power of computing with proteins on membranes. In *Membrane Computing. Lecture Notes In Computer Science* (pp. 448-460). Springer Berlin Heidelberg.
4. García-Arnau, M., Pérez, D., Rodríguez-Patón, A., & Sosík, P. (2007). Spiking neural P systems: Stronger normal forms. *Fifth Brainstorming Week on Membrane Computing, RGCN Report*, 1, 157-178.

Organización y estructura de la memoria

La tesis doctoral que se presenta en esta memoria se divide en tres partes: una primera con título *Estado de la cuestión y planteamiento del problema*, una segunda con título *Soluciones propuestas* y una última parte que recoge las *Conclusiones y Líneas futuras de investigación*.

La parte *Estado de la cuestión y Planteamiento del problema*, está dividida en tres capítulos. En el primero de ellos, capítulo 1, se hace una revisión del estado del arte de los sistemas P. En el segundo, capítulo 2, se estudia una variante de los sistemas P, los sistemas P basados en neuronas de impulsos. Ambos capítulos familiarizarán al lector con los conceptos básicos de ambas clases de sistemas P. Por último y en esta misma parte, en el capítulo 3, se expone el planteamiento del problema al que esta tesis pretende dar respuesta.

La parte *Soluciones propuestas* está dividida dos capítulos. En el capítulo 4 se presentan los resultados logrados tras el análisis del poder computacional de los sistemas P basados en neuronas de impulsos a los que se les ha limitado alguna de sus funcionalidades. Además se incluyen las Nuevas Formas Normales descubiertas. En el capítulo 5, se presentan los resultados logrados tras resolver el problema QSAT en tiempo polinómico, un problema perte-

neciente a la clase ESPACIOP, usando para ello un sistema P con proteínas en las membranas.

Por último, la parte *Conclusiones y Líneas futuras de investigación*, incluye dos capítulos en los que se detallan las conclusiones y líneas futuras de trabajo derivadas de esta tesis doctoral.

Parte I

Estado de la Cuestión

Capítulo 1

Sistemas celulares de computación con membranas (Sistemas P)

En este capítulo, a lo largo de diferentes secciones se describe de manera abreviada el estado del arte de los sistemas de computación celular con membranas, también llamados sistemas P. En la sección 1.2 se incluye una breve introducción a los conceptos biológicos básicos que han servido como motivación para la creación de estos sistemas. En las secciones: 1.3, 1.4 y 1.5 se describen las características fundamentales, su formalización y un ejemplo de funcionamiento. Por último, en la sección 1.6 se detallan algunas de las variantes más representativas del modelo inicial de sistemas P, haciendo especial hincapié en la sección 1.6.5, a la variante que incluye proteínas en el interior de las membranas. Como se verá en el capítulo 5, los sistemas P con proteínas en las membranas son computacionalmente eficientes resolviendo problemas pertenecientes a la clase *ESPACIOP*.

1.1. Introducción a los Sistemas Celulares de Computación con Membranas

Los sistemas celulares de computación con membranas son una rama más de la computación natural [KR08]. Su propósito es abstraer modelos computacionales, denominados sistemas de membranas (Membrane Systems o simplemente sistemas P), a partir de la observación de la estructura y funcionamiento de elementos tales como las células, los tejidos celulares, los órganos o una población de células.

Las primeras investigaciones en esta disciplina se inician en el año 1998 [Pău98])y tienen una rápida aceptación en la comunidad científica. Tanto es así, que en el año 2003 el Instituto Thomson for Scientific Information declara que la computación con membranas constituye un área de investigación emergente en el mundo de las ciencias de la computación.

En la actualidad existen numerosas publicaciones científicas en este ámbito, varias tesis, libros y volúmenes especiales, todos ellos enfocados a describir en detalle esta novedosa área de investigación y sentar una sólida base de conocimiento útil para todo científico que pretenda introducirse en el área. Asimismo se ha puesto a disposición de la comunidad una web [Sit15] que sirve de punto de acceso y punto en común para todos los investigadores de esta nueva disciplina.

Los sistemas de membranas reciben el sobrenombre de sistemas P (P Systems en inglés). Posiblemente deban su sobrenombre a la primera inicial del apellido de su creador, George Paun. En la actualidad estos sistemas P han evolucionado y existen muchos tipos y variantes. Algunos de ellos tienen motivación matemática, otros biológica y otros en cambio informática.

El origen de la computación con membranas tiene lugar durante la bús-

queda de un modelo matemático inspirado en el funcionamiento y estructura de una célula de un ser vivo o, mejor dicho, inspirado en el papel que juegan las membranas dentro de dicha célula. Las membranas celulares como se sabe, separan o compartimentalizan las funciones metabólicas aislando cada uno de los componentes biológicos que intervienen durante todo el ciclo de vida celular.

De forma resumida, los primeros diseños de sistemas P poseían las siguientes características: disposición jerárquica de membranas, agrupación de objetos en multiconjuntos y evolución de los objetos de acuerdo a reglas que actúan sobre ellos. De acuerdo a estas características es sencillo ver el paralelismo del modelo de computación y una célula de un ser vivo. Podemos imaginar que una célula es una gran membrana que encierra en su interior funciones, y además esas funciones están restringidas a la membrana que las contiene. En una célula, el núcleo, las mitocondrias o las vacuolas delimitan en definitiva cierta función celular. Por tanto, existe una cierta jerarquía de membranas que delimitan regiones en las que se realizan diversas operaciones sobre los elementos que encierran.

Pero el modelo anteriormente descrito se ha ido completando a medida que han ido avanzando las investigaciones en el área. Estos avances han dado lugar a que en la actualidad existan nuevos tipos de sistemas y que además también se hayan incorporado nuevas características al sistema inicial. Ejemplos de ello son los sistemas basados en tejidos, propuesto por Martín-Vide *et al.* [MVPPRP03], en los que las membranas ya no se disponen jerárquicamente sino que su disposición está basada en un grafo, donde los nodos corresponderían a membranas y los vértices a canales de comunicación entre membranas.

Uno de los últimos aportes ha sido la incorporación de un nuevo tipo de

sistemas basados en células neuronales cuya motivación principal surge tras el estudio de las redes de neuronas de impulsos (Spiking Neural Networks) [MB99] [Maa02]. Este nuevo tipo de sistemas es conocido como sistemas P basados en neuronas de impulsos; nos referiremos a ellos lo largo de esta monografía como sistemas SN P, sobrenombre derivado de sus siglas en inglés (Spiking Neural P Systems). Esta nueva variante fue propuesta por Ionescu *et al.* en [IPY06].

1.2. Consideraciones biológicas básicas sobre los sistemas P

Las nociones biológicas que se presentan en este apartado sustentan los fundamentos biológicos del origen de los sistemas P. Los sistemas P usan como fuente de inspiración conceptos biológicos a diferentes niveles de organización celular:

- A nivel subcelular, donde se incluyen la estructura y componentes que forman la célula.
- A nivel celular, nivel centrado en los procesos celulares y el funcionamiento de la propia célula.
- A nivel supracelular, nivel que estudia como operan organismos multicelulares o poblaciones de células.

En la sección 1.2.1 se describen tanto a nivel subcelular como celular dos tipos diferentes de células que forman parte de todos los organismos vivos, las células procariotas y las eucariotas. También se estudia un tipo especial de células eucariotas, las neuronas. Por otra parte, en la sección

1.2.3 se estudian varios aspectos de la composición y funcionamiento de las membranas celulares, elemento esencial en los sistemas P.

Para un detalle más avanzado de los temas biológicos tratados en este apartado, pueden ser consultadas diferentes monografías [BAL⁺14], [LBK⁺07], [Coo00] que tratan el detalle de la célula y los procesos celulares. A modo de introducción también puede ser consultado el capítulo 2 del HandBook of Membrane Computing [PRS10].

1.2.1. La célula

Esta sección está dedicada al estudio de las características fundamentales de las células, tanto eucariotas como procariotas, así como un tipo específico de célula procariota, la neurona (ver sección 1.2.2).

Todos los organismos vivos ya sean unicelulares o pluricelulares están compuestos del mismo tipo de moléculas, obviamente ensambladas de forma distinta en cada uno de ellos, pero que comparten cierta similitud en su estructura y organización.

En relación a su composición, la mayor parte de los componentes bioquímicos que la célula importa de su entorno extracelular son moléculas orgánicas como el azúcar y las vitaminas. También importa iones inorgánicos como el calcio, el sodio y el potasio y sobre todo agua. En cambio los polímeros, moléculas de mayor tamaño, como los lípidos, las proteínas, los polisacáridos y los ácidos nucleicos son menos abundantes y por regla general serán sintetizados en el interior de la célula. Las proteínas son variadas y llevan a cabo numerosas funciones dentro de la célula. Las proteínas están formadas por uno o varios polipéptidos que se pliegan formando una estructura tridimensional que les proporcionan la funcionalidad. Cada uno de los polipéptidos están formados a su vez por unidades más pequeñas, los aminoácidos. Se-

cuencias concretas de aminoácidos determinan en la cadena polipéptida a los genes. Los genes están compuestos por ácido desoxirribonucleico (ADN) y por medio de ellos la célula dirige la síntesis de las proteínas, garantizando el correcto funcionamiento de los procesos celulares.

Por otro lado, en relación a la estructura organizacional de la célula distinguimos dos tipos, las células eucariotas y las células procariotas. Su principal diferencia es que en las células eucariotas existe un verdadero núcleo que protege el material genético de la célula, en cambio en las procariotas el material genético no está aislado por ninguna membrana. Pero a pesar de esa diferencia estructural interna, ambos tipos sí se aíslan del exterior por medio de una membrana celular. Esta membrana funciona como una barrera selectiva, que dejar pasar al interior determinados componentes para que los procesos celulares tengan lugar. Esta membrana gobierna un gran número de funciones como son: el movimiento bidireccional de elementos químicos entre la célula y el entorno, el control de entrada de nutrientes y salida de residuos de deshecho, el control de la diferencia de potencial entre el pH interno y externo de la célula, la trasducción de señales hacia la célula, la respuesta antigénica contra materiales externos o el paso de metabolitos entre las células externas.

1.2.2. La neurona

Una neurona [LBK⁺07] es una célula eucariota con cuatro partes funcionales distintas: el cuerpo, el axón, las sinapsis y las dendritas. Cada una de las partes desarrolla diferentes funciones en la célula. Esta forma de ver las neuronas y que difiere notablemente del resto de células debe agradecerse en parte a los trabajos llevados a cabo por Santiago Ramón y Cajal [RyC99] a finales del siglo XIX.

Existen diversos tipos de células neuronales y cada una presenta caracte-

rísticas morfológicas distintas dependiendo de la funcionalidad que desempeñan. Por ejemplo, las encargadas del sistema motor poseen un largo y único axón normalmente aislado por una vaina de mielina. Las neuronas sensoriales, que son encargadas de transmitir los impulsos sensoriales de las células receptoras, en cambio poseen un axón de doble rama que se divide al final del cuerpo de la neurona. Las neuronas multipolares que conectan las neuronas sensoriales entre sí poseen un conjunto denso de ramas dendríticas y un único axón.

Por otro lado, el cuerpo celular de la neurona contiene el núcleo y otros orgánulo, lugar en el que son sintetizadas las proteínas. El axón constituye una prolongación del cuerpo celular y su función es conectar cada neurona a un circuito de muchas neuronas por medio de sus terminaciones, las sinapsis. El axón está especializado en la rápida transferencia de impulsos eléctricos de manera unidireccional dentro de la célula, concretamente desde el cuerpo hacia las sinapsis. A este hecho también se le conoce con el nombre de acción potencial. Los impulsos eléctricos son originados a partir del ruido eléctrico producido en las dendritas, si el ruido eléctrico es lo suficientemente fuerte se disparará una señal.

Una vez que la señal eléctrica es conducida a través del axón alcanza las sinapsis provocando un aumento de la concentración de calcio, que como consecuencia libera neurotransmisores. Un neurotransmisor es una solución química alojada en una pequeña vesícula intracelular que se fusiona con la membrana sináptica mediante exocitosis. Tras la fusión, el neurotransmisor se difunde en un espacio restringido entre la neurona excitada y la célula postsináptica; posteriormente se une a unos receptores específicos (inhibidores o excitadores) situados en la membrana postsináptica. Su unión dispara la apertura de canales iónicos en la célula postsináptica provocando un cambio

en su potencial eléctrico y dando lugar a un impulso eléctrico. La señal eléctrica que llega a través de una célula presináptica y se propaga a lo largo de su axón, en un primer momento es transformada en una señal química para posteriormente volver a ser cambiada en una señal eléctrica.

Si la célula postsináptica es una neurona, el neurotransmisor se recibe en las dendritas, que no son más que una extensión del cuerpo celular pero especializadas en la conducción del impulso eléctrico hacia la célula. Las dendritas están muy ramificadas sobre todo en aquellas células que forman parte del sistema nervioso central. Esta ramificación dendrítica hace posible una mayor interconexión con el resto de células presinápticas.

1.2.3. La membrana celular

La membrana celular, así como las diferentes membranas que rodean los orgánulos internos de las células eucariotas, están formadas por lípidos, proteínas e hidratos de carbono. El grado de abundancia de cada uno de estos componentes varía dependiendo del tipo de orgánulo y el tipo de célula. Las células procariotas poseen mayoritariamente en su plasma dos tipos de lípidos: fosfolípidos y glucolípidos. En cambio, en las eucariotas encontramos: fosfolípidos, glucolípidos, esfingolípidos y colesterol en su membrana exterior, y fosfolípidos fundamentalmente en las membranas internas que recubren los orgánulos.

Existen por su composición molecular cuatro clases de fosfolípidos. La mayoría son similares estructuralmente con una parte polar, que está eléctricamente cargada y es conocida como cabeza, y una parte apolar conocida comúnmente como cola. Los fosfolípidos poseen una parte hidrófoba y otra hidrofílica, y gracias a estas propiedades se pueden unir formando una bicapa con dos extremos hidrofílicos en la que las cabezas de los fosfolípidos pueden

interaccionar con las disoluciones acuosas. Las membranas están constituidas por esta estructura de doble capa de fosfolípidos que hace de barrera entre los dos compartimentos acuosos.

1.3. Sistemas de Membranas con multiconjuntos de Objetos

El modelo más básico y primero en ser estudiado ha sido el sistema de membranas con objetos encerrados en su interior. Estos sistemas están constituidos por una estructura de membranas, que encierran multiconjuntos de objetos y reglas de evolución.

La membrana puede ser vista como una región tridimensional en el espacio, que encierra en su interior vesículas; estas vesículas no serán más que nuevas membranas que a su vez pueden alojar a otras. La membrana más exterior a todas ellas recibe el nombre de *piel* (skin). Las membranas que no alojan en su interior nuevas membranas se denominan *membranas elementales* y las que sí alberguen en su interior a otras membranas, se las denomina *membranas no elementales*.

Cada membrana delimita a una *región* y esta región a su vez constituye una parte en los sistemas de membranas, existiendo una correspondencia directa entre región y membrana. Es habitual referirse a objetos de una región o membrana de manera indistinta. En todo momento se debe tener también presente que la noción de membrana es una idealización de la membrana biológica y por tanto simplificada. La membrana, en el modelo inicialmente propuesto, marca el límite de lo que está dentro y lo que se queda fuera, careciendo su interior de una consistencia o estructura específica.

Figura 1.1: Diagrama de un sistema P idealizado a partir de las membranas que componen una célula de un ser vivo ¹

1.4. Formas de representación y formalización

Una forma de representación de las características de los sistemas P descritas en el apartado anterior, es mediante el uso de diagramas de Venn, tal y como se representa en la figura 1.1. La parte derecha de la figura representa un diseño básico y simplificado de las membranas de una célula biológica y en la parte izquierda se incluye la idealización del sistema P. Para sistemas P sencillos la representación mediante diagramas de Venn resulta ilustrativa. Pero aunque estos diagramas son visualmente atractivos y próximos a la idealización de la equivalencia de un sistema P y una célula biológica, lo normal es formalizar estos sistemas utilizando un constructo matemático, similar al usado en la teoría de autómatas y de lenguajes formales. La formalización utilizada en los sistemas P se basa en la usada en la teoría de autómatas o las máquinas de Turing.

Otra posible alternativa a la representación mediante diagramas de Venn es utilizar cadenas de pares de corchetes $[_n]_n$. Cada par de corchetes representa una membrana y el subíndice n su etiqueta. Siguiendo esta alternativa

¹Imagen celular obtenida en <http://m.harunyahya.com/tr/works/45254/The-miracle-of-the-cell-membrane>

de representación, la figura 1.1 quedaría de la siguiente forma:

$$[1[2]2[3[5]5]3[4]4]1$$

Hasta el momento se ha detallado y formalizado un único elemento del sistema P, la *membrana*. Si bien la membrana es un elemento esencial y que además da nombre al sistema, no es el único. El siguiente ingrediente a representar es los *multiconjuntos de objetos*. Los multiconjuntos no son más que objetos encerrados dentro de una de las membranas y gobernados por las *reglas de evolución*. Las reglas de evolución constituyen un ingrediente más y, al igual que las cadenas de objetos, también están asociadas a una región y se agrupan en multiconjuntos. El multiconjunto es un conjunto de múltiples copias de un mismo objeto que puede aparecer en una región del sistema P.

Partiendo de las nociones biológicas previas, tanto a nivel celular como subcelular, el modelo de cómputo bioinspirado quedará definido de acuerdo a las siguientes características:

1. Una membrana, que sirve de elemento delimitador y al igual que la membrana biológica preserva los objetos en ella encerrados del efecto de otros elementos externos.
2. Los multiconjuntos de objetos, que se asemejan a moléculas como los polímeros, los lípidos, las proteínas, los polisacáridos y ácidos nucleicos. Es sencillo observar el paralelismo entre las moléculas anteriores y los multiconjuntos de los sistemas P. Los multiconjuntos de objetos son a los sistemas P el equivalente a las moléculas existentes en las regiones celulares.

3. Las reglas de evolución, que representan la funcionalidad desarrollada sobre los objetos del sistema P en una determinada región. Estas reglas de evolución se asemejan a las funciones metabólicas que la célula desarrolla en cada uno de los orgánulos internos que la constituyen.

Vistas las características del sistema P y una breve motivación biológica, a continuación se incluye un equivalente formal para este tipo básico de sistema P. Un sistema P se define como la tupla:

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_0) \quad (1.1)$$

donde:

1. O es un alfabeto que define elementos que denominaremos objetos.
2. μ define la estructura de membranas. Un sistema P podrá tener m membranas tal que $m \geq 1$. Por tanto, representa el número de regiones dentro del sistema P o a lo que también llamamos *grado de Π* .
3. $w_i, 1 \leq i \leq m$ son cadenas que representan multiconjuntos sobre O asociadas a una de las regiones definidas sobre μ
4. $R_i, 1 \leq i \leq m$ constituye el conjunto finito de reglas de evolución sobre O . R_i está asociada a la región i de μ . Una regla de evolución tiene la forma $u \rightarrow v$, donde u es una cadena sobre O y v una cadena sobre el conjunto O_{tar} , donde, $O_{tar} = O \times TAR$, para $TAR = \{here, out\} \cup \{in_j | 1 \leq j \leq m\}$.
5. $i_0 \in \{1, 2, \dots, m\}$ identifica las membranas de salida del sistema.

Al conjunto formado por los símbolos *here*, *out*, *in_j* se le denomina *comando objetivo* y al tamaño de u en la regla de evolución $u \rightarrow v$ se le denomina *radio de regla de evolución*. Como simplificación a la hora de representar el modelo el comando objetivo *here* es omitido.

Los sistemas con reglas que poseen un radio $|u| > 1$ diremos que son *sistemas cooperativos* y *no cooperativos* en caso contrario. Un caso particular de los *cooperativos* lo constituyen los sistemas *catalíticos*.

1.4.1. Funcionamiento

Todas las reglas definidas sobre R_i se aplican de forma síncrona en todas las regiones i pertenecientes a μ . Las reglas y objetos, a aplicar y evolucionar respectivamente, son seleccionadas de forma no determinista y en un grado de paralelismo máximo.

Para todo sistema P se supone un reloj común que gobierna el tiempo en todas las regiones i definidas en el sistema. A su vez, se define cada estado del sistema o configuración como el conjunto de todos los objetos de los multiconjuntos dispuestos en las regiones en un instante. Cada instante lo marca el reloj global del sistema P.

Análogamente a los autómatas los sistemas P también definen transiciones. Dada una configuración $C_1 = \{w_1, \dots, w_n\}$ y otra configuración $C_2 = \{w'_1, \dots, w'_n\}$, definimos transición como $C_1 \Rightarrow C_2$ sobre Π si hemos podido pasar de C_1 a C_2 usando para ello las reglas de evolución definidas en las regiones (R_i) .

Las reglas de evolución $u \rightarrow v$ anteriores se aplican de la siguiente forma: las ocurrencias de objetos pertenecientes al multiconjunto definido en una región que coincidan con la parte u de la regla, se sustituyen por los símbolos indicados en la parte izquierda v , pero teniendo en cuenta el *comando objetivo*

definido en la regla. Es decir: $TAR = \{here, out\} \cup \{in_j | 1 \leq j \leq m\}$.

- Si la regla posee *here*, los objetos del multiconjunto de la parte izquierda (u) de la regla, son sustituidos por los indicados en la parte derecha (v) permaneciendo en la misma región en la que se encuentra la regla.
- Si la regla posee in_i , los objetos del multiconjunto de la parte izquierda (u) de la regla son sustituidos por los indicados en la parte derecha (v), pero en este caso los objetos viajarán a través de la membrana para ir a parar a la región indicada en i . La membrana i deberá ser adyacente, de lo contrario la regla no será aplicable.
- Si la regla posee *out*, los objetos definidos en v abandonarán la región en la que se encuentran yendo a parar a la membrana inmediatamente superior.

En cada uno de los pasos del sistema P los objetos y reglas de evolución se eligen de forma no determinista y manteniendo el carácter masivamente paralelo. Reglas y objetos se aplican una vez en cada paso.

A continuación podemos ver un ejemplo de cómo es el funcionamiento de las reglas de evolución. Dado el multiconjunto de símbolos $aaddbb$ y las reglas de evolución $r_1 : aa \rightarrow d$, $r_2 : dd \rightarrow dbb$ y $r_3 : db \rightarrow d$ podrían darse los siguientes resultados tras aplicar alguna de la reglas:

- Si se eligen las r_1 y r_2 se obtendría como resultado el multiconjunto $ddbddd$
- Pero también podrían haber sido elegidas las reglas r_1 y dos veces r_3 dando como resultado el multiconjunto ddd

En definitiva, los objetos a evolucionar se eligen de forma no determinista buscando el máximo paralelismo en la aplicación de las reglas. Además, una regla podrá ser aplicada tantas veces como copias de un objeto existan en una región. En caso de que algún objeto no esté afectado por alguna de las reglas, éste permanecerá invariable en la región.

Vista la forma de aplicar las reglas y su paralelismo, existe otro concepto relevante en el funcionamiento de los sistemas P, el concepto de computación. Se considera computación a la secuencia de transiciones entre dos configuraciones definidas sobre Π ; además diremos que esta computación es exitosa si y solo si el sistema se para, es decir, no es posible aplicar más reglas sobre los objetos presentes en el sistema. El resultado o salida del sistema lo constituye el conjunto de objetos presente en la región de salida del sistema, una vez el sistema se pare. Cualquier objeto alojado en otra de las regiones que no sea la de salida no será tenido en cuenta. Es importante destacar que un sistema no producirá salida si no alcanza un estado de parada.

1.5. Ejemplo de funcionamiento

Una vez que se ha detallado el conjunto de reglas que siguen este tipo de sistemas para su funcionamiento, a continuación se detalla un ejemplo básico para familiarizar al lector con la forma en que un sistema de membranas opera.

Dado el siguiente sistema P:

$$\Pi_i = (O, \mu, w_1, w_2, R_1, R_2, i_0) \quad (1.2)$$

con los componentes:

$$O = \{a, b, c\},$$

$$\mu = [{}_1[{}_2]{}_2]{}_1,$$

$$w_1 = a^2$$

$$w_2 = \lambda$$

$$R_1 = \{a \rightarrow a(b, in_2)(c, in_2)^2, a^2 \rightarrow (a, out)^2\},$$

$$R_2 = \emptyset,$$

$$i_0 = 2$$

El sistema P quedará representando utilizando diagramas de Venn de acuerdo al diagrama de la figura 1.2:

Figura 1.2: Diagrama de Venn del sistema en su configuracion inicial

Si el sistema se pusiese en funcionamiento, teniendo en cuenta que se quedaría parado siempre que se ejecutara la regla R_2 , después de n ejecuciones la salida obtenida en la región de salida 2 sería $b^{2n}c^{4n}$. Como ejemplo se muestra en la figura 1.3 una sucesión de diagramas en los que se pueden ver las transiciones del sistema a medida que se aplican en cada iteración las

Figura 1.3: Cronograma de ejecución para el sistema

reglas R_1 y R_2 .

1.6. Extensiones al modelo básico

Tal y como se describe en el libro [Pău02] la mayor parte de sistemas P contruidos son computacionalmente completos o equivalentes a una máquina de Turing, sin embargo se han explorado nuevas extensiones al modelo con el fin de aumentar en algunos casos la eficiencia computacional. Un caso claro de ello son los sistemas P con proteínas en las membranas que, como se demuestra el capítulo 5, pueden resolver en tiempo polinómico problemas pertenecientes al **ESPACIOP**. Pero a parte de los sistemas P con proteínas, surgen nuevas extensiones al modelo básico, algunas de ellas con origen biológico y otras en cambio con motivación matemática. A lo largo de los siguientes apartados se destaca sin entrar en demasiado detalle algunas de ellas.

1.6.1. Disolución de la membrana

La disolución de membranas constituye un añadido simple al modelo básico visto en el apartado 1.4. El nuevo ingrediente hace posible disolver la membrana que encierra los objetos. Para modelar esta nueva funcionalidad se añade el símbolo δ en la parte derecha de la regla, de este modo: $u \rightarrow v\delta$. El funcionamiento de la regla es similar al que se ha visto anteriormente; todo símbolo de u se cambia por los indicados en la parte v de la regla y una vez transformados la membrana será disuelta. Por tanto, todo el conjunto de objetos o membranas que contenga la membrana a ser disuelta pasan a formar parte de la membrana inmediatamente superior. Finalmente, las reglas que en ella estén contenidas desaparecen.

Hay que destacar que la membrana exterior del sistema P no puede ser disuelta, es decir la regla $u \rightarrow v\delta$ no es aplicable cuando $m = 1$.

Con esta nueva regla que entra en juego, en la formalización del sistema P ya no es necesario identificar a una membrana elemental como la membrana de salida, puesto que una membrana que incluya otra en su interior empleando la regla $u \rightarrow v\delta$ puede transformar una membrana en elemental. Por tanto el sistema P queda definido como: $\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_0)$ donde el conjunto R_i podrá contener reglas del tipo $u \rightarrow v\delta$ e i_0 no tendrá que ser necesariamente una membrana elemental de μ .

Además la configuración de membranas no tendrá por qué ser la misma que al inicio, ya que tras sucesivas computaciones la configuración inicial de membranas se puede alterar. Pero en cualquier caso el concepto de computación permanece inalterable, es decir, para dos configuraciones $C_1 = (\mu', w'_{i_1}, \dots, w'_{m_k})$ y $C_2 = (\mu'', w''_{i_1}, \dots, w''_{m_k})$ definidas sobre Π , diremos que existe una transición de $C_1 \Rightarrow C_2$, si podemos pasar de la configuración C_1 a C_2 utilizando las reglas de evolución de $R_{i_1}, \dots, R_{i_k}$ incluidas en las

regiones $i_1, \dots, i_k$.

Obviamente si en alguna de las regiones existe una regla de tipo δ esta disolvería la membrana. Todos sus elementos pasarían a formar parte de la membrana inmediatamente superior y siempre acabarían perteneciendo a una membrana, puesto que la membrana que constituye la piel del sistema P no puede ser disuelta.

Se denota como $NOP_m(\alpha, tar, \delta)$ a la familia de conjuntos de números $N(\Pi)$ generados por los sistemas P tal que $m \geq 1$ e incluya reglas de disolución de membranas. De forma general todos los conjuntos de números obtenidos por computación de una clase de sistema P lo denotaremos como NOP , un subíndice que limita el número de membranas y a continuación entre paréntesis el tipo de reglas que emplea (ncoo, cat, coo u otro tipo) y su control de comunicación. En medio de los paréntesis también se pueden especificar otras características pero es importante destacar que sólo se incluirán aquellas que sean utilizadas por el sistema. Por tanto, si no especificamos una característica, esta no será utilizada por el sistema.

Si fuese necesario especificar un tipo de un sistema dado, se permitiría el uso de expresiones de la forma $\Pi = (ncoo, tar \neg \delta)$ indicando el tipo de reglas que son utilizadas y las que no, pero sólo como recurso para hacer énfasis en el hecho de aquellas reglas que son y no son utilizadas.

El uso de la regla de disolución de membranas que hemos detallado anteriormente incrementa la potencia de los sistemas P con objetos o símbolos en las membranas. Apoyándonos sobre la demostración seguida en [Pău02] se deduce:

Teorema 1.6.1 $NE0L \subseteq NOP_2(ncoo, tar, \delta)$

Demostración Dado un sistema $E0L$ $G = (V, T, w, R)$ construimos un sistema de grado 2

$$\Pi = (V \cup \{d\}, [1[2]_2]_1, 0, dw, R_1, R_2, 1)$$

con los conjuntos de reglas siguientes:

$$\begin{aligned} R_1 &= \{a \rightarrow a \mid a \in V - T\} \\ R_2 &= \{d \rightarrow d, d \rightarrow \delta \mid a \in V - T\} \end{aligned}$$

La membrana interna se utiliza para simular los pasos de derivación sobre G mientras haya símbolos d presentes en la membrana; cuando la membrana es disuelta, la simulación se para. Si algún símbolo no perteneciente a T sigue presente, entonces podemos utilizar infinitamente la regla $a \rightarrow a$ definida en R_1 . Por tanto, sólo las derivaciones terminales sobre G pueden ser simuladas mediante una interrupción de la computación en Π .

Es importante destacar que no es posible generar todos los conjuntos de lenguajes $E0L$ con sistemas de membranas con tan solo dos membranas y sin utilizar el truco de considerar como membrana de salida a la membrana más externa. Este resultado se deriva del Teorema 3.3.2 descrito en [Pău02]. Si un sistema P con dos membranas tal que la membrana de salida sea la interna del propio sistema, ya que ni la interna ni la piel pueden ser disueltas, el sistema no puede hacer uso de la regla de disolución de la membrana. Por tanto, este sistema solo puede generar algunos conjuntos de longitud de los lenguajes libres de contexto.

Para más detalles relacionados con la potencia computacional de este tipo de sistemas puede ser consultada la monografía [Pău02]. Podemos destacar que la inclusión de la regla de disolución de membranas y que define una variante del sistema P inicial, aporta mayor capacidad computacional pero no la suficiente como para obtener la completitud computacional o universalidad. Esto hace que sea necesario incluir nuevas características al sistema P original.

1.6.2. Prioridades entre las reglas de evolución

La universalidad se alcanza en los sistemas P que añaden esta nueva característica: incluir relaciones de prioridad en las reglas de cada región, además de la regla de disolución ya estudiada en el apartado anterior (ver 1.6.1) y el resto de componentes vistos anteriormente. Obtenemos pues la siguiente formalización para esta nueva variante:

$$\Pi_i = (O, \mu, w_1, \dots, w_m, (R_1, \rho_1), \dots, (R_m, \rho_m), i_0) \quad (1.3)$$

Obsérvese la inclusión de un nuevo elemento (ρ) para indicar la prioridad a la hora de definir la reglas para cada una de la regiones del sistema P.

1.6.3. Sistemas P con membranas activas

La característica estática de las membranas en los sistemas P puede verse como una debilidad desde el punto de vista biológico puesto que las membranas no evolucionan durante la computación del sistema P. Para ello se han considerado diversas modificaciones de la membrana, siendo una de ellas la disolución de la misma (ver apartado 1.6.1). Pero en este apartado veremos un proceso más complejo de evolución de membranas, los sistemas P con membranas activas.

En esta clase de sistemas P se establece una relación entre las membranas y las propias reglas de evolución lo que hace posible que las membranas puedan evolucionar durante las computaciones del sistema P.

Un tipo de regla incluida en este modelo es la división de membranas. Por ejemplo, dada la regla $[a]_h \rightarrow [b]_h[c]_h$ donde a, b, c son objetos y h la región, implicaría dividir la membrana h inducida por el objeto a ; seguidamente se generarían dos nuevas membranas: con la primera se sustituiría una copia de

a por el objeto b y en la segunda membrana generada se sustituiría el objeto a por c .

Mediante el uso de este tipo de reglas tanto el número de membranas como el número de objetos en una membrana puede crecer exponencialmente. Este hecho implicaría la existencia de membranas con etiquetas con el mismo nombre pero no causaría problemas en la aplicación de las reglas ya que mantienen la etiqueta sobre la membrana que son aplicadas.

Otro tipo de reglas son las que permiten al sistema P crear nuevas membranas. La regla es del tipo $a \rightarrow [b]_h$ donde a y b son objetos y h la etiqueta de la membrana. Al aplicar la regla se crea una nueva membrana elemental a partir de la existencia de un objeto a con una etiqueta h y encerrando en su interior un objeto b . A estas reglas también se las puede asignar a una región determinada, por ejemplo: $[a \rightarrow [b]_h]_g$. En este caso la regla $a \rightarrow [b]_h$ solo aplica a los objetos a existentes en la región g .

Existen más tipos de sistemas P que modelan otros comportamientos celulares en los cuales intervienen las membranas, como son la endocitosis o la exocitosis, aunque no se detallarán en esta monografía por quedar fuera del alcance. El resto de detalles de estas nuevas operaciones están descritos en el capítulo 11 de [PRS10]. Si el lector siente interés en conocer los detalles de la demostración de la universalidad, algo natural al definir una nueva clase de P sistema, podrá encontrarla en [Pă99].

1.6.4. Sistemas P con cadenas

Los sistemas P que hemos descrito hasta el momento utilizan objetos atómicos, es decir, cada uno de los objetos es independiente y el conjunto de objetos carece de estructura. Pero en el mundo de la biología parece lógico también considerar una variante como es el caso de las cadenas de objetos

pues en definitiva, en las regiones interiores de una célula, las proteínas, el ADN y el ARN, forman estructuras basadas en cadena.

La característica principal de esta clase de sistemas es que para definir los objetos se utilizan cadenas definidas sobre un alfabeto. Estas cadenas están presentes en el sistema al comienzo de la computación y se definirá el conjunto de reglas que operarán sobre estas cadenas y la forma en la que se obtendrán resultados al finalizar la computación. En esta clase de sistemas P, las cadenas pueden ser consideradas como un conjunto o un multiconjunto. Las reglas serán reglas de procesamiento de cadenas como por ejemplo reglas de reescritura paralela similares a los sistemas Lindenmayer [Lin68]. Los sistemas de Lindenmayer o L-Systems, como también son conocidos, fueron propuestos en 1968 por Aristid Lindenmayer. Constituyen un formalismo para simular el desarrollo de organismos multicelulares. Este formalismo toca de cerca los lenguajes formales y los autómatas lo que suscitó un fuerte interés por parte de los investigadores del área de la informática teórica.

Otra interesante idea para este tipo de sistemas y más relacionada con el mundo biológico es utilizar reglas específicas del procesamiento de cadenas de ADN, como la de corte-unión «splicing» o el insertado y eliminación. Para más detalles con este tipo de sistemas se pueden consultar los capítulos 7 y 8 de la monografía [PRS10].

1.6.5. Sistemas P con proteínas en las membranas

A continuación incluimos una nueva variante al sistema P inicial. Esta variante es estudiada por primera vez por Paun and Popa en [PP06b]. Esta nueva variante tiene un fundamento biológico, pues las proteínas tienen un papel fundamental en las membranas celulares de un ser vivo. Según Paun y Popa [PP06b] en biología muchas de las reacciones que tienen lugar en los

compartimentos de una célula están catalizados por una proteína embebida en la propia membrana. Es por eso que parece lógico estudiar la inclusión de esta nueva característica en los sistemas P. Se estima que en una célula animal, las proteínas constituyen alrededor de un 50% de la masa de las membranas; el resto está constituido por lípidos y pequeñas cantidades de carbohidratos.

Tal y como se extrae de [BAL⁺14], existen varios tipos de proteínas embebidas en las membranas de una célula: las proteínas integrales que pueden funcionar tanto en la parte interior como exterior de la membrana y las proteínas periféricas que únicamente pueden funcionar en una región de la célula. Con la intención de capturar esta característica biológica y enriquecer las características ya existentes en los sistemas P, Paun y Popa proponen la variante de sistemas P con proteínas en las membranas.

En los sistemas P con proteínas los objetos pueden estar en las regiones o en las membranas. El caso más restrictivo es el que considera que la información principal a ser procesada está codificada en multiconjuntos de las regiones del sistema P y esos objetos podrán evolucionar de acuerdo a un número limitado de proteínas alojadas en las membranas. Además, en las reglas de uso que son restrictivas, los objetos que puedan pasar las membranas estarán bajo el control de la proteína, cambiando o no los objetos o las proteínas durante la operación. De alguna forma es similar a una ampliación de las reglas de sinport/antiport que pueden ser consultadas en [AFR06] y [PP02].

En esta nueva clase de sistemas P se consideran dos tipos de objetos, las proteínas y los propios objetos que forman parte de los multiconjuntos. Las proteínas se colocan en la membrana y el resto de objetos dentro de la región delimitada por la membrana. Para identificar este hecho, para una proteína p

alojada en una membrana etiquetada con i escribiremos $[_ip]$. Tanto la región como la propia membrana pueden contener multiobjetos y proteínas.

La expresión, $[_1p_1^2p_2|a^3bd[_2p_3|b^5cd^2]_2]_1$ indica que tenemos una estructura de dos membranas, etiquetadas con 1 y 2. La membrana interna 2 contiene un multiconjunto de objetos b^5cd en su interior, además de una proteína p_3 . La membrana 1 por su parte posee los objetos a^3bd y el multiconjunto de proteínas $p_1^2p_2$.

A continuación se detallan los tipos de reglas usados por esta clase de sistemas P, la interacción entre proteínas y objetos. En todas ellas los objetos a, b, c, d son objetos y los elementos etiquetados como p son proteínas. Además distinguimos dos tipos de reglas, las restrictivas que se identifican como *res* y las de cambio de proteína que se identifican como *cp*.

Tipo	Regla	Efecto
1res	$[_ip a \rightarrow [_ip b$ $a[_ip]_i \rightarrow b[_ip]_i$	modifica un objeto, pero no lo mueve
2res	$[_ip a]_i \rightarrow a[_ip]_i$ $a[_ip]_i \rightarrow [_ip a]_i$	mueve un objeto sin modificarlo
3res	$[_ip a]_i \rightarrow b[_ip]_i$ $a[_ip]_i \rightarrow [_ip b]_i$	modifica y mueve un objeto
4res	$a[_ip b]_i \rightarrow b[_ip a]_i$	intercambia dos objetos
5res	$a[_ip b]_i \rightarrow c[_ip d]_i$	intercambia y modifica dos objetos

En todos los casos anteriores, las proteínas no cambian; podemos decir que tienen el papel de catalizadoras. En la tabla mostrada a continuación se presenta el segundo tipo de reglas que implican un cambio de proteínas.

Tipo	Regla	Efecto (además del cambio correspondiente de proteína)
1cp	$[ip a]_i \rightarrow [ip' b]_i$ $a[ip] \rightarrow b[ip']_i$	modifica un objeto, pero no lo mueve
2cp	$[ip a]_i \rightarrow a[ip']_i$ $a[ip] \rightarrow [ip' a]_i$	mueve un objeto sin modificarlo
3cp	$[ip a]_i \rightarrow b[ip']_i$ $a[ip] \rightarrow [ip' b]_i$	modifica y mueve un objeto
4cp	$a[ip b]_i \rightarrow b[ip' a]_i$	intercambia dos objetos
5cp	$a[ip b]_i \rightarrow c[ip' d]_i$	intercambia y modifica dos objetos

Para más detalles del funcionamiento de este tipo de sistemas P, que va más allá el propio alcance de esta memoria, pueden ser consultas las referencias [PP06b], [Sos14]

1.6.6. Sistemas P de tejidos y neuronas

Hasta el momento hemos detallado características de los sistemas P cuya estructura de las membranas equivale a una jerarquía. Una jerarquía considera la disposición de membranas de forma jerárquica asimilable a una estructura en forma de árbol. En esta forma arbórea el nodo padre o raíz lo constituye la membrana exterior y resto de membranas internas son los nodos hijos.

Pero fruto del estudio y la observación de las células surge otra posible variante en la que la disposición de las membranas ya no es jerárquica sino basada en un grafo. La disposición jerárquica destaca la importancia que tienen la disposición espacial de los orgánulos internos de una célula; en cambio la disposición en grafo destaca la importancia del intercambio de información (elementos químicos) que puede producirse en un momento dado. El grafo permite representar canales de comunicación existentes entre las diferentes membranas que componen la célula.

En los sistemas P basados en grafos distinguimos por ejemplo, las po-

blaciones de sistemas P en los que dinámicamente cambian sus enlaces de comunicación entre membranas durante sucesivas computaciones pudiendo el sistema P crear o destruir canales de comunicación. Otro ejemplo serían los sistemas P tipo Neurona; esta clase de sistemas P está inspirada en un tipo particular de células, las neuronas.

Los sistemas P tipo neurona, en lugar de células poseen neuronas y además incluyen un estado asociado a la neurona. Este estado controlará la evolución de los objetos. Ahora no tendremos reglas de comunicación como las que se han visto hasta el momento; las nuevas reglas de reescritura serán del tipo: $su \rightarrow s'v$ donde s, s' determinan estados y $u \rightarrow v$ es la regla tipo que se ha visto para el resto de clases de sistemas P, donde v puede incluir los indicadores para dirigir el movimiento de los objetos en las membranas. Pero ahora existe sinapsis, y un objeto será replicado a todas las membranas destino que posean sinapsis con la membrana origen.

En el área de la exploración neuronal de los sistemas P surge otra variante más: los sistemas P de neuronas de impulsos, en adelante SNPS. Estos SNPS sólo consideran un objeto posible, el *spike* o impulso eléctrico producido en la comunicación neuronal. Los impulsos eléctricos definidos en el SNPS son procesados mediante reglas del tipo $E/a^c \rightarrow a; d$. En la regla anterior la parte E constituye una expresión regular sobre el alfabeto $\{a\}$ y c, d son números naturales. Esta regla pretende modelar disparos en la neurona si se cumplen ciertas condiciones. Si la neurona contiene un número de *spikes* que cumpla la expresión regular definida en E entonces se consumirán c *spikes* y se producirá un nuevo spike. El *spike* se envía por las sinapsis definidas en el SNPS después de tener en cuenta el retraso definido en d . La característica del retraso es una abstracción de periodo refractario existente en el funcionamiento neuronal de un ser vivo. Quizás una de las características más innovadora

que proporciona esta clase de sistemas P es que no codifica información mediante los objetos encerrados en las membranas sino mediante las sucesiones de *spikes* producidas en el sistema utilizando el tiempo como soporte de la información.

Capítulo 2

Sistemas P basados en Neuronas de Impulsos

En este capítulo se hace un recorrido por el estado del arte de la variante de sistemas P basados en neuronas de impulsos, así como los trabajos previos existentes en materia de simplificación de esta nueva variante. La necesidad de simplificar el modelo surge debido a las numerosas características que se incluyeron en el modelo cuando éste fue concebido, que si bien pueden tener sentido biológico no aportan un mayor poder computacional.

2.1. Introducción

Sin duda las neuronas son un tipo de células que forman parte del cuerpo humano y son además de lo más interesante. A día de hoy todavía estamos lejos de la creación de un sistema que pueda llevar a cabo tareas similares a las que un conjunto de neuronas puede realizar. Las neuronas de nuestro cerebro colaboran para llevar a cabo tareas que la tecnología de la cual disponemos en la actualidad no es capaz de realizar; tareas como pensar, tomar

conciencia de una situación, intuir, etc. Pero no sólo esas tareas que a priori parecen complejas, sino que tareas como el reconocimiento de patrones y el reconocimiento del lenguaje que han sido estudiadas durante años en el área de la computación todavía son a día de hoy más fiables y rápidas a través de la mente humana que usando un ordenador.

El gran rendimiento que el cerebro alcanza procesando información, en parte radica en su característica inherente de paralelismo de procesos. En el área de la computación con membranas, esta característica ha inspirado una nueva variante de los sistemas de membranas: los sistemas P basados en neuronas de impulsos (Spiking Neural P Systems en inglés). Estos nuevos sistemas de membranas se asemejan a los sistemas de tejidos pero con características afines al tejido neuronal y no al celular. Las primera ideas de este modelo de computación así como sus características fundamentales están recogidas en el trabajo de Ionescu et al. [IPY06]. La presentación de este nuevo modelo ha dado lugar a una nueva área de investigación en el campo de los sistemas P y en la actualidad son diversas las publicaciones que han aparecido en este campo, entre las que destacamos de forma no exhaustiva las siguientes [CEI⁺07b], [CFI⁺07], [CH⁺08], [CIP⁺06],[IPP⁺07], [IW06].

En esencia, los Spiking Neural P Systems, en adelante sistemas SN P, incorporan ideas procedentes de las redes de neuronas basadas en impulsos [Maa02][MB99]. Básicamente son modelos computacionales inspirados en la forma de comunicación neuronal. Esta forma de comunicación consiste en la propagación de impulsos eléctricos de idéntica forma, llamados *spikes* o impulsos. Las neuronas se disponen en los nodos de un grafo cuyos arcos representan las sinapsis. En el interior de la neurona se encuentran *spikes* representados como múltiples copias de un símbolo, y estos *spikes* serán procesados de acuerdo a las reglas de disparo (spiking rules) y las reglas de

olvido (forgetting rules). Una neurona puede además producir *spikes* que son enviados con un retraso específico a todas las neuronas enlazadas a la neurona emisora. Existe un reloj global de sincronización y en cada unidad de tiempo cada neurona que pueda aplicar una regla la aplicará; por lo tanto el sistema es paralelo a nivel de comunicación pero secuencial a nivel neuronal. Por regla general, aplicar una regla vendrá determinado por una expresión regular asociada a la propia regla y el conjunto de *spikes* que contiene la neurona. Algunas neuronas pueden identificarse como neuronas de entrada y otras como neuronas de salida. Existe un concepto denominado tren de impulsos (Spike train) que no son más que secuencias de impulsos que entran o salen del sistema. Estos impulsos son procesados, es decir, generados, aceptados, computados o traducidos por un sistema SN P partiendo de una configuración inicial. La configuración inicial establece el conjunto de *spikes* de partida en la neurona. Teniendo en cuenta el reloj global del sistema en cada instante se ejecuta una computación que consiste en aplicar una regla de todas las neuronas que sea posible.

En caso de que la computación se detenga, se podrá leer el resultado de la computación de diferentes formas: teniendo en cuenta el número de impulsos obtenidos en una de las neuronas, teniendo en cuenta el número de impulsos emitidos al entorno o teniendo en cuenta el intervalo de tiempo en número de pasos que transcurren entre la emisión de dos impulsos. Por tanto, un sistema SN P puede computar funciones en: $f : \mathbb{N} \rightarrow \mathbb{N}$

2.2. Sistemas P de neuronas: definición y ejemplo

En los sistemas de SN P, como ya se comentó en el apartado anterior, las células son neuronas y forman parte de un grafo dirigido llamado grafo sináptico. El contenido de cada neurona está constituido por múltiples copias de un objeto de un único tipo, denominado impulso o *spike*. Cada célula, además podrá contener en su interior dos tipos de reglas: reglas de disparo y reglas de olvido. Las reglas de disparo permiten a la neurona propagar la información en forma de *spikes* que se acumularán en las neuronas de destino. La condición de disparo de cada regla vendrá determinada por el conjunto de expresiones regulares asociadas a cada regla. En cada unidad de tiempo, si una neurona puede usar más de una regla solo usará una de ellas y ésta será elegida de forma no determinista. Por tanto, las reglas son ejecutadas de manera secuencial en cada neurona pero de forma paralela entre neuronas. Al igual que en otros modelos de computación con membranas, también en los SN Ps se da por supuesto la existencia de un reloj global que regulará el tiempo de todo el sistema, haciendo que el sistema sea síncrono. Cuando una neurona se dispare el modelo establece que permanecerá inactiva por un periodo de tiempo específico; esta característica simula el equivalente al periodo refractario de la neurona biológica. Durante este periodo tampoco podrá ni recibir impulsos ni emitir otros nuevos. Otro aspecto importante, abstraído de la neurona biológica, es el retraso en la recepción de impulsos como consecuencia de las diferentes longitudes de los axones. Este aspecto se modela en los SN Ps asignando un retraso al disparo de alguna de las reglas de cada una de las neuronas.

El otro tipo de reglas que anteriormente citábamos, las reglas de olvido,

se dispararán cuando no pueda ser aplicada ninguna de las reglas de disparo. Esta regla simplemente eliminará un número prefijado de impulsos de la neurona.

Tomando como referencia la formalización matemática descrita en [IPY06] un sistema SN P queda formulado de la siguiente manera:

$\Pi = (o, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in, out)$, donde:

1. $o = \{a\}$ es un conjunto con un único elemento en su alfabeto y modeliza los impulsos o spikes.
2. $\sigma_1, \sigma_2, \dots, \sigma_m$ representan a las neuronas de acuerdo a $\sigma_i = (n_i, R_i), i \leq m$, tal que:
 - a) $n_i \geq 0$ representa el número de *spikes* contenidos en la neurona σ_i .
 - b) R_i es un conjunto finito de reglas de los dos tipos descritos anteriormente, de disparo y de olvido:
 - 1) las reglas de *disparo*, también conocidas como reglas *spike*, quedan definidas por: $E/a^c \rightarrow a; d$ donde E define una expresión regular sobre el a , y $c \geq 1, d \leq 0$ son número enteros. Si $E = a^c$ en ese caso se puede simplificar la regla en la forma: $a^c \rightarrow a; d$. En el caso de que $d = 0$ podrá también ser omitida.
 - 2) las reglas de *olvido* son de la forma $a^s \rightarrow \lambda$ para cualquier $s \geq 1$, teniendo en cuenta la restricción de que para cada regla de la forma $E/a^c \rightarrow a; d$ de tipo disparo perteneciente a R_i tenemos que $a^s \notin L(E)$. $L(E)$ es el lenguaje regular definido por E .

- c) $syn \subseteq 1, 2, \dots, m \times 1, 2, \dots, m$, con $(i, i)/syn$ para $1 \leq i \leq m$, describe el grafo dirigido de sinapsis entre neuronas.
- d) $in, out \in 1, 2, \dots, m$ indica la entrada y la salida de las neuronas de Π .

Una regla de disparo $E/a^c \rightarrow a; d \in R_i$ puede ser aplicada en una neurona σ_i si contiene $K \geq c$ impulsos, y $a^k \in L(E)$. La ejecución de esta regla elimina c impulsos de σ_i y por tanto dejará libres $K - c$ impulsos, preparando un impulso para ser enviado a todas las neuronas σ_j tal que $(i, j) \in syn$. En el caso de que $d = 0$, entonces el impulso será emitido inmediatamente; en caso contrario será emitido d computaciones más tarde. Durante este estado computacional la neurona está cerrada y no puede recibir nuevos impulsos. En el caso de que una neurona envíe un impulso a una neurona que esté cerrada ese impulso se perderá. Una regla de olvido como $a^s \rightarrow \lambda$ puede ser aplicada en una neurona σ_i si contiene exactamente s impulsos y no puede aplicarse ninguna otra regla. La ejecución de esta última regla elimina todos los impulsos s que contenga la neurona σ_i .

El sistema puede ser definido con una configuración inicial indicando en cada neurona el número de impulsos presentes y con todas las neuronas abiertas, es decir, sin neuronas cerradas y listas tanto para emitir como para recibir impulsos. Cada computación se describe mediante el contenido de cada neurona y su estado, que puede ser expresado como el número de pasos de espera hasta que vuelve a estar abierta. Por tanto $\langle r_1/t_1, \dots, r_m/t_m \rangle$ constituye la configuración donde la neurona σ_i contiene $r_i \geq 0$ impulsos y volverá a estar abierta $t_i \geq 0$ pasos más tarde para $i = 1, 2, \dots, m$. Siguiendo esta notación identificaremos el estado inicial como $C_0 = \langle n_1/0, \dots, n_m/0 \rangle$.

Una computación comienza en el estado inicial definido por la configuración inicial. En el modelo de SN P, al igual que en muchas de las primeras

variantes, el número está codificado como el intervalo de tiempo transcurrido entre la inserción de dos entradas en la neurona. Para pasar de una configuración a otra, en cada neurona se elige una regla aplicable y se ejecuta. Por regla general, una computación podría no detenerse; sin embargo, en cualquier caso los impulsos emitidos por la neurona de salida se enviarán al entorno. Los momentos de tiempo en los que una neurona de salida emite un impulso se marcan como 1, y en caso contrario como 0. Esta secuencia binaria se denomina *tren de impulsos* del sistema; este tren de impulsos o tren de *spikes* podría ser infinito en caso de que la computación no se detenga. La esencia de los SN P la constituye la forma de codificación de la salida, los SN Ps codifican la información mediante la distancia entre impulsos consecutivos emitidos al entorno por la neurona de salida.

Se podrán definir SN Ps que funcionen en modo aceptación, en los que un número natural es introducido en la neurona de entrada. Este número natural se codificará como el intervalo de tiempo consecutivo entre dos impulsos y será aceptado si la computación se detiene. Si se ignora la neurona de entrada y por tanto se inicia el sistema con una configuración predefinida, se podrá definir un SN P generativo. También sería posible definir uno que integrase los dos modos de funcionamiento, es decir, generativo o aceptación.

Con estos SN Ps se pueden obtener lenguajes definidos sobre diferentes alfabetos utilizando las reglas de la forma $E/a^c \rightarrow a^p; d$. Para una neurona que cumpla E , se consumirán c impulsos y se producirán p y serán transmitidos a todas las neuronas conectadas después de d pasos. Por tanto, con cada paso o computación el sistema envía i impulsos, a lo que se asocia el símbolo b_i , con lo que se puede obtener un lenguaje sobre el alfabeto con tantos símbolos como número de impulsos se produzcan de forma simultánea. Como extensión se podrán utilizar varias neuronas del sistema como neuro-

nas de salida, lo que permitirá ofrecer salidas de vectores de números y no simplemente números.

Un sistema SN P de grado $m \geq 1$ es una construcción de la forma

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, i_0),$$

donde:

1. $n_i \geq 0$ representa el número de impulsos iniciales contenidos en la neurona σ_i .
2. R_i es un conjunto finito de reglas de los dos tipos descritos anteriormente, de disparo y de olvido:
 - a) regla de *disparo* o también *spike* definida por: $E/a^r \rightarrow a; t$ donde E define una expresión regular sobre el a , y $r \geq 1, t \leq 0$ son números enteros. Si $E = a^r$, en ese caso se puede simplificar la regla en la forma: $a^r \rightarrow a; t$. En el caso de que $t = 0$ podrá también ser omitida.
 - b) reglas de *olvido* serán de la forma $a^s \rightarrow \lambda$ para cualquier $s \geq 1$, teniendo en cuenta la restricción de que para cada regla de la forma $E/a^r \rightarrow a; t$ de tipo disparo perteneciente a R_i tenemos que $a^s \notin L(E)$. $L(E)$ es el lenguaje regular definido por E .
3. $syn \subseteq 1, 2, \dots, m \times 1, 2, \dots, m$, con $(i, i)/syn$ para $1 \leq i \leq m$, constituye el grafo dirigido de sinapsis entre neuronas.
4. $i_0 \in 1, 2, \dots, m$ indica la neurona o neuronas de salida para Π .

2.2.1. Reglas de disparo

Las reglas de disparo descritas anteriormente actúan cuando el número de impulsos presentes en la neurona queda cubierto por la expresión regular E . En ese momento la neurona se dispara, consumiendo r impulsos, y t instantes después se emite un nuevo impulso replicándolo por todas las conexiones sinápticas de las neuronas vecinas. Los impulsos estarán disponibles en las neuronas receptoras en el siguiente paso computacional. En el intervalo t entre el disparo y la liberación se asume que la neurona estará cerrada, omitiendo durante este intervalo la recepción de cualquier otro impulso; por supuesto no podrá ser disparada. En el caso que $E = a^r$ simplemente se escribe la regla de la forma $a^r \rightarrow a : t$.

Existe una versión extendida de reglas de disparo introducida por [PP07] y [CII⁺08] en la cual la neurona tras cada disparo produce $p \geq 1$ impulsos en lugar de uno, quedando la regla de la forma $E/a^r \rightarrow a^p : t$

Los SN P usan el paralelismo a nivel de sistema; en un paso, todas las neuronas que pueden usar una regla tienen que usarla. Sin embargo, solo se puede aplicar como mucho una regla dentro de la neurona y además ésta deberá ser elegida de forma no determinista.

2.2.2. Reglas de olvido

Las reglas de olvido definidas anteriormente se aplican sólo cuando la neurona contiene exactamente s impulsos, lo que a su vez implica que serán todos eliminados de la neurona. La definición anterior de SN P no permite elegir de forma no determinista entre las reglas de olvido y disparo, es decir, en un paso una neurona tiene que dispararse u olvidar sin que sea posible elegir libremente entre ambas.

2.2.3. Definición de Computación

Una computación en un sistema SN P es una secuencia de pasos durante la cual se van aplicando las reglas de la forma descrita anteriormente. Una computación comienza con una configuración inicial para cada neurona σ_i conteniendo n_i *spikes* tal que, $1 \leq i \leq m$. La computación se detiene si alcanza un estado en el que no se pueden aplicar reglas, constituyendo una parada fuerte (strong halting) si no queda ningún impulso en el sistema.

La forma usual de interpretar las salidas de los sistemas SN P es teniendo en cuenta los intervalos entre disparos de la neurona identificada como neurona de salida i_0 , teniendo en cuenta dos casos:

- caso fuerte: Cuando se tiene en cuenta el intervalo exactamente entre el disparo de 2 impulsos. En este caso al conjunto de enteros generados se le denota como $N_2(\pi)$.
- caso parada fuerte: Si además del caso fuerte, sólo se tienen en cuenta los casos de parada, los conjuntos resultado de estos sistemas serán: $N_2^h(\Pi)$ y $N_2^h(\Pi)$.

En la figura 2.1 se incluye un ejemplo básico de un sistema P basado en impulsos. Este ejemplo gráfico ilustra la posible representación basada en grafos de un SN P que si bien es útil para facilitar el entendimiento del funcionamiento básico no es suficiente para modelizar sistemas complejos. Como alternativa se puede usar la notación formal para modelizarlo. Este sistema funcionará en modo generativo y por tanto no se indica neurona de entrada; además se utiliza la forma simplificada para especificar las reglas: dada una regla de la forma $E/a^c \rightarrow a; d$ donde $L(E) = \{a^c\}$ se podrá especificar la regla de la siguiente forma: $a^c \rightarrow a; d$.

Figura 2.1: Ejemplo básico de SNP

$$\Pi_1 = (O, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7, syn, out)$$

$$O = \{a\}$$

$$\sigma_1 = (2, a^2 \rightarrow a; 0, a \rightarrow \lambda)$$

$$\sigma_2 = (2, a^2 \rightarrow a; 0, a \rightarrow \lambda)$$

$$\sigma_3 = (2, a^2 \rightarrow a; 0, a \rightarrow \lambda)$$

$$\sigma_4 = (0, a \rightarrow a; 1, a \rightarrow a : 0)$$

$$\sigma_5 = (0, a \rightarrow a : 0)$$

$$\sigma_6 = (0, a \rightarrow a : 0)$$

$$\sigma_7 = (2, a^2 \rightarrow a; 0, a^3 \rightarrow \lambda)$$

$$syn = \{(1, 4), (3, 6)(4, 1), (4, 2), (4, 3), (5, 1), (5, 2), (6, 3), (4, 7), (5, 7), (6, 7)\}$$

$$out = 7$$

Esta formalización equivale a la representación gráfica de la figura 2.1.

En la primera ejecución, las neuronas $\sigma_1, \sigma_2, \sigma_3$ y σ_7 podrán ser disparadas enviando un impulso a σ_4, σ_5 y σ_6 , y a su vez σ_7 emitirá la primera salida.

En el siguiente paso, en la neurona σ_4 se podrá elegir de forma no determinista cuál será la regla a disparar. En caso de ejecutarse la regla $a \rightarrow a : 1$ la emisión del *spike* se retrasará una ejecución.

En caso de aplicarse la regla de olvido $a \rightarrow \lambda$ los impulsos serán eliminados de la neurona. Es importante destacar que si se diese el caso en el que una neurona σ_x recibiese simultáneamente un pulso de otra neurona σ_y , el impulso de la neurona σ_y se perdería ya que la neurona σ_x estaría cerrada.

Capítulo 3

Planteamiento del Problema

A lo largo de los capítulos anteriores se han resumido las características fundamentales de los sistemas de computación celular basados en membranas, también conocidos como sistemas P. En el capítulo 1 se detallan tanto los elementos que caracterizan al modelo primigenio de sistemas P como aquellas extensiones posteriores al modelo inicial. Al mismo tiempo, en el capítulo 2 se estudia un tipo específico de estos sistemas, los sistemas P de neuronas basados en impulsos, que en favor de la claridad a lo largo de la memoria identificamos con el acrónimo SN P del inglés: «Spiking Neural P Systems».

Los sistemas P inicialmente propuestos por Paun [Pău02], así como sus derivados, están englobados en el área de la computación celular con membranas. El área de la computación con membranas se enmarca en la computación natural y en informática teórica y tiene como objetivo abstraer ideas y modelos de cómputo a partir de la estructura o funcionamiento de las células biológicas, independientemente de cómo estén organizadas, ya sea en tejidos u otras estructuras de orden superior.

Los SN P, fueron inicialmente propuestos por Ionescu en [IPY06]. Este nuevo modelo toma como base los trabajos en las redes de neuronas basadas

en impulsos investigadas por Maass y Bishop [MB99] y [Maa02].

Ambos modelos comparten características como es su capacidad de paralelismo computacional, trabajan con símbolos como unidades mínimas información y evolucionan de acuerdo a una serie de reglas previamente definidas; en caso de que dos reglas puedan ser aplicadas en un mismo paso computacional estas se seleccionan de forma no determinista. Existe un reloj global que gobierna todos los pasos computacionales y que permite gobernar el funcionamiento de los sistemas.

Los sistemas P son una estructura jerárquica de regiones de cómputo con reglas en su interior. Pueden funcionar con símbolos o cadenas de símbolos que evolucionan de acuerdo a unas reglas preestablecidas en el sistema. Las reglas pueden tener o no tener prioridades, algunas permiten que la membrana que las encierra sea disuelta, otras envían objetos a una membrana hija más interna o a una membrana padre más externa. Poseen reglas que pueden dividir la membrana o simular el intercambio de elementos entre membranas y esto no es más que una pequeña enumeración de la cantidad de modificaciones, añadidos o evoluciones al modelo inicial, muchas con alguna raíz biológica y otras en cambio centradas en incrementar la potencia computacional de estos sistemas.

Los SN P poseen una estructura de grafo que interconecta cada una de las neuronas que forman el sistema. Las reglas, una o varias, se preestablecen en el interior de cada neurona y gobiernan el paso y evolución de impulsos dentro del sistema. Las reglas pueden ser principalmente de dos tipos: reglas de disparo que en función del número de impulsos que posean producen impulsos a sus sinapsis o reglas de olvido que reducen el número de *spikes* en el interior. También incluyen la posibilidad de adjuntar a las reglas de disparo un retraso a la hora de ejecutar la regla.

Pues bien, en el caso de los SN P con todas sus características y tomando como punto de partida los trabajos de Ibarra et al. [IPP⁺07], se plantea la siguiente pregunta: ¿sería posible tal vez, limitar alguna de las características de este nuevo modelo de cómputo manteniendo su capacidad computacional?. En caso de eliminar características, ¿qué capacidad computacional se obtendría con el sistema restringido?. Por otro lado, para el caso de los sistemas P, ¿qué característica biológica puede ser añadida a estos sistemas demostrando mayor eficiencia computacional? ¿podrían los sistemas P con proteínas en las membranas mostrarse computacionalmente eficientes resolviendo problemas del ESPACIOP-Completo como es el caso del SAT Cuantificado (QSAT)?

En los dos capítulos presentados a continuación se da respuesta a las cuestiones anteriormente planteadas. En concreto, en el capítulo 4 y tomando como base los trabajos de simplificación llevados a cabo por Ibarra et al. en [IPP⁺07], se demuestra que es posible simplificar las expresiones regulares usadas en las reglas y eliminar los retrasos sin perder la universalidad. También se demuestra que es posible usar expresiones regulares simples y mantener la condición fuerte de parada o usar expresiones regulares simples con el sistema funcionando en modo aceptación. Por último, en el capítulo 5 se incluyen los resultados del análisis de la eficiencia computacional de los sistemas P con proteínas. En ese capítulo se demuestra que es posible construir un sistema P que resuelva el problema del QSAT en tiempo polinómico.

Parte II

Soluciones Propuestas

Capítulo 4

Desarrollo de nuevas Formas Normales para los sistemas P basados en Neuronas de Impulsos

Este capítulo presenta los resultados de investigación derivados del análisis de los sistemas P basados en neuronas de impulsos, en adelante SN P, acrónimo usado de manera habitual en el área y procedente del inglés «Spiking Neural P Systems». En los primeros apartados se realiza una breve introducción a los trabajos previos que han servido como base para lograr resultados novedosos en la materia. Posteriormente, en los apartados siguientes se incluyen las nuevas formas normales descubiertas así como su correspondiente demostración. Todos estos resultados se encuentran publicados en [GPRS09] y [GPRS08]

4.1. Definiciones previas

Denotamos como:

$$Spik_2^\beta(E)P_m^{(nsym)}(rule_k, cons_p, forg_q, dley_r, outd_s)$$

a la familia de todos los conjuntos $N_\beta^2(\Pi)$ con $\beta \in \{h, \underline{h}\}$ para todos los sistemas Π , con como mucho m neuronas, cada neurona con un máximo de k reglas y s sinapsis de salida, cada una de las reglas consumiendo como mucho p *spikes* y un retraso máximo de r . Las reglas de olvido no eliminarán más que q impulsos. El símbolo *nsyn* incluido en esta definición indica que el sistema no es síncrono y la E indica el uso de algunas extensiones a las reglas básicas. También podemos considerar las reglas de la forma $rule_k^*$ si las reglas de disparo son de la forma $E/a^r \rightarrow a; t$ con expresiones regulares de la forma $E = \lambda$ o $E = a^*$.

En el caso de que el sistema SN P funcione en modo aceptación, se indica mediante el subíndice *acc*. El sistema obtendrá una entrada en forma de intervalo entre dos impulsos consecutivos. Los impulsos llegarán desde el exterior a la neurona indicada por i_0 . La entrada n será aceptada si tras recibir los dos impulsos el sistema se detiene, no tiene que ser necesariamente en el momento de recibir el segundo impulso. Además el sistema SN P debe comportarse de forma determinista. Para indicar el determinismo se añade el prefijo D a la clase correspondiente de conjuntos de enteros aceptados por el sistema.

Además de la definición formal de un SN P también se presentan a continuación las definiciones básicas de la máquina de registros. Esta máquina se utiliza para probar el poder computacional de estos sistemas. Para ello se construye un sistema que pueda simular dicha máquina. Si lo logra, se puede decir que el sistema construido es equivalente en poder computacional a la máquina de registros.

Una máquina de registros no determinista [Min67] se define como $M = (m, H, l_0, l_h, I)$ donde m define el número de registros, H es el conjunto de

etiquetas de instrucciones, l_0 es la etiqueta inicial de comienzo, l_h identifica la etiqueta de parada (asignada a la instrucción *HALT*) e I es el conjunto de instrucciones. Cada una de las etiquetas pertenecientes a H define una instrucción de I aunque una misma instrucción podrá ser asignada a más etiquetas. Las instrucciones por tanto se definen de la siguiente forma:

- $l_1 : (ADD(r), l_2, l_3)$ suma 1 al registro r y luego salta a una de las instrucciones etiquetada como l_2, l_3 ,
- $l_1 : (SUB(r), l_2, l_3)$ si el registro no está vacío se le resta 1 y se salta a la instrucción etiquetada como l_2 , en otro caso se salta a l_3 ,
- l_h *HALT* (la instrucción de parada).

Cuando las máquinas de registros funcionan en modo generativo, la computación comienza con todos los registros a 0 y se aplican las instrucciones en el orden indicado por las etiquetas. Cuando se alcanza la instrucción de parada el número n almacenado en el primer registro se considera como el resultado del cómputo. Al conjunto de números generado por las máquinas de registros M se le denota como $N(M)$. Las máquinas de registros pueden generar recursivamente todos los conjuntos de los enteros no negativos. Sin perder su capacidad generativa podemos asumir que en la configuración de parada todos los registros excepto el primero están vacíos.

Además de en modo generativo las máquinas de registro también funcionan en modo aceptación. En este modo de funcionamiento se inician con un número n en el primer registro estando el resto de registros vacíos. Si en algún momento de los pasos de computación el sistema se para, diremos que acepta el número n . Las máquinas de registros en modo aceptación también son universales incluyendo también a aquellas que son deterministas, es decir, son deterministas aquellas máquinas de registros donde $l_2 = l_3$ para la

instrucción $l_1 : (ADD(r), l_2, l_3)$. En el caso de las máquinas deterministas la instrucción ADD puede ser denotada como $l_1 : (ADD(r), l_2)$. Una vez más al igual que anteriormente podemos asumir que en la configuración de parada todos los registros están vacíos.

4.2. Trabajos previos

El primer trabajo de investigación que estudió el poder computacional de las características elementales de un sistema P basado en neuronas de impulsos y como afectaba su omisión, fue el de Ibarra *et al* publicado en 2007 [IPP⁺07]. Su sección 5 demuestra que el grado sináptico de salida de cualquier neurona en un sistema SN P puede ser disminuido a 2 añadiendo algún retraso en el sistema, a menos que la neurona sea usada en un circuito síncrono en el que añadir un retraso provoque un funcionamiento anormal. Afortunadamente esto no es un caso frecuente, por tanto esta propiedad puede ser usada simultáneamente con otras muchas restricciones. Otro resultado análogo es restringir a 2 el grado sináptico de entrada a la neurona sin que el sistema pierda la universalidad tal y como se demuestra en Paun et al [PPJS06].

Otra característica importante es la estudiada en la sección 4 del artículo de Ibarra; en este apartado, los autores demuestran que el sistema no pierde su universalidad aunque se eliminen los retrasos en las reglas de disparo. En esencia, se puede reemplazar una neurona con un retraso de n que no use su periodo refractario, por una secuencia de n neuronas sin retraso. Sin embargo, existen casos más complicados en los que hay que encontrar una solución ad-hoc.

A continuación se resumen los resultados más relevantes que se han ob-

tenido:

Teorema 4.2.1 (*Ibarra et al. [IPP⁺07]*)

$$(i) \text{Spik}_{\underline{2}}^{\beta} P_*(rule_3, cons_4, forg_q, dley_0, outd_2) = NRE,$$

$$(ii) \text{DSpik}_{2,acc}^{\beta} P_*(rule_2, cons_3, forg_2, dley_2, outd_2) = NRE,$$

donde $\beta \in \{h, \underline{h}\}$ o se omite β , y $q = 5$ si $\beta = \underline{h}$ en caso contrario $q = 4$

Algunos de los casos cubiertos por las dos sentencias anteriores no derivan directamente de los trabajos de Ibarra. Pero tomando como base lo descrito en su sección 4 se puede deducir, empleando para ello la técnica para reducir el grado de salida de su sección 5.

Otras características estudiadas por Ibarra, en la sección 6 de su artículo, son las reglas de olvido. Al igual que con el caso de los retrasos, prescindir de ellas tampoco hace que se pierda la universalidad. El principio que sigue la demostración es reemplazar la reglas de olvido por expresiones regulares de la forma $a^x(a^y)^*$, donde, $x \neq y$. Una neurona con una expresión regular de este tipo puede acumular múltiples *y spikes* sin que afecte a su funcionamiento ya que estos *spikes* o impulsos serían consumidos por una regla de olvido en un sistema SN P normal. Sin embargo, esta modificación no permite mantener la característica de condición fuerte de parada.

Teorema 4.2.2 (*Ibarra et al. [IPP⁺07]*)

$$(i) \text{Spik}_{\underline{2}}^{\beta} P_*(rule_2, cons_3, forg_0, dley_1, outd_2) = NRE,$$

$$(ii) \text{DSpik}_{2,acc}^{\beta} P_*(rule_2, cons_3, forg_2, dley_1, outd_2) = NRE,$$

donde $\beta = \{h, \underline{h}\}$ o se omite β

La declaración (ii) tampoco aparece en esta forma en la publicación de Ibarra et al. [IPP⁺07], pero se puede hacer uso del módulo *INPUT* para obtener un sistema que no use reglas de olvido y funcione en modo aceptación.

Además el módulo *ADD* que se usa en la demostración puede ser fácilmente convertido en determinista omitiendo alguna neurona.

En relación con la declaración (i) derivada del Teorema 6.1 de Ibarra, que establece $Spik_2^\beta P_*(...) = NRE$, pero tomando como base la demostración y en concreto el módulo *FIN* únicamente se producen de forma exacta 2 spikes; es por ello que reemplazamos la notación $Spik_2$ por $Spik_{\underline{2}}$.

La secuencia de resultados obtenidos en Ibarra et al. concluyen en su sección 7. En esta última sección trata con las expresiones regulares en las reglas de la forma $E/a^r \rightarrow a; t$, en particular con aquellas restringidas por $E = \lambda$ o $E = a^*$. De los resultados que obtiene, a priori parecía que ninguna otra restricción sería posible sin perder la universalidad, por ejemplo eliminando la posibilidad de $E = a^*$, que restringiría la cantidad de estados internos a un sistema de números finitos. Sin embargo, los resultados de la sección 7 de Ibarra fueron extendidos por García-Arnagu et al.[GPRS09], artículo en el que además se añadieron casos como la condición fuerte de parada y el modo aceptación. Estos resultados así como sus demostraciones aparecen descritos en esta monografía en las secciones del capítulo 4. A continuación simplemente se resumen:

Teorema 4.2.3 (*Ibarra et al.; García-Arnau et al. [IPP⁺07]*) [*GPRS09*]

(i) $Spik_{\underline{2}}^\beta P_*(rule_2, cons_2, forg_q, dley_2, outd_2) = NRE$,

(ii) $DSpik_{\underline{2}, acc}^\beta P_*(rule_2, cons_2, forg_q, dley_2, outd_2) = NRE$,

donde $\beta = \{h, \underline{h}\}$ o se omite β y $q = 2$ cuando $\beta = h$, en caso contrario $q = 1$

El principio seguido en la demostración está relacionado con la realidad biológica. Las neuronas biológicas se disparan cuando acumulan suficientes impulsos como para llegar a un umbral. Este funcionamiento es imitado con

las reglas del tipo $a^*/a^r \rightarrow a; t$. Otro hecho conocido es el ciclo circulante de impulsos que en las neuronas biológicas juega un rol esencial. Ahora se pueden eliminar las expresiones regulares de las reglas y consecuentemente restringir el modelo a neuronas que se disparan al superar el umbral. Entonces las neuronas con muchos impulsos acumulados se dispararán cada pocos pasos, de acuerdo al retraso definido, a menos que estén bloqueadas. Por tanto, si se quisieran almacenar impulsos para un uso posterior debería existir un ciclo circulante de impulsos que evitase el vaciado de cada neurona. En otras palabras, esas dos características biológicas de las redes de neuronas están matemáticamente relacionadas.

Otro problema frecuente estudiado en la investigación de los sistemas P es la sincronidad frente a la no sincronidad. Este problema fue tratado por Cavaliere [CEI⁺07a] en el contexto de los sistemas P. De hecho, el artículo citado demuestra que aunque se elimine el reloj global al sistema y el sistema sea asíncrono se sigue manteniendo su universalidad. Sin embargo será necesario usar una serie de reglas adicionales que permitan a las neuronas enviar más de un impulso simultáneamente.

Teorema 4.2.4 (*Cavaliere et al. [CEI⁺07a]*)

- (i) $Spik_{tot}^\beta EP_*^{nsym}(rule_*, cons_*, forg_*, dley_0, outd_*) = NRE$,
- (ii) $DSpik_{tot,acc}^\beta EP_*^{nsym}(rule_*, cons_*, forg_*, dley_0, outd_2*) = NRE$.

La notación *tot* del teorema anterior indica que se tendrán en cuenta todos los impulsos enviados desde el entorno. Hay que tener en cuenta que eliminar la sincronidad tiene una influencia enorme sobre otros parámetros; en este caso su valor depende de los detalles de la máquina de registros, de forma particular, en el número de instrucciones *SUB* actuando sobre varios

de los registros, y que por tanto no estará limitada. Es probable que el valor de grado de salida pueda ser nuevamente limitado a 2, pero sigue siendo un problema abierto si la asincronía, sin usar las extensiones de las reglas en los sistemas SN P, da lugar a un SN P universal.

Por último, el artículo [GPRS09] extendiendo los resultados de Ibarra [IPP⁺07], demuestra que si se eliminan simultáneamente retrasos y expresiones regulares el sistema mantiene su universalidad simplemente usando sus reglas de olvido. La idea fundamental de la demostración está basada en el hecho de que las neuronas con retrasos usan su periodo refractario para perder *spikes* y pueden ser simuladas por los módulos de la figura 4.1 sin usar retrasos ni expresiones regulares.

Se ha encontrado una solución individual para neuronas combinando reglas con y sin retrasos de forma no determinista. Si bien también hay que resaltar que ciertos valores de los parámetros se han incrementado con respecto a los valores del Teorema 4.2.3

Teorema 4.2.5 (*García-Arnau et al. [GPRS09]*)

- (i) $Spik_2^\beta P_*(rule_3^*, cons_3, forg_3, dley_0, outd_2) = NRE$,
 - (ii) $DSpik_{2,acc}^\beta P_*(rule_3^*, cons_3, forg_3, dley_0, outd_2^*) = NRE$.
- donde $\beta \in \{h, \underline{h}\}$ o se omite β

Una vez más, siguiendo el Teorema 3.1 publicado en [GPRS09] que establece que no implica la condición fuerte de parada, la construcción de la demostración garantiza que únicamente los *spikes* que permanecen en el sistema después de la parada se encuentran en el módulo *FIN*. Por lo tanto si se reemplaza el módulo *FIN* por un módulo alternativo como el presentado en la figura 4.7 se garantiza la condición fuerte de parada. Además resulta sencillo convertir en determinista el módulo *ADD* añadiendo el módulo *FIN* de la figura 4.8.

4.3. Formas Normales Fuertes

Tal y como se ha visto en 4.1 se puede denotar de la forma

$$Spik_2^\beta P_m(rule_k, cons_p, forg_q, dley_r, outd_s)$$

a la familia de todos los conjuntos de $N_2^\beta(\Pi)$ tal que $\beta \in \{h, \underline{h}\}$ para todos los sistemas con un máximo de m neuronas, en los que cada neurona tenga como mucho k reglas, s sinapsis de salida, cada una de las reglas de disparo consume un máximo de p spikes, teniendo un retraso máximo de r , cada una de las reglas de olvido eliminará no mas de q spikes. También se podría denotar $rule_k^*$ en el caso de especificar reglas de disparo de la forma $E/a^r \rightarrow a; t$ donde la expresión regular E sería: $E = \lambda$ o $E = a^*$.

El primer trabajo de análisis del poder computacional centrado en la simplificación de esta clase de sistemas es el de *Ibarra et al* [IPP⁺07]. En dicho trabajo se presentan las primeras formas normales descubiertas. El punto de partida de los resultados de estas tesis es el teorema 7.1 del citado artículo e incluido aquí como teorema 4.3.1. Para su demostración se construye una máquina de registros, como la descrita en la sección 4.1, mediante SN Ps. Si se consigue construir una máquina de registros usando sistemas SN P con las restricciones del teorema quedará demostrada su universalidad.

Teorema 4.3.1 $Spik_2^\beta P_*(rule_2^*, cons_2, forg_1, dley_2, outd_2) = NRE$, donde $\beta = h$ o β se omite

Para la demostración de este teorema *Ibarra et al.* [IPP⁺07] construye un sistema SN P que simule una máquina de registros y utilice únicamente expresiones regulares de la forma λ o a^* . La idea fundamental de la demostración es la construcción de un módulo con un circuito dinámico de *spikes* en el que simular un registro de la máquina de registros. En este módulo se

consigue añadir o remover *spikes* del circuito cerrado ajustando el tiempo mediante la entrada de spikes. La entrada de *spikes* al registro estará gobernada por los módulos *ADD* y *SUB* que simulan las instrucciones de la máquina de registros.

En las siguientes secciones y tomando como base este resultado, se proponen las nuevas formas normales descubiertas.

4.4. Forma Normal eliminando retrasos y simplificación de las expresiones regulares

En esta sección se consideran las mismas restricciones de las expresiones regulares del sistema *SN P* definido en el teorema 4.3.1 pero no se tendrán en cuenta los retrasos. Sorprendentemente el *SN P* resultante sigue siendo universal. Además, excepto para el caso no determinista, la neurona c_3 del módulo *ADD* utilizará una regla de la forma $a^*/a \rightarrow o$ o $a^r \rightarrow a$ y como mucho dos reglas $a^r \rightarrow \lambda$ con $r, y \leq 3$. Finalmente, el grado de cada neurona también está restringido a 2. Al comparar los resultados obtenidos con 4.3.1 podemos observar que la eliminación de los retrasos conlleva un cierto coste computacional en favor de otros parámetros, como el máximo grado de reglas de olvido, el número de reglas por neurona y el número máximo de *spikes* consumidos tras aplicar una regla.

Los principales resultados en esta sección están basados en el hecho de que las neuronas con retrasos, que usan el periodo refractario de la neurona para ignorar los *spikes* que reciben durante ese periodo, pueden ser simuladas por ciertos módulos de neuronas sin retrasos y empleando expresiones regulares simples. La construcción de estos módulos da lugar al siguiente lema:

Lema 4.4.1 1. *El comportamiento de entrada/salida de una neurona con la regla simple $a \rightarrow a; 1$ puede ser simulada por el módulo Π_{d_1} de la figura 4.1 .*

2. *El comportamiento de entrada/salida de una neurona con una regla simple $a \rightarrow a; 2$ puede ser simulada por el módulo Π_{d_2} en la figura 4.2.*

Demostración. Tomando como referencia la figura 4.1 supongamos que X emite dos *spikes* consecutivamente al módulo Π_{d_1} . El *spike* se recibe en c_1 y c_2 en el instante t y será emitido hacia c_3 en el instante $t + 1$, mientras tanto el segundo *spike* emitido por X alcanza c_1 y c_2 . Seguidamente la neurona c_3 se dispara, consumiendo solo uno de los dos *spikes* y emitiendo un *spike* en $t + 2$. En ese momento c_1 y c_2 se disparan nuevamente hacia c_3 que ahora contendrá tres *spikes* que serán suprimidos en $t + 3$ al aplicar la regla de olvido $a^3 \rightarrow \lambda$.

Insertando otra neurona c_4 , usando la regla $a \rightarrow a$ entre c_3 y la salida Y y añadiendo una retropropagación de c_4 a c_3 , se puede simular el módulo Π_{d_2} pero con retardo de dos. Su funcionamiento es análogo al descrito para el módulo Π_{d_1}

Teorema 4.4.2 $DS\text{pik}_{\underline{2}}^{\beta}P_*(rule_3, cons_3, forg_3, dley_0, outd_2) = NRE$, donde $\beta = h$ } o β se omite

Demostración: Tomando como base la demostración del Teorema 7.1 del artículo [IPP⁺07] se reemplazan todas las neuronas que tienen retraso, según el caso, por una cadena de neuronas (aquellas donde el retraso únicamente se use para posponer la emisión de un impulso) o bien por uno de los módulos Π_{d_1} o Π_{d_2} derivados de la demostración del Lema 4.4.1 (ver figuras 4.1 y 4.2).

De acuerdo con la demostración del teorema 7.1 antes mencionado, el registro dinámico almacena un número igual al número de *spikes* que están

Figura 4.1: Módulo Π_{d_1} neurona con retraso 1 Figura 4.2: Módulo Π_{d_2} neurona con retraso 2

circulando de manera continua en el circuito cerrado $r - z - t - u$. En este circuito se pueden sustituir algunas de esas neuronas por los módulos Π_{d_1} o Π_{d_2} con dos neuronas de entrada c_1 y c_2 . Esto implica duplicar las sinapsis de la máquina de registros. Además, mantener el grado ≤ 2 implica replicar algunas de las neuronas junto con sus respectivas sinapsis para poder mantener la misma funcionalidad. En la figura 4.3 se presenta el resultado de la máquina de registros dinámico después de introducir todos los cambios.

Cuando el grupo de neuronas U se dispara emite un pulso que se divide generando 6 nuevos *spikes* hacia el módulo R . Tanto r_1 como r_2 necesitan recibir tres *spikes* para garantizar que R se disparará cada 3 computaciones mientras tenga spikes. Por tanto, para llevar a cabo la operación *ADD* deberán ser lanzadas tres *spikes* simultáneos a las neuronas r_1 y r_2 . Como el módulo R está conectado con S y T , cuando R emite un *spike* se envía simultáneamente a s_1, s_2, t_1 y t_2 . Por último, los módulos S y T también están conectados mediante sinapsis con el módulo U . En este caso, para que se dispare el módulo U , S y T tendrán que enviar tres *spikes* cada una para alimentar a las tres neuronas de U .

Las computaciones en nuestro registro dinámico son cíclicas cada seis pasos cuando el valor $n = 1$. Los 6 *spikes* presentes en el paso t se consumen para dar lugar a un nuevo *spike* hacia S y T en el instante $t + 3$. En cambio, S y T envían tres spikes, uno a cada neurona de U , en el instante $t + 5$. Por último, U emite otra vez seis *spikes* a R en $t + 6$ cerrando el ciclo. Además, el ciclo de computación de seis pasos se compone de 3 pasos idénticos cuando almacena un valor $n > 1$.

Simulación de la instrucción *ADD* $l_i : (ADD(r), l_j, l_k)$ (Figura 4.4)

Para evitar el uso de retrasos en el módulo *ADD*, se ha reemplazado la neurona no determinista c_4 por una nueva que contiene tres reglas c_3 . En el

Figura 4.3: Máquina de registros con circulación dinámica de *spikes* y que no usa demora/retrasos

Figura 4.4: Módulo que simula $l_i : (ADD(r), l_j, l_k)$

paso uno, las neuronas l_i, l'_i, l''_i y l'''_i envían un *spike* a c_1 y c_2 y un grupo de 6 al módulo R del registro dinámico incrementando en uno el valor almacenado. En el siguiente paso dos neuronas alcanzan a las neuronas c_3 y c_4 . Por tanto, si se elige la regla $a^2/a \rightarrow a$, se enviará un *spike* a c_6 y c_7 mientras el otro permanece en c_3 . En el siguiente paso, c_3 aplica la regla $a \rightarrow a$ y dos nuevos *spikes* llegarán a c_6 y c_7 , uno procedente de c_3 y el otro de c_5 . Esto hace que c_7 se dispare mientras c_6 aplica la regla de olvido y elimina sus tres spikes. Por otro lado, si c_3 primero elige la regla $a^2 \rightarrow a$, emitirá un *spike* a c_6 y c_7 que recibirán otra nueva en el siguiente paso desde c_5 . Esto hace que c_6 se dispare, llevándonos a la computación de l_j , mientras c_7 olvida los dos spikes. Finalmente, las neuronas c_8 y c_9 pueden ser reemplazadas por una cadena de seis neuronas con 0 retraso.

Simulación de la instrucción SUB $l_i : (SUB(r), l_j, l_k)$ (Figura 4.5)

El módulo *SUB* únicamente contiene una neurona c_5 que hará uso de su periodo refractario, será reemplazada por el módulo de tipo Π_{d1} . El resto de neuronas que tengan retrasos serán reemplazadas por una cadena básica de neuronas con 0 como retraso. Algunas neuronas serán replicadas con el objetivo de mantener el grado sináptico ≤ 2 . El módulo se inicia cuando se envía un *spike* a la neurona l'_i . Esto provoca que en el paso tres dos *spikes* salgan hacia el módulo *V*, dando comienzo a la desincronización del registro dinámico y decrementando el valor del registro en una unidad. Seguidamente el módulo *T* es forzado a enviar un *spike* a c_6 en el paso 6. Después de eso, c''_4 se dispara en el paso 8 y continúa con la ejecución de la instrucción l_j . En el caso de que el registro almacene un cero, la neurona c''_4 no se disparará en el paso 8, ya que c'_4 anulará el *spike* mediante su regla de olvido emitida por c_1 en el paso 6. Por tanto, dos *spikes* alcanzarán c_7 en el paso 11 de la computación y se continuará con la ejecución de la instrucción l_k . Es importante hacer notar que en caso de existir más de un módulo de *SUB* que decrementase el registro se necesitarían más conexiones entre *T* y la neurona c_4 .

Figura 4.5: Módulo que simula $l_i : (SUB(r), l_j, l_k)$

Fin del proceso de computación FIN - módulo FIN (Figura 4.6)

El módulo FIN tiene el mismo comportamiento que el descrito en el

teorema 7.1 publicado en [IPP⁺07]. Sin embargo, al igual que en el módulo *SUB*, se han realizado cambios estructurales con la intención de eliminar los retrasos de las neuronas l_h, c_9, c_{10}, d_9 y d_{10} . Las neuronas c_9 y d_9 han sido sustituidas por dos neuronas básicas con cero como retraso. Por otro lado, las neuronas c_{10} y d_{10} , que utilizan su periodo refractario, se han reemplazado por el módulo Π_{d_1} . Por último, se han duplicado las neuronas c_2, c_7, d_7 y c_8 para mantener el grado sináptico ≤ 2 . También es importante hacer notar que para mantener la claridad del diagrama no se ha reemplazado la neurona c_3 por las cuatro neuronas básicas.

Figura 4.6: Módulo FIN

4.5. Forma Normal basada en expresiones regulares simplificadas

El teorema 7.1 publicado en [IPP⁺07] e incluido en esta monografía como Teorema 4.4.2 en ningún caso considera la condición de parada estricta o fuerte, ni el modo de aceptación de los sistemas SN P. En esta sección se ampliará el teorema 4.3.1 y se demostrará que sigue siendo válido aunque se impongan nuevas restricciones. A continuación se analiza en primer lugar lo que ocurre en el caso de la condición de parada estricta o fuerte.

Teorema 4.5.1 $Spik_2^h P_*(rule_2^*, cons_2, forg_2, dley_2, outd_2) = NRE$

Demostración: Se parte de la asunción de que todos los registros de una máquina, excepto el primer registro, están vacíos al final de la computación. Por tanto se verifica que siguiendo la demostración del teorema 7.1 incluida en [IPP⁺07] sólo las neuronas que contienen *spikes* en el momento de la parada están presentes en el módulo FIN. Se propone por tanto, un nuevo módulo *FIN* que satisface la condición fuerte de parada cuyo diagrama se incluye en la figura 4.7. Para lograr la condición fuerte de parada y simplificar su construcción es necesario incluir otra restricción al módulo original: las reglas de la forma $a^r \rightarrow a; t$ y $a^s \rightarrow \lambda$ en la misma neurona, tal que $s < r$.

Dejando el registro de salida 1 con un valor $n > 1$ y con el ciclo de neuronas 1 y c_1 conteniendo n *spikes* ($n - 1$ en la neurona 1 y un *spike* en la neurona c_1), ambas neuronas se disparan en cada ejecución del sistema.

En el paso de ejecución 1 la neurona l_h recibe un *spike* y se dispara. La neurona de salida (*out*) se dispara por primera vez en el paso 8, recibiendo un impulso de la neurona c_5 . En el paso 3, tanto la neurona c_2 como la c_3 , reciben *spikes* y comenzarán a dispararse en cada paso. La neurona c_4

Figura 4.7: Módulo FIN con expresiones regulares simplificadas y condición fuerte de parada

recibirá en cada paso dos *spikes* que serán eliminados. Simultáneamente, en el paso 5 la neurona c_1 comienza a recibir *spikes* de la neurona c_7 que decrementa en 1 el número de *spikes* en cada paso en el ciclo $1 - c_1$ (los *spikes* se consumen en la neurona c_1). Después de n pasos de cómputo todos los *spikes* del ciclo $1 - c_1$ se eliminan. En el paso $n + 5$ la neurona c_2 no recibe un spike. Consecuentemente, en el paso $n + 7$ la neurona c_4 recibe solo un *spike* de la neurona c_8 y se dispara. Finalmente, en el paso $n + 8$, exactamente n pasos después del primer disparo, la neurona *out* produce una nueva salida disparándose por segunda vez.

Después de eliminar todos los *spikes* en el ciclo $1 - c_1$, la neurona c_7 envía dos *spikes* más a c_1 . Para eliminar esos dos *spikes* y dar comienzo al ciclo $1 - c_1$, son las c_9 y c_{10} las neuronas que juegan el papel central. Envían dos *spikes* adicionales a la neurona c_1 la cual, gracias a la reglas $a^2 \rightarrow \lambda$ limpia el ciclo $1 - c_1$ y finaliza el cómputo.

En caso de $n = 1$, la neurona 1 se dispara en cada paso y la neurona c_1 se dispara cada paso impar. Para eliminar ese *spike* de la neurona c_1 , esta debe recibir su primer *spike* de la neurona c_7 en un paso impar. Yendo hacia atrás en cascada, también la neurona l_h debería recibir un *spike* en el paso impar. Por tanto, en la demostración del Teorema 7.1, cada instrucción de la máquina de registros se simula en 6 pasos exactos del SN P. Según esto, la neurona l_h recibe *spikes* en un paso tal que $6k + 1, k \geq 0$ y también el nuevo módulo de FIN funciona convenientemente.

Otra extensión a esta forma normal es el caso de los sistemas SN P funcionando en modo aceptación. En el modo aceptación [IPY06], el sistema SN P obtiene una entrada en forma de intervalo entre dos *spikes* consecutivos enviados desde el exterior a la neurona de entrada i_0 . Por lo tanto es necesario un módulo *INPUT* que traduzca este valor de entrada en un número de *spikes* presentes en una neurona etiquetada como 1. Además, el sistema SN P resultante deberá comportarse determinísticamente. Ambas condiciones pueden ser satisfechas y el resultado y la prueba de ello se muestra a continuación.

Teorema 4.5.2 $DS_{\text{spik}_{2acc}}^{\beta} P_*(rule_2^*, cons_2, forg_2, dley_2, outd_2) = NRE$
donde (i) $\beta = h, \alpha = 1$, o (ii) $\beta = \underline{h}, \alpha = 2$ o (iii) se omite β y $\alpha = 1$

Demostración: Una vez más se parte de la demostración del teorema 7.1 publicado en [IPP⁺07]. Se puede observar que el módulo *SUB* es determinista. Por tanto, tan solo será necesario hacer determinista el módulo *ADD*. Para ello la regla $a \rightarrow a : 1$ de la neurona c_{i4} tendrá que ser eliminada.

Queda por construir el módulo *INPUT* que será el encargado de enviar el número de *spikes* correspondientes al valor de la entrada del módulo *ADD* que está conectado al registro 1. Sin embargo, puesto que el módulo *ADD*

Figura 4.8: Módulo *INPUT* con expresiones regulares simples

funciona de forma cíclica y síncrona cada 3 pasos, será necesario enviar *spikes* al registro 1 cada tres pasos computacionales o cualquier número múltiplo de 3. En cualquier otro caso los *spikes* serán consumidos en el módulo *ADD*.

El módulo *INPUT* que lleva a cabo estas tareas es el presentado en la figura 4.8. Tres pasos después de que la neurona i_0 reciba el primer *spike* de entrada, las neuronas $c_3 - c_6$ comenzarán a dispararse en cada paso. De forma análoga, tres pasos después de que la neurona i_0 reciba el segundo *spike* de entrada, las neuronas $c_3 - c_6$ dejarán de dispararse y eliminarán todos sus spikes. Por tanto, la neurona c_7 recibirá $3n$ spikes, donde n constituye el periodo entre el primer y el segundo *spike* de entrada. La neurona c_7 emite un *spike* en cada computación pero la neurona c_8 solo emite cada 3 pasos. Por tanto, la neurona 1 correspondiente al registro de entrada, cada 3 pasos recibe *spikes* y el número de *spikes* que recibe es exactamente n .

Por último, se observa que no queda ningún *spike* en las neuronas una vez finalice la computación. Al igual que ocurre con los registros *ADD* y *SUB* descritos en el teorema 7.1, el sistema se detiene siempre haciendo uso de la condición fuerte de parada. Si no fuese necesaria la condición estricta de parada habría que emitir las reglas $a^2 \rightarrow \lambda$ en las neuronas $c_3 - c_6$. Por tanto el parámetro *forg* se reduciría a 1.

El sistema SN P descrito anteriormente simula correctamente una máquina de registros funcionando en modo aceptación y la inclusión $NRE \subseteq DSpik_{2_{acc}}^\beta P_*(rule_2^*, cons_2, forg_\alpha, dley_2, outd_2)$ se sostiene. La inclusión opuesta sigue la tesis de Church-Turing.

4.6. Forma Normal sin retrasos y sin reglas de olvido

En esta sección se generalizan parcialmente los resultados de las secciones 4, 5, 6 y 7 del trabajo de Ibarra et al. [IPP⁺07]. Se demuestra que los sistemas SN P son universales cuando se eliminan los retrasos y las reglas de olvido a la vez. Además se mantiene simultáneamente restringido el grado sináptico de salida en el sistema a 2. Sin embargo, debido a la complejidad del diagrama, en algunos casos no es posible la representación gráfica completa del sistema resultante. Pero aunque gráficamente se complica eliminar estas características, tiene costo en otros parámetros ya que, diferencia de la sección 4 de Ibarra [IPP⁺07], el número máximo de reglas utilizadas dentro de una neurona se incrementa a 3 y se pierde la condición fuerte de parada.

Teorema 4.6.1 $Spik_2^h P_*(rule_3^*, cons_3, forg_0, dley_0, outd_2) = NRE$.

Demostración:

Siguiendo la tesis de Turing-Church, la inclusión $Spik_2^h P(rule_*, cons_*, forg_*, dley_*, outd_*) \subseteq NRE$ es válida. Para completar la demostración se debe probar que $NRE \subseteq Spik_2^h P(rule_3, cons_3, forg_0, dley_0, outd_2)$. Para ello, se construirá un sistema SN P que se dispara, únicamente dos veces en un intervalo, que corresponde con el número computado por la máquina de registros M simulada con el SN P. El sistema consiste en la simulación de las instrucciones ADD y SUB de la máquina M y el módulo de salida FIN que se encarga de los *spikes* finales del sistema. Como todas las reglas que se han utilizado poseen retraso 0, se escribirán de la manera más simple omitiendo el retraso, es decir: $E/a^r \rightarrow a$.

Cada registro r de la máquina M será asociado a un grupo de neuronas del sistema SN P, etiquetándolas como $d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8$ y s . A diferencia de otras demostraciones, el registro que contenga el valor n albergará $2n + 2$ *spikes* en su neurona central s . Cualquier registro que represente el valor 0 contendrá por tanto dos *spikes*. Esta modificación en la forma de representar los números, permitirá detectar correctamente si un registro contiene el valor 0 en el módulo SUB sin hacer uso de las reglas de olvido o los retrasos.

Simulando la instrucción ADD $l_i : (ADD(r), l_j, l_k)$ ver (figura 4.9)

Esta instrucción añade 1 al registro r y salta de forma no determinista a la etiqueta l_j o l_k . Este módulo se inicia cuando un *spike* entra en la neurona con etiqueta l_i . La neurona l_i envía un *spike* a las neuronas c_1 y c_2 . En el siguiente paso un *spike* proveniente de las neuronas anteriores alcanza la neurona s añadiendo 1 al contenido del registro. A la vez, el *spike* emitido por c_1 llega a c_3 y c_4 ; c_3 a su vez enviará otro *spike* a c_6 en el siguiente paso, a c_4 ; paralelamente el *spike* emitido por c_2 alcanza c_4 y c_5 . En ese momento la neurona c_4 puede decidir de forma no determinista si irá a l_j o l_k . Si c_4

Figura 4.9: Módulo *ADD* que simula la instrucción $l_i : (ADD(r), l_j, l_k)$

usa la regla $a^2 \rightarrow a$ entonces dos *spikes* quedarán bloqueados en c_8 (los que provienen de c_4 y c_5) y tan solo uno llegará a la neurona c_7 que esperará por un nuevo *spike*. En el siguiente paso, un *spike* proveniente de c_6 alcanza la neurona c_7 y se dispara activando un paso más tarde la neurona l_j .

Por otro lado, si c_4 utiliza la regla $a^2/a \rightarrow a$, consume únicamente uno de sus dos *spikes*. Esto implica que en el siguiente paso, c_7 recibe un *spike* y c_8 recibe dos (uno de c_4 y uno de c_5). Un paso más tarde, c_4 hace uso de su regla $a \rightarrow a$ y envía un *spike* a c_7 (que recibe otro de c_6 y por tanto no puede disparar) y otro a c_8 que ahora contiene tres *spikes* y se dispara, activando en el siguiente paso l_k .

El lector advertirá que después de cada instrucción *ADD*, las neuronas c_7 y c_8 contienen 3 y 2 *spikes* respectivamente, dependiendo de la regla seleccionada de forma no determinista en la neurona c_4 . Gracias a las expresiones regulares utilizadas en c_7 y c_8 no se producen interrupciones.

En esta demostración, las neuronas c_1 y c_2 tienen un grado sináptico de tres. Sin embargo, es trivial ver que podría ser reducido a dos situando dos neuronas más entre ellas y las neuronas c_3 , c_4 y c_5 , tal y como explica Ibarra en su sección 5 [IPP⁺07].

Simulando la instrucción $SUB\ l_i : (SUB(r), l_j, l_k)$ ver (figura 4.10)

El módulo se inicializa cuando llega un *spike* a la neurona l_i . Esta neurona se dispara y el *spike* generado alcanza d_0 y e_1 . Después, a partir de la neurona de entrada del registro $r(d_0)$, el *spike* alcanza d_1 , d_2 y s . Las reglas de la neurona nos permiten diferenciar si el registro está vacío o no. Tal y como hemos explicado previamente, almacenar el valor n significa contener $2n + 2$ spikes. Por tanto, cuando $r > 0$ (contendrá al menos 4 spikes) el *spike* proveniente de d_0 hace que se dispare la regla $aaa(aa)^+/a^3 \rightarrow a$ enviando un *spike* de d_4 a d_5 . Al mismo tiempo, otro *spike* que viene de d_2 alcanza d_5 y esto hace que no pueda ser disparada. En paralelo, se envía un *spike* de d_1 a d_3 y en el siguiente paso, llegará a d_4 . Esta neurona se disparará porque ya contiene un *spike* haciendo posible alcanzar por un lado l_j .

Por otro lado, cuando r almacena un 0 (s contiene dos spikes) el *spike* recibido por d_0 dispara la regla $a^3/a^2 \rightarrow a$. Después se dispara la neurona s , consumiendo dos de los tres *spikes* que tiene disponibles. Por tanto, el *spike* producido se envía a las neuronas d_4 y d_5 . Otro *spike* desde d_2 alcanza simultáneamente d_5 , mientras que d_3 recibe el *spike* que proviene de d_1 . En el siguiente paso, s se dispara otra vez (usa la regla $a \rightarrow a$) consumiendo su último *spike* y enviando un nuevo *spike* a d_4 y d_5 (ahora el valor del registro está degradado y necesita ser reconstruido). Este *spike* alcanza la neurona d_4 simultáneamente con otra procedente de d_3 . Entonces d_4 no puede ser disparada ya que contiene tres spikes; mientras tanto d_5 recibe un *spike* desde s pasando por tanto d_5 a tener 3 spikes. Esto provoca que se dispare

Figura 4.10: Módulo SUB que simula la instrucción $l_i : (SUB(r), l_j, l_k)$ junto con el registro r

Figura 4.11: Módulo *AND*

d_5 , permitiendo a su vez disparar las neuronas d_6 y d_8 en el siguiente paso. Cada una de las dos neuronas anteriores emite después un *spike* a s que reconstituye el valor 0 en el registro antes de alcanzar el estado l_k .

Se puede comprobar que los *spikes* que permanecen en las neuronas d_4 y d_5 no interrumpen computaciones futuras. En este caso, el grado de salida puede también ser reducido a 2, usando una pareja de neuronas intermedias entre $l_i - d_1$ y $r - d_2$.

El módulo *AND* detallado en la figura 4.11, conectado a d_7 y d_8 al igual que las neuronas de e_1 a e_5 , se usa para evitar un funcionamiento incorrecto del sistema en el caso de que diferentes instrucciones *SUB* funcionen sobre el mismo registro. Actualmente, en esta situación, todos los módulos *SUB* podrían recibir los *spikes* disparados por las neuronas d_7 o d_8 del registro. Se puede verificar también que el módulo *AND* emite un *spike* de salida si y solo si recibe dos *spikes* de entrada simultáneos. De esta forma, los módulos *AND* incluidos en la figura 4.10 permiten que un *spike* final alcance las neuronas l_j o l_k solo para el módulo *SUB* cuya neurona l_i haya recibido un *spike* 5 pasos antes.

Fin de la computación *FIN* módulo de *FIN* ver figura (4.12)

Figura 4.12: Módulo *FIN*

Cuando la computación se detiene en M , un *spike* alcanza la neurona l_h en el sistema SN P. En ese momento, el registro 1 de la máquina M almacena el valor n y en este caso la neurona 1 contiene $2n+2$ spikes. El *spike* emitido por l_h alcanza la neurona 1 y por tanto contendrá un número par de spikes. Esto provoca que la neurona 1 se dispare de forma continua, consumiendo 2 *spikes* en cada paso. Un paso después de recibir un *spike* de l_h , la neurona 1 dispara un *spike* que alcanza f_1 y la neurona *out*. Seguidamente, la neurona *out* se dispara y emite *spikes* por primera vez. A partir de este paso, la neurona *out* de forma simultánea recibe una pareja de *spikes* desde la neurona 1 y f_1 que no le permiten disparar otra vez hasta que la neurona 1 se dispare por última vez. Cuando la neurona 1 deja de disparar, la neurona *out* todavía recibirá un *spike* de f_1 haciendo que esta se dispare emitiendo su segundo y último spike, es decir, exactamente n pasos después de su primer disparo.

Fin de la computación *FIN* módulo de *FIN* ver figura (4.12)

Una vez ha finalizado la computación, la neurona 1 contendrá un *spike* y la neurona *out* contendrá $2n - 1$. Puesto que este diseño deja *spikes* en el sistema después de que este se pare, no puede ser extendido para cumplir con la condición fuerte de parada.

El módulo *ADD* incluido en la figura 4.9 puede ser visto como una exten-

sión no determinista análoga a la determinista propuesta por Ionescu [IPY06]. Las únicas diferencias son: Ionescu usa $2n$ *spikes* frente a las $2n + 2$ para representar el valor entero n y es necesario usar dos *spikes* en lugar de uno en las neuronas asociadas con las etiquetas que identifican las instrucciones de la máquina de registros. Esto significa que el no determinismo puede ser evitado en la demostración presentada mientras se mantiene la universalidad de la máquina de registros. Este factor será tenido en cuenta en la demostración del siguiente teorema.

Teorema 4.6.2 $DSpik_{2,acc}^h P_*(rule_3, cons_3, forg_0, dley_0, outd_2) = NRE$.

Demostración: Tal y como se mencionó anteriormente, el módulo *ADD* puede fácilmente ser transformado en determinista. Le falta incluir un módulo especial de entrada *INPUT* que traduzca la entrada en un intervalo de tiempo entre dos *spikes* consecutivos al número de $2n + 2$ *spikes* presentes en la neurona etiquetada como 1. Este módulo de entrada (*INPUT*) diseñado por Ibarra [IPP⁺07] se puede aplicar con un ligero cambio que consiste en que la neurona 1 sea iniciada con 2 *spikes*.

Capítulo 5

Análisis de la Eficiencia

Computacional de los sistemas P con Proteínas en las Membranas

5.1. Introducción a los sistemas P con proteínas en las membranas

En el capítulo 1.6.5 se han introducido brevemente las características de esta variante de sistemas P considerada como una extensión al modelo básico. A lo largo de este capítulo se presentan los resultados obtenidos tras haber analizado su poder computacional. El análisis está centrado en estudiar la eficiencia computacional a la hora de tratar con problemas del **ESPACIO-P**.

El análisis de cómo tiene lugar el paralelismo en los sistemas P que no incluyen proteínas ha sido visto como poco realista por algunos autores que sugieren que la forma en la que se procesan las diferentes especies de moléculas dentro de la membrana no es cercana a la realidad. Con una cierta

motivación por acercar a la realidad biológica el mecanismo de procesamiento de las moléculas surge este modelo de sistema P con proteínas. Este nuevo modelo en cierta forma limita el paralelismo modelando lo que en la naturaleza biológica se conoce como transporte transmembranal.

Otra motivación detrás de este nuevo tipo de sistemas deriva del estudio de Luca Cardelli [Car05]. Cardelli sugiere que muchas de las reglas a nivel celular operan a nivel de membrana y no a nivel de la región que encierran. En el trabajo de Cardelli conocido como *brane calculi* tan solo es necesario utilizar objetos denominados proteínas dispuestas en las membranas, mientras que la evolución está basada en operaciones realizadas sobre las membranas como son: exocitosis, fagocitosis, etc.

En el área de la computación celular con membranas, como se ha visto en el capítulo 1, existen reglas asociadas a cada región definida por la membrana; en los últimos años las reglas definidas para la membrana trabajan fundamentalmente con objetos en lugar de con cadenas de objetos. Los sistemas celulares con proteínas en las membranas poseen dos tipos de reglas, las que gobiernan la evolución a nivel de la región definida por la membrana y las controladas por una proteína. La razón de considerar estos dos tipos de reglas en el sistema es porque en biología muchas de las reacciones que tienen lugar en los compartimentos de las células están controladas/catalizadas por una proteína encerrada entre la membrana de doble capa.

Existen varios tipos de proteínas embebidas en las membranas de una célula, que de forma simple podemos agrupar en dos tipos: las proteínas integrales y las proteínas periféricas. Las primeras operan tanto dentro como fuera de la membrana, mientras que las segundas únicamente operan en el interior de la célula.

A lo largo de este capítulo se demuestra que los sistemas P con proteí-

nas pueden resolver en tiempo polinómico los problemas de tipo **PSPACE**, expresado matemáticamente de la siguiente forma:

$$M - PTIME = M - NPTIME = \mathbf{PSPACE}$$

Donde, $M - (N)PTIME$ define el tipo de problemas que puede resolver una máquina M no determinista en tiempo polinómico; en este caso la máquina M será un sistema P con proteínas en sus membranas. Esta relación es conocida como la *Tesis de Computación Paralela*, donde los dispositivos de cómputo con esta propiedad se conocen como la *segunda clase de máquinas*. Otros miembros de esta clase son las máquinas de Turing alternantes, SIMDAG también conocidas como SIMD PRAM y otros modelos de computación paralela [vEB91].

En los sistemas P estudiados en este capítulo, además de las definiciones básicas de un sistema P detalladas en el capítulo 1, se incluyen las definiciones de la sección 1.6.5. En esta variante de los sistemas P se utilizan dos tipos de objetos: las proteínas y los objetos que utilizan estos sistemas de forma habitual. Las proteínas forman parte de la propia membrana mientras que el resto de objetos forman parte de la propia región delimitada por la membrana. Para identificar este hecho, para una proteína p alojada en una membrana etiquetada con i escribiremos $[_ip]$. Tanto la región como la propia membrana pueden contener multiobjetos y proteínas.

La expresión, $[_1p_1^2p_2|a^3bd[_2p_3|b^5cd^2]_2]_1$ indica que tenemos una estructura de dos membranas etiquetadas con 1 y 2. La membrana interna 2 contiene un multiconjunto de objetos b^5cd en su interior, además de una proteína p_3 . La neurona 1 por su parte posee los objetos a^3bd y el multiconjunto de proteínas $p_1^2p_2$.

A continuación se detallan los tipos de reglas usados por esta clase de sistemas P para la interacción entre proteínas y objetos. En todas ellas los

elementos a, b, c, d son objetos y los elementos etiquetados como p son proteínas. Además se distinguen dos tipos de reglas: las restrictivas que se identifican con *res* y las de cambio de proteína que se identifican como *cp*.

Tipo	Regla	Efecto
1res	$[ip a]_i \rightarrow [ip b]_i$ $a[ip]_i \rightarrow b[ip]_i$	modifica un objeto, pero no lo mueve
2res	$[ip a]_i \rightarrow a[ip]_i$ $a[ip]_i \rightarrow [ip a]_i$	mueve un objeto sin modificarlo
3res	$[ip a]_i \rightarrow b[ip]_i$ $a[ip]_i \rightarrow [ip b]_i$	modifica y mueve un objeto
4res	$a[ip b]_i \rightarrow b[ip a]_i$	intercambia dos objetos
5res	$a[ip b]_i \rightarrow c[ip d]_i$	intercambia y modifica dos objetos

En todos los casos anteriores las proteínas no cambian, pudiéndose decir que tienen el papel de catalizadoras. En la tabla mostrada a continuación se presenta el segundo tipo de reglas que implican un cambio de proteínas.

Tipo	Regla	Efecto (además del cambio correspondiente de proteína)
1cp	$[ip a] \rightarrow [ip' b]_i$ $a[ip] \rightarrow b[ip']_i$	modifica un objeto, pero no lo mueve
2cp	$[ip a] \rightarrow a[ip']_i$ $a[ip] \rightarrow [ip' a]_i$	mueve un objeto sin modificarlo
3cp	$[ip a] \rightarrow b[ip']_i$ $a[ip] \rightarrow [ip' b]_i$	modifica y mueve un objeto
4cp	$a[ip b] \rightarrow b[ip' a]_i$	intercambia dos objetos
5cp	$a[ip b] \rightarrow c[ip' d]_i$	intercambia y modifica dos objetos

Otro caso que podría ser intermedio a los dos anteriores sería el cambio de proteínas, pero de manera restrictiva, permitiendo como mucho dos estados para cada proteína $p, \bar{p}$ y las reglas de la primera tabla sin cambio de proteína o cambiando p a $\bar{p}$ y volviendo a p como ocurre en la catálisis de los biestables. Estas reglas de tipo proteínas flip-flop se denotan como *nff*, $n = 1, 2, 3, 4, 5$. En este caso se permiten ambos tipos de reglas, en las que las proteínas no cambian y en las que cambian de p a $\bar{p}$ y vuelven al estado inicial.

Tanto en el caso de las reglas de tipo ff como en las de tipo cp (cambio de proteína) se puede hacer que las proteínas se muevan a su estado complementario de p a $\bar{p}$ y viceversa. Este tipo de reglas se dice que son de tipo *pure ff* o *pure cp* y se indica anotándolo de la siguiente forma: ffp y ccp respectivamente.

Para dividir la membrana se utilizarán reglas como la mostrada a continuación, donde p, p', p'' son proteínas: $[ip]_i \rightarrow [ip']_i[ip'']_i$.

La membrana i no puede ser la elemental. La regla no cambia la membrana etiquetada con i y en su lugar en el siguiente paso una membrana tendrá dos membranas con la misma etiqueta i y los mismos contenidos que la original, es decir, objetos y otras membranas.

Definición Un sistema P con proteínas en las membranas y división de membranas es un sistema de la forma $\Pi = (O, P, \mu, w_1/z_1, \dots, w_m/z_m, E, R_1, \dots, R_m, i_0)$, donde:

- m es el grado del sistema (número de membranas).
- O es el conjunto de objetos y P es el conjunto de proteínas tal que $(O \cap P = \emptyset)$.
- μ define la estructura de membranas.
- $w_1, \dots, w_m$ son las cadenas que representan los multiconjuntos de objetos presentes en las m regiones de la estructura de membranas.
- $z_1, \dots, z_m$ son los multiconjuntos de proteínas presentes en las m membranas de μ .
- $E \subseteq O$ es el conjunto de objetos presente en el entorno, de forma arbitraria y con un elevado número de copias.

- $R_1, \dots, R_m$ son los conjuntos finitos de reglas asociadas con la membrana m en μ .
- i_0 es la etiqueta que identifica la membrana de salida.

Las reglas se usan de forma no determinista y masivamente paralela. En cada paso, el máximo número de reglas que se puedan usar se usarán, esto es, hasta que no se pueda aplicar ninguna regla a los objetos y proteínas que no hayan sido utilizados dentro del multiconjunto. En cada paso se establece la condición de que cada objeto y cada proteína puede evolucionar tras la aplicación de como mucho una regla, pero la membrana no se ve afectada por la aplicación de estas reglas a excepción de la regla de división de la membrana, por lo que la misma membrana puede aparecer en cualquier número de reglas del tipo 1-5 a la vez. La condición de parada del sistema se entiende como una secuencia de configuraciones tras la cual el sistema se para, es decir, no existe ninguna regla que pueda ser aplicada considerando los objetos y proteínas presentes en el sistema. Cuando se detiene la computación se toma como resultado los múltiples objetos presentes en la región i_0 . Se denota como $N(\Pi)$ al conjunto de números computados de esta forma por el sistema Π . También es habitual denotar como $NOP_m(\text{pro}_r, \text{list} - \text{of} - \text{types} - \text{of} - \text{rules})$ a la familia de conjuntos $N(\Pi)$ generados por los sistemas Π con al menos m membranas, que usan reglas de acuerdo con las especificadas en $\text{list} - \text{of} - \text{types} - \text{of} - \text{rules}$ y con como mucho r proteínas presentes en una membrana. En caso de no limitar m o r se utilizará el subíndice $*$ para denotarlo.

Ejemplo: Se considera el siguiente sistema P

$$\Pi = (O, P, \mu, w_0/z_0, w_1/z_1, E, R_0, R_1, i_0) \quad (5.1)$$

con los componentes:

$$\begin{aligned}
 O &= \{a_1, \dots, a_n\}, \\
 P &= \{p, q\}, \\
 \mu &= [0[1]1]_0, \\
 w_0 &= z_0 = E = \emptyset \\
 w_1 &= a_1, \\
 z_1 &= \{p\}, \\
 R_1 &= \emptyset, \\
 R_2 &= \{[{}_ip]_1 \rightarrow [{}_1q]_1[{}_1q]_1, [{}_1q|a_n]1 \rightarrow a_n[{}_1q]_1\} \\
 &\quad \cup \{[{}_1q|a_i]_1 \rightarrow [{}_1q|a_{i+1}]_1 | 1 \leq i \leq n-1\}, \\
 i_0 &= 0
 \end{aligned}$$

Según la configuración inicial, el sistema contiene dos membranas y un objeto. En cada ejecución impar todas las membranas etiquetadas como 1 se dividen y las proteínas de sus membranas se cambian de p a q . En cada ejecución par las proteínas cambian de q a p y los objetos a_i en las membranas evolucionan de a_{i+1} , para $1 \leq i \leq n-1$. Por tanto, cada dos pasos de ejecución se dobla el número de membranas etiquetadas como 1. En el paso $2n$ los objetos a_n son expulsados a la membrana etiquetada como 0, que es la membrana de salida y el sistema se detiene. La computación del sistema se ilustra en la figura 5.1. Por tanto, se puede escribir que $N(\Pi) = \{2^n | n \in \mathbb{N}\}$

Hay varios resultados previos que demuestran la universalidad de esta variante de sistemas P [PP06b] [PP06a]; de todos ellos destacan únicamente dos:

Figura 5.1: Un ejemplo de sistema P con proteínas en las membranas

$$NOP_1(pro_2; 2cpp) = NRE,$$

$$NOP_1(pro_*; 3ffp) = NRE.$$

Donde NRE es el conjunto de enteros no negativos recursivamente enumerable. Sin embargo, este estudio se centra en los sistemas P funcionando en modo aceptación que son capaces de resolver problemas de decisión.

5.2. Familias de sistemas P y su capacidad para resolver problemas de decisión

La mayoría de los sistemas P son universales, es decir, es posible construir una máquina universal capaz de resolver cualquier problema computable por una máquina de Turing. Sin embargo, cuando hacemos uso del paralelismo masivo de los sistemas P a la hora de tratar de buscar soluciones eficientes a problemas intratables, el concepto de un sistema P resolviendo todas las instancias del problema se convierte en algo más restrictivo. El uso eficiente del paralelismo debe ser restringido por la estructura particular del sistema

P. Por ejemplo, la profundidad de la estructura se mantiene fija durante las computaciones de la mayoría de los sistemas P; sin embargo, para una solución paralela con varias instancias, son necesarias varias profundidades en la estructura de membranas del sistema.

Por tanto, es frecuente el uso de familias de sistemas P en lugar de usar un único sistema P. A continuación se formaliza el concepto de *sistema P reconocedor* utilizado para resolver problemas de decisión y que debería cumplir con los siguientes requisitos:

- el alfabeto de trabajo contendrá únicamente dos elementos distinguibles *yes* y *no*;
- todas las computaciones han de pararse;
- en caso de que la computación sea aceptada un único objeto *yes* deberá ser enviado a la región de salida del sistema y un *no* en caso de rechazo de la computación. Este hecho solo puede producirse en el último paso computacional.

Dado el problema de decisión $X = (I_X, \theta_X)$ donde I_X es un lenguaje definido sobre un alfabeto cuyos elementos denominamos instancias y θ_X es una función booleana sobre I_X .

Definición Una familia $\Pi = \{\Pi(w) : w \in I_X\}$ de sistemas P reconocedor sin membrana de entrada es polinómicamente uniforme por una máquina de Turing si existe una máquina de Turing determinista trabajando en tiempo polinómico que construya el sistema $\Pi(w)$ a partir de la instancia $w \in I_X$.

Los resultados presentados en los siguientes apartados de esta sección tratan con sistemas reconocedores sin membrana de entrada, es decir, una instancia w del problema X está codificada en la estructura del sistema P

$\Pi(w)$. Se supone que el sistema $\Pi(w)$ resuelve una instancia w . Formalmente, define las condiciones de robustez y completitud de Π con respecto a X . Una conjunción de esas dos condiciones asegura que para cada $w \in I_X$ si $\theta_X(w) = 1$, entonces cada computación de $\Pi(w)$ es aceptada, y si $\theta_X(w) = 0$ entonces cada computación de $\Pi(w)$ es rechazada [PJ10].

Es importante destacar que el sistema $\Pi(w)$ puede ser generado de manera no determinista, es decir podría haber diferentes computaciones posibles, pero con el mismo resultado. A este tipo de sistemas se les llama *confluentes*.

Definición Un problema de decisión X es resoluble en tiempo polinómico por una familia de sistemas P reconocedores y sin membrana de salida pertenecientes a la clase $\mathcal{R}$ $\Pi = \{\Pi(w) : w \in I_X\}$, denotado por $X \in \mathbf{PMC}_{\mathcal{R}}^*$ si cumple lo siguiente:

- La familia Π es polinómica uniforme por las máquinas de Turing.
- La familia Π es polinómicamente limitada, esto es, existe un número natural $k \in \mathbb{N}$ que para cada instancia $w \in I_X$ cada computación de $\Pi(w)$ lleva a cabo como mucho $|w|^k$ pasos computacionales.
- La familia Π es robusta y completa con respecto a X .

La familia Π se dice que proporciona una solución semi-uniforme al problema X . En este caso, para cada instancia de X se tendrá un sistema P especial. Formalmente se denotará por $\mathcal{MP}$ a la clase de sistemas P con proteínas en las membranas. La siguiente relación ha sido demostrada en [PP06a] para los sistemas P con proteínas en las membranas:

$$\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{MP}}^*$$

5.3. Resolución del problema QSAT en tiempo lineal

En esta sección se demuestra que los sistemas P con proteínas en las membranas pueden resolver el problema QSAT perteneciente a la clase *ESPACIO-P*-completo en tiempo lineal. De forma más precisa, existe una familia semi-uniforme de sistemas P con proteínas tal que para cada instancia del QSAT, un sistema P apropiado para resolver una instancia en tiempo lineal puede ser construido en tiempo polinómico con respecto al tamaño de la instancia. También se observa una interesante compensación entre el uso de ciertas operaciones elementales de los sistemas P.

El problema QSAT(satisfiability of quantified propositional formulas) es un problema estándar perteneciente al tipo *ESPACIO-P*-completo. Pregunta si sí, o si no una fórmula booleana cuantificada en forma normal conjuntiva asume el valor *true*. La fórmula sería de la forma:

$$\gamma = Q_1x_1Q_2x_2\dots Q_nx_n(C_1 \wedge C_2 \wedge \dots \wedge C_m), \quad (5.2)$$

donde cada Q_i , $1 \leq i \leq n$ es cualquier $\forall$ o $\exists$, y cada C_j , $1 \leq j \leq m$ es una cláusula en forma de disyunción:

$$C_j = y_1 \vee y_2 \vee \dots \vee y_r, \quad (5.3)$$

con cada y_k siendo una variable proposicional o su negación. Por ejemplo, si se considera la fórmula proposicional:

$$\beta = Q_1x_1Q_2x_2[(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)], \quad (5.4)$$

resulta sencillo ver que será *true* cuando $Q_1 = \forall$ y $Q_2 = \exists$ pero será *false* cuando $Q_1 = \exists$ y $Q_2 = \exists$

Teorema 5.3.1 $\text{PSPACE} \subseteq \text{PMC}_{\mathcal{MP}}^*$

Demostración Se considera la fórmula proposicional γ de acuerdo a 5.2 con

$$C_i = y_{i,1}, \vee \dots \vee y_{i,p_i},$$

para algún $p_i \geq 1$ y $y_{i,p_i} \in \{x_k, \neg x_k | 1 \leq k \leq n\}$, para cada $1 \leq i \leq m$, $1 \leq j \leq p_i$

Se construye el sistema P

$$\Pi = (O, P, \mu, w_0/z_0, w_1/z_1, \dots, w_{n+1}/z_{n+2}, \emptyset, R_0, R_1, \dots R_{n+2}, 0)$$

con los componentes

$$O = \{a_i, t_i, f_i | 1 \leq i \leq n\} \cup \{r_i, \bar{r}_i | 1 \leq i \leq m\} \{c_i | 1 \leq i \leq 3m + 5m\} \\ \cup \{t, s, yes, no\},$$

$$P = \{p_0, p_+, p_-, p_x\},$$

$$\mu = [0[1 \dots [n[n+1]_{n+1}[n+2]_{n+2}]_{n \dots}]_1]_0,$$

$$w_0 = c_1, w_1 = a_1 no,$$

$$w_i = a_i, \text{ para cada } i = 2, \dots, n,$$

$$w_{n+1} = r_1 r_2 \dots r_m,$$

$$w_n + 2 = \lambda,$$

$$z_0 = p_0, z_1 = p_x,$$

$$z_i = p_0, \text{ para todo } i = 2, \dots, n + 2.$$

Las reglas contenidas en los conjuntos R_i se definen a continuación:

Para R_i , $1 \leq i \leq n$:

$$[ip_x]i \rightarrow [ip_+][ip_-]_i, [ip_+|a_i]_i \rightarrow [ip_+|t_i]_i, [ip_-|a_i]_i \rightarrow [ip_-|f_i]_i \quad (5.5)$$

Para R_i , $1 \leq i \leq n - 1$:

$$t_i[i + 1p_0]_{i+1} \rightarrow [i+1p_x|t_i]_{i+1}, f_i[i+1p_0]_{i+1} \rightarrow [i+1p_x|f_i]_{i+1} \quad (5.6)$$

Para R_i , $3 \leq i \leq n$:

$$t_j[ip_0] \rightarrow [ip_0|t_j]_i, f_j[ip_0]_i \rightarrow [ip_0|f_j]_i \text{ para todo } j \text{ } 1 \leq j \leq j - 2 \quad (5.7)$$

Para R_{n+1} :

$$t_i[n+1p_0|r_j]_{n+1} \rightarrow r_j[n+1p_0|t_i]_{n+1}, [n+1p_0|t_i]_{n+1} \rightarrow t_i[n+1p_0]_{n+1} \quad (5.8)$$

para todo $i \leq i \leq n$ y $j, 1 \leq j \leq m$ tal que C_i contenga x_i

$$f_i[n+1p_0|r_j]_{n+1} \rightarrow r_j[n+1p_0|f_i]_{n+1}, [n+1p_0|f_i]_{n+1} \rightarrow f_i[n+1p_0]_{n+1} \quad (5.9)$$

para todo $i \leq i \leq n$ y $j, 1 \leq j \leq m$ tal que C_j contenga $\neg x_i$

Para R_{n+2} :

$$r_1[n+2p_0]_{n+2} \rightarrow [n+2p_0|\bar{r}_1]_{n+2} \quad (5.10)$$

$$r_{i+1}[n+2p_0|\bar{r}_i]_{n+2} \rightarrow \bar{r}_i[n+2p_0|\bar{r}_{i+1}]_{n+2} \text{ para todo } i, \text{ } 1 \leq i \leq n - 1 \quad (5.11)$$

$$[n+2p_0|\bar{r}_m]_{n+2} \rightarrow t[n+2p_0]_{n+2} \quad (5.12)$$

Para R_i , $1 \leq i \leq n$ tal que $Q_i = \forall$:

$$[{}_i p_- | t]_i \rightarrow s[{}_i p_- |]_i, \quad s[{}_i p_+ | t]_i \rightarrow t[{}_i p_+ | s]_i \quad (5.13)$$

Para R_i , $1 \leq i \leq n$ tal que $Q_i = \exists$:

$$[{}_i p_- | t]_i \rightarrow t[{}_i p_- |]_i, \quad [{}_i p_+ | t]_i \rightarrow t[{}_i p_+ |]_i \quad (5.14)$$

Para R_1 :

$$t[{}_1 p_- | no]_1 \rightarrow s[{}_1 p_- | yes]_1 \quad (5.15)$$

Para R_0 :

$$[{}_0 p_0 | c_i]_0 \rightarrow [{}_0 p_0 | c_{i+1}]_0 \quad \text{para todo } i, \quad 1 \leq i \leq 3m + 5n - 1 \quad (5.16)$$

Para R_1 :

$$\begin{aligned} c_{3m+5n}[{}_1 p_- | yes]_1 &\rightarrow yes[{}_1 p_- | c_{3m+5n}]_1, \\ c_{3m+5n}[{}_1 p_- | no]_1 &\rightarrow no[{}_1 p_- | c_{3m+5n}]_1 \end{aligned} \quad (5.17)$$

Figura 5.2: Expansión de la membrana inicial a un árbol binario. Los símbolos en los nodos indican las proteínas presentes en las membranas.

Resulta sencillo verificar que el tamaño del sistema P $\Pi(\gamma)$ (el número de membranas, reglas, el tamaño de la configuración inicial, etc.) es $\mathcal{O}(nm)$, siendo n el número de variables y m el número de cláusula. Además el sistema puede ser construido en tiempo polinómico por una máquina determinista de Turing cuya entrada es la formula γ .

La fase inicial de computación del sistema P Π se ilustra en la figura 5.2. En el primer paso la membrana no elemental, usando la primera regla (5.5) en el nivel 1, se divide en dos partes con diferentes proteínas en las membranas. En el siguiente paso, se producen los símbolos f_1 y t_1 en las dos membranas resultantes (ver las reglas después de (5.5)). En el tercer paso computacional los símbolos anteriores son movidos un nivel inferior a las membranas etiquetadas como 2 (ver (5.6)). Las proteínas en esa membrana se cambian a p_x .

Este ciclo se repite n veces y propaga las reglas (5.5) resultantes de la división por el árbol de membranas hacia las hojas. De manera simultánea, los símbolos producidos t_i y f_i se mueven hacia las hojas del árbol gracias a las reglas definidas en (5.7). Esta fase finaliza después de $3n - 1$ pasos cuando la estructura de membranas forma un árbol binario equilibrado, tal y como ilustra la figura 5.2. Cada uno de sus 2^n nodos en el nivel n contiene un conjunto de objetos $\{x_1, x_2, \dots, x_n\}$, donde $x_i \in \{f_i, t_i\}$, $1 \leq i \leq n$ a fin de que todas las n -tuplas estén presentes.

La segunda fase consiste en verificar si la fórmula sin los cuantificadores es satisfecha por las n -tuplas de valores lógicos $(x_1, x_2, \dots, x_n)$. Esta verificación se hace para las n -tuplas en paralelo. Comienza trasladando aquellos objetos r_i , $1 \leq i \leq m$, correspondientes a la clase C_i y que han sido satisfechos por una n -tupla desde la membrana $[_{n+1}]_{n+1}$ hacia $[_n]_n$. Las reglas (5.10)-(5.11) son las responsables de llevar a cabo este proceso. Cada vez que los

objetos $r_1, \dots, r_n$ aparezcan en la membrana $[n]_n$, comienza otro proceso cuyo propósito es verificar si están presentes todos $r_i, 1 \leq i \leq m$. Esto es llevado a cabo haciendo uso de los movimientos desde y hacia la membrana $[n+2]_{n+2}$ empleando las reglas (5.12)-(5.14). Finalmente, cuando todos los r_i están presentes, lo cual quiere decir que todas las cláusulas son satisfechas por $(x_1, x_2, \dots, x_n)$ correspondientes a una membrana $[n]_n$ en particular, un objeto t es liberado a la membrana $[n]_n$.

Las reglas que se aplican durante la segunda fase pueden solaparse parcialmente con las de la fase inicial: cada vez que el primer objeto t_i o f_i llegue a la membrana $[n]_n$, da comienzo la segunda fase, mientras que el resto de t_i y f_i lleguen más tarde. Sin embargo, la aplicación de las reglas en la segunda fase y descritas anteriormente, no son alternas.

Por último, durante la tercera fase de la computación se verifica si la fórmula completa, incluyendo cuantificadores, es satisfecha. Los objetos t ascienden por la estructura de membranas en árbol, verificando en cada nivel un cuantificador $\forall$ o $\exists$. Es importante hacer notar que las reglas (5.15)-(5.16) permiten la existencia de más de un símbolo t por membrana (en el caso de $\exists$) el cual sin embargo no altera la computación. Finalmente, el objeto t aparece en la membrana 0, indicando que la fórmula ha sido satisfecha. En este caso, las reglas (5.17) garantizan que el objeto *no* se elimina de la membrana etiquetada como 1 mediante la proteína p_- y es reemplazada por el objeto *yes*.

La primera fase de la computación descrita anteriormente finaliza en $3n-1$ pasos, la segunda fase lleva un total de $3m$ pasos y la tercera fase un total de $2n$, por tanto, en total son $5n + 3m$ pasos. De forma simultánea con las fases 1-3, los símbolos $c_i, 1 \leq i \leq 5n + 3m$ junto con las reglas (5.17)-(5.16) en la membrana $[0]_0$ implementan un temporizador el cual, después de $5n + 3m$

pasos, mueve el símbolo *yes* o *no* que está presente en la membrana 1 con la proteína p_- a la membrana de salida 0 y la computación se detiene. El objeto *yes* o *no* está presente si y solo si la fórmula γ se evalúa a *verdadero* o *falso*, por tanto el sistema es robusto y completo. Como se demuestra anteriormente, el sistema siempre se para después de $(O)(n + m)$ pasos y por tanto está polinómicamente limitado.

La demostración anterior está basada en una técnica empleada por [Sos03] al tratar los sistemas P con membranas activas. Sin embargo, puesto que la función de la membrana con proteínas es diferente, la técnica ha sido ligeramente adaptada. En los sistemas P con membranas activas la división es llevada a cabo tanto por los contenidos de la membrana como por la polarización, mientras que el caso de las proteínas es controlado únicamente por la proteína que forma parte de la membrana. En el caso descrito en Sosik [Sos03] la estructura de membranas se divide siguiendo una estrategia *bottom-up* mientras que en el caso descrito aquí se emplea una estrategia *top-down*.

Se puede observar también que las reglas (5.6) son las únicas reglas de tipo $2cp$; el resto son de tipo restrictivo o de división. Además, esas reglas $2cp$ son las que se utilizan para controlar el proceso de división de las membranas. Por tanto, la división de las membranas es controlada únicamente por la presencia de una proteína específica en la membrana. Supongamos que se introducen reglas similares a las usadas con los sistemas de membranas activas, es decir: $[ip|a]_i \rightarrow [ip|b]_i[ip|c]_i$ que usan un cierto objeto para controlar el proceso. Entonces las reglas (5.6) no serían necesarias y todo el sistema P podría utilizar únicamente reglas de tipo restringido y de división.

Por tanto, resulta que la única necesidad de incluir proteínas en las membranas es el control de la división de la membrana. Si se compara la demos-

tración con la de Sosik [Sos03] se observa que el rol jugado por la polarización de membranas, que en cierto modo es una generalización de las proteínas en las membranas, en la demostración anterior se sustituye por el uso de las reglas antiport. Por tanto, existe un cierto equilibrio entre los sistemas P con polarización y proteínas y los sistemas P con reglas antiport.

Desde el punto de vista de la eficiencia no existe diferencia sustancial entre el uso de reglas restringidas y de cambio de proteína. Tal y como se demuestra en [PP06a] la universalidad puede ser alcanzada únicamente con reglas restringidas. Sin embargo, existe otra cierta relación entre el número de membranas y el uso de las reglas de cambio de proteínas.

5.4. Simulación de un Sistema P con proteínas en las Membranas en Espacio Polinómico

En este apartado se demuestra la relación inversa dada en el teorema 4.2.1; para ello se usa un algoritmo para simular un sistema P con proteínas. El sistema P simulado es confluyente (posiblemente no determinista); por tanto las condiciones de la Tesis de Computación paralela se satisfacen. Sin embargo, la simulación por si misma es determinista; en cada paso sólo se simulan las reglas aplicadas sobre un multiconjunto previamente elegido, lo que implica que solo se simule una posible secuencia de configuraciones de sistemas P. El algoritmo que selecciona las reglas que son aplicadas implica utilizar una prioridad débil entre las reglas: (i) prioridad «bottom-up» entre las reglas asociadas a diferentes membranas; (ii) prioridad entre reglas en la misma membrana dado por el orden en el que están listadas, entre 1-6. La condición de confluencia garantiza que la simulación produzca siempre el resultado correcto.

Se emplea la técnica de simulación de tiempo-reverso que proviene de la teoría de complejidad cuando trata con el segundo tipo de máquinas. En lugar de simular una computación del sistema P desde su configuración inicial hacia adelante, lo que requeriría un espacio exponencial para almacenar configuraciones, se crea una función recursiva **State** que devuelve el estado a cualquier membrana h después de un número de pasos dados. La llamada recursiva evalúa contenidos de las membranas que interaccionan con h en orden inverso, es decir, hacia la condición inicial. Es clave observar que en el paso previo de computación el estado de la membrana se determina por su propio estado, los estados de las membranas embebidas y el estado de la membrana padre. De esta manera no es necesario almacenar un estado de la membrana; en su lugar se calcula recursivamente siempre que sea necesario. La profundidad de las llamadas recursivas es proporcional al número de pasos simulados por un sistema P. Además, a cada nivel de la llamada a la pila, se debe almacenar un estado de una membrana simple que puede ser realizado en espacio polinómico. De esta forma un resultado de cualquier computación $\mathcal{T}(n)$ – *tiempo – limitado* de un sistema P reconocedor con proteínas en las membranas puede ser encontrado en el espacio polinómico $\mathcal{T}(n)$.

Teorema 5.4.1 $\text{PMC}^*_{\mathcal{MP}} \subseteq \text{PSPACE}$

Si ponemos conjuntamente el teorema 4.2.1 y el 4.2.2 se obtiene la tesis de computación paralela para las familias semi-uniformes de los sistemas P con proteínas en las membranas confluyentes:

Corolario 5.4.2 $\text{PMC}^*_{\mathcal{MP}} = \text{PSPACE}$.

Parte III

Conclusiones y Líneas Futuras de Investigación

Capítulo 6

Conclusiones

En esta memoria se han presentado resultados de investigación en dos áreas de la computación celular con membranas (sistemas P). En primer lugar, se han detallado los resultados logrados tras el análisis del poder computacional de los sistemas P basados en neuronas de impulsos (SN P); y en segundo lugar, se han descrito los resultados del análisis de la eficiencia computacional de los sistemas P con proteínas en las membranas.

En relación con los SN Ps se planteaba la cuestión de si sería posible simplificar las numerosas características que posee el modelo sin perder poder computacional. De acuerdo a lo descrito en la sección 4.4 queda demostrado que es posible simplificar las expresiones regulares y eliminar los retrasos sin que el sistema pierda poder computacional. También se ha podido probar que los sistemas SN P mantienen la universalidad usando expresiones regulares simples y la condición fuerte de parada al igual que usando expresiones regulares simples con el sistema operando en modo aceptación (ver sección 4.5).

Queda patente que el modelo original de SN P es potente y robusto. Esto hace que pueda soportar que se elimine o restrinja alguna de las caracterís-

ticas estudiadas como: el grado sináptico tanto de entrada como de salida, los retrasos en las reglas, las reglas de olvido, las expresiones regulares o la sincronicidad; en todos esos casos el poder computacional no se ve afectado. Además queda demostrado que varias de esas características pueden ser eliminadas conjuntamente, por ejemplo limitar a dos el grado sináptico de salida y posiblemente también el de entrada combinado con alguna de las otras características mencionadas a excepción de la sincronicidad.

Sin embargo, la simplificación tiene un precio; simplificar y restringir algunos de los parámetros implica complicar o incrementar otros. En muchos casos como se ha podido comprobar incluso gráficamente, restringir el modelo implica generar diagramas más complejos, complicar las expresiones regulares, incrementar el número máximo de impulsos consumidos por cada neurona, incrementar el número de neuronas o incrementar el número de reglas por neurona. Desde el punto de vista computacional podríamos decir que los SN P poseen exceso de características que no conllevan a una mayor potencia de cálculo y por tanto son prescindibles. Sin embargo, simplificar el modelo de algún modo lo aleja posiblemente de la biología ya que en parte las características incluidas en el modelo inicial poseen cierta inspiración biológica.

En relación con los sistemas P con proteínas, se ha demostrado que las familias uniformes de estos sistemas P reconocedores y sin membrana de entrada pueden resolver los problemas de la clase ESPACIO-P (**PSPACE**), proporcionando una solución semi-uniforme a los problemas de esta clase. Por lo tanto, estos sistemas P son computacionalmente equivalentes a otros modelos de computación paralela como son las PRAM (Parallel Random Access Machine) o las máquinas alternantes de Turing [vEB91]. Además, suponemos que los mismos resultados obtenidos por las familias uniformes de sistemas P

reconocedores con entrada, proporcionarán soluciones uniformes, tal y como se deduce de [PJ10] aunque no existe por el momento demostración que lo pruebe. Una construcción similar a la propuesta en [AMVP03] podría resolver este problema. Por otro lado, la caracterización del poder computacional de los sistemas P no confluentes y no deterministas sigue abierta. La demostración utilizada no podrá adaptarse a este caso utilizando una máquina de Turing no determinista, ya que no se pueden almacenar elecciones no deterministas de este tipo de sistemas P a lo largo de una traza de computación debido a que requeriría un espacio exponencial.

Es importante destacar que ciertas operaciones utilizadas por los sistemas P con proteínas, como son la división de las membranas no elementales, parecen sugerir que en la práctica limitan su escalabilidad. Sin embargo, por otro lado el paralelismo masivo, el mínimo consumo de energía y la dimensión microscópica de los elementos de computación los hacen muy atractivos para buscar formas sobre cómo emplear la maquinaria microbiológica para resolver tareas algorítmicas.

Capítulo 7

Líneas Futuras

Como líneas futuras de investigación, un primer objetivo será estudiar el poder computacional probando nuevas simplificaciones en el modelo SN P. Como continuidad a los trabajos presentados en esta memoria, se estudiará el poder computacional de los SN P que no hagan uso de las reglas ampliadas y además no sean síncronos, los que no usen ni reglas de olvido ni expresiones regulares y los que no hagan uso ni de retrasos, ni de reglas de olvido ni de expresiones regulares.

El lector se preguntará si alguna de estas líneas de investigación planteadas no habrá sido ya resuelta. El motivo no es otro que el tiempo que ha transcurrido desde la publicación de los artículos de investigación y la publicación de esta memoria. Pues bien, estos problemas todavía a fecha de publicación de esta memoria de tesis siguen sin estar resueltos.

Para los problemas planteados sí podemos adelantar que la salida de los sistemas resultantes no podrá ser medida como el retraso entre el disparo de dos *spikes*. Para el primer caso resulta obvio, ya que no es un sistema síncrono; en cambio para los otros dos casos hay que fijarse en que la única posibilidad para bloquear o eliminar temporalmente *spikes* circulando a través de las

sinapsis del sistema es retrasar el periodo de la neurona. La neurona usa su periodo de retraso para dispararse tarde o temprano. Por supuesto, podemos bloquear los *spikes* de forma permanente en una neurona con reglas del tipo $a \rightarrow a$ y alimentarla con dos o más *spikes* simultáneamente. Por consiguiente, si algún *spike* circula dentro del sistema un periodo de tiempo suficiente y si el grafo sináptico conecta la región de actividad con la neurona de salida y si además no hay neuronas con bloqueo permanente, entonces tendrá lugar una salida después de un cierto retraso.

Como alternativa se podría tomar la salida únicamente como la actividad del sistema en ciertos intervalos de tiempo predefinidos e ignorar el resto de la actividad como un subproducto del metabolismo del sistema. Es posible que entonces sí se pueda lograr la universalidad. De lograrse sería relevante ya que existiría un cierto grado de afinidad con la biología. En biología los organismos vivos se comunican de esta forma, es decir, seleccionando cadenas de información producidas por otros y solo reconociendo aquellas partes que tienen sentido para ellos. Por último un gran objetivo a lograr sería alguna aplicabilidad de este tipo de sistemas, ya sea en el mundo biológico o de las ciencias de computación.

Otro problema abierto consistirá en analizar si sería posible mejorar estos resultados eliminando alguna característica más del sistema o incluso las tres simultáneamente. En caso de no lograr demostrar la universalidad, también podrá ser interesante analizar cuál será el poder computacional del sistema restringido. Obviamente también interesa la cuestión de si sería posible eliminar los límites inferiores de los parámetros computacionales para alguna de las demostraciones, tales como: el número de reglas en las neuronas, el número de *spikes* consumidos por una regla, etc.

Como líneas futuras en materia de los sistemas P con proteínas uno de

los objetivos a lograr será analizar la eficiencia computacional de su variante restringida. El análisis deberá centrarse en estudiar como cambiaría el poder computacional de las familias uniformes de los sistemas P con proteínas en el caso de que solo ciertos tipos de reglas estuviesen permitidas.

Las líneas de investigación anteriores son líneas abiertas que tienen continuidad con los resultados incluidos en esta memoria. Sin embargo, como se advertía en párrafos anteriores, en el área de los SN P se echa de menos la baja utilidad de los modelos descubiertos. En el último año se ha trabajado en una línea de investigación ambiciosa y cuyo objetivo es averiguar si los SN Ps podrían usarse como estimadores de la carga mental. En la siguiente sección se plantea esta nueva línea de investigación.

7.1. Sistema P de neuronas como modelo para medir/estimar la carga mental

Medir o estimar la carga mental de trabajo es un área de investigación que suscita mucho interés por sus potenciales usos en múltiples campos. Uno de ellos es la evaluación de la carga mental de los controladores de tráfico aéreo. Se entiende carga de trabajo, de acuerdo con Willian S. et al [PBCL96], como un elemento que no puede ser observado ni medido directamente, pero puede ser inferido basándose en medidas u observaciones de otros elementos tales como la mente o las tareas físicas que se llevan a cabo. Hoy en día existen muchos métodos disponibles para medir la carga mental. Es posible usar métodos invasivos o directos [HW90] usando sensores conectados directamente al humano, entre los que se incluye utilizar los datos proporcionados por un electroencefalograma (EGG) [LGNL01]. En cambio también se pueden usar métodos no invasivos en los que la carga mental puede ser inferida a partir

de la carga de tareas sobre los procesos cognitivos, según la Teoría de Recursos Múltiples propuesta por Wickens [Wic91] e implementada en el modelo computacional W/INDEX creado por Robert A. Nord y Victor A [NR89].

El objetivo común de todos los métodos descritos anteriormente, es obtener una cuantificación de una incierta capacidad cerebral que podría asemejarse a una medida de carga mental. El cerebro y cómo éste opera, todavía no se conoce en detalle; sin embargo, desde el descubrimiento de la célula neuronal por R. Cajal [RyC99] se han descubierto detalles de cómo operan las redes de las neuronas. Se ha descubierto, por ejemplo, que las redes de neuronas a bajo nivel, operan enviando trenes de impulsos (spikes) de manera paralela entre las diferentes neuronas. En la actualidad existen modelos de redes de neuronas conocidos como redes de neuronas basadas en impulsos. Estos modelos incluyen los nuevos descubrimientos [PMB12] [IPY06] que han tenido lugar en la neurociencia y la computación natural y cuya aplicación podría ser útil en otras disciplinas como es el caso de la medición de la carga mental en el control aéreo.

Los nuevos avances en neurociencia computacional [Maa02] [LM14] y computación natural [IPY06] podrían ser útiles para idear un modelo capaz de estudiar los efectos de la automatización sobre la carga mental. Para ello, como línea futura se propone crear un modelo inspirado en la forma en la que las neuronas operan, es decir, un modelo basado en las redes de neuronas de impulsos estudiadas en esta memoria (SN P). Para ello será necesario traducir tareas a la unidad mínima de información con la que trabajan las neuronas, los impulsos. Puede que sea posible crear un modelo basado en sistemas P de redes de neuronas de impulsos que permita averiguar el efecto que un entorno de multitarea provoca al cerebro, como es el caso del control aéreo. El futuro modelo de investigación podría medir o detectar problemas

de carga mental midiendo el retraso entre las respuesta de impulso o los cuellos de botella ocasionados por la demanda mental de tareas.

Bibliografía

- [AFR06] Artiom Alhazov, Rudolf Freund, and Yurii Rogozhin. *Computational power of symport/antiport: history, advances, and open problems*. Springer, 2006.
- [AMVP03] Artiom Alhazov, Carlos Martín-Vide, and Linqiang Pan. Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, 2003.
- [BAL⁺14] Alberts Bruce, Johnson Alexander, Julian Lewis, Morgan David, Raff Martin, Roberts Keith, and Peter Walter. *Molecular Biology of the Cell, Sixth Edition*. Garland Science, 6 edition, 2014.
- [Car05] Luca Cardelli. Brane calculi. In *Computational methods in systems biology*, pages 257–278. Springer, 2005.
- [CEI⁺07a] M. Cavaliere, O. Egecioglu, O.H. Ibarra, S. Woodworth, M. Ionescu, and Gh. Paun. Asynchronous spiking neural P systems technical report 9/2007. Technical Report 9-2007, Microsoft Research University of Trento, 2007.

- [CEI⁺07b] Matteo Cavaliere, Ömer Egecioglu, Oscar H. Ibarra, Mihai Ionescu, Gheorghe Paun, and Sara Woodworth. Asynchronous spiking neural P systems: Decidability and undecidability. In Max H. Garzon and Hao Yan, editors, *DNA*, volume 4848 of *Lecture Notes in Computer Science*, pages 246–255. Springer, 2007.
- [CFI⁺07] Haiming Chen, Rudolf Freund, Mihai Ionescu, Gheorghe Paun, and Mario J. Pérez-Jiménez. On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75:141–162, 2007.
- [CII⁺08] Haiming Chen, Mihai Ionescu, Tseren-Onolt Ishdorj, Andrei Paun, Gheorghe Păun, and Mario J. Pérez-Jiménez. Spiking neural P systems with extended rules: universality and languages. *Natural Computing*, 7(2):147–166, 2008.
- [CIP⁺06] Haiming Chen, Mihai Ionescu, Andrei Paun, Gheorghe Păun, and Bianca Popa. On trace languages generated by (small) spiking neural P systems. In *8th International Workshop on Descriptive Complexity of Formal Systems - DCFS 2006, Las Cruces, New Mexico, USA, June 21 - 23, 2006. Proceedings*, pages 94–105, 2006.
- [Coo00] G. M. Cooper. *The Cell - A Molecular Approach 2nd Edition*. Sunderland (MA): Sinauer Associates, 2000.
- [GPRS08] Marc García-Arnau, David Pérez, Alfonso Rodríguez-Patón, and Petr Sosík. On the power of elementary features in spiking neural P systems. *Natural Computing*, 7(4):471–483, 2008.

- [GPRS09] Marc García-Arnau, David Pérez, Alfonso Rodríguez-Patón, and Petr Sosík. Spiking neural P systems: Stronger normal forms. *IJUC*, 5(5):411–425, 2009.
- [HW90] Sandra G Hart and Christopher D Wickens. Workload assessment and prediction. In *Manprint*, pages 257–296. Springer, 1990.
- [IPP⁺07] Oscar H. Ibarra, Andrei Păun, Gheorghe Păun, Alfonso Rodríguez-Patón, Petr Sosík, and Sara Woodworth. Normal forms for spiking neural P systems. *Theor. Comput. Sci.*, 372(2-3):196–217, March 2007.
- [IPY06] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. Spiking neural P systems. *Fundam. Inf.*, 71(2,3):279–308, August 2006.
- [IW06] Oscar H. Ibarra and Sara Woodworth. Characterizations of some restricted spiking neural P systems. In *Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006, Revised, Selected, and Invited Papers*, pages 424–442, 2006.
- [KR08] Lila Kari and Grzegorz Rozenberg. The many facets of natural computing. *Commun. ACM*, 51(10):72–83, October 2008.
- [LBK⁺07] Harvey Lodish, Arnold Berk, Chris A. Kaiser, Monty Krieger, Matthew P. Scott, Anthony Bretscher, Hidde Ploegh, and Paul Matsudaira. *Molecular Cell Biology (Lodish, Molecular Cell Biology)*. W. H. Freeman, 6th edition, June 2007.

- [LGNL01] C Ling, H Goins, A Ntuen, and R Li. EEG signal analysis for human workload classification. In *SoutheastCon 2001. Proceedings. IEEE*, pages 123–130. IEEE, 2001.
- [Lin68] Aristid Lindenmayer. Mathematical models for cellular interaction in development: Parts i and ii. *Journal of Theoretical Biology*, 18, 1968.
- [LM14] Robert Legenstein and Wolfgang Maass. Ensembles of spiking neurons with noise support optimal probabilistic inference in a dynamically changing environment. *PLOS Computational Biology*, 2014.
- [Maa02] W. Maass. Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8:32–36, 2002.
- [MB99] Wolfgang Maass and Christopher M. Bishop, editors. *Pulsed Neural Networks*. MIT Press, Cambridge, MA, USA, 1999.
- [Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
- [MVPPRP03] Carlos Martín-Vide, Gheorghe Păun, Juan Pazos, and Alfonso Rodríguez-Patón. Tissue P systems. *Theoretical Computer Science*, 296(2):295–326, March 2003.
- [NR89] Robert A North and Victor A Riley. W/INDEX: A predictive model of operator workload. In *Applications of human performance models to system design*, pages 81–89. Springer, 1989.

- [Pă99] Gheorghe Păun. P systems with active membranes: Attacking NP complete problems. *Journal of Automata, Languages and Combinatorics*, 6:75–90, 1999.
- [Pău98] Gheorghe Păun. Computing with membranes. Technical Report 208, Turku Center for Computer Science-TUCS, 1998. (www.tucs.fi).
- [Pău02] Gheorghe Păun. *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
- [PBCL96] William S Pawlak, Christopher R Brinton, Kimberly Crouch, and Kenneth M Lancaster. A framework for the evaluation of air traffic control complexity. In *Proceedings of the AIAA Guidance Navigation and Control Conference, San Diego, CA*, 1996.
- [PJ10] Mario J Pérez-Jiménez. A computational complexity theory in membrane computing. In *Membrane Computing*, pages 125–148. Springer, 2010.
- [PMB12] Hélène Paugam-Moisy and Sander Bohte. Computing with spiking neuron networks. In *Handbook of Natural Computing*, pages 335–376. Springer, 2012.
- [PP02] Andrei Păun and Gheorghe Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3):295–305, May 2002.
- [PP06a] A. Paun and B. Popa. P systems with proteins on membranes and membrane division. In *Proceedings 10th DLT Conf. (invi-*

- ted talk*), Santa Barbara, USA, 2006, volume 4036 of *Lecture Notes in Computer Science*, pages 292–303. Springer, 2006.
- [PP06b] Andrei Paun and Bianca Popa. P systems with proteins on membranes. *Fundam. Inform.*, 72(4):467–483, 2006.
- [PP07] Andrei Păun and Gheorghe Păun. Small universal spiking neural P systems. *BioSystems*, 90(1):48–60, 2007.
- [PPJS06] Gh. Paun, M.J. Perez-Jimenez, and A. Salomaa. Bounding the indegree of spiking neural P systems. Technical report, TUCS Technical Report 773, 2006.
- [PRS10] Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
- [RyC99] S Ramón y Cajal. Textura del sistema nervioso del hombre y de los vertebrados: estudios sobre el plan estructural y composición histológica de los centros nerviosos, adicionados de consideraciones fisiológicas fundadas en los nuevos descubrimientos. *Madrid N Moya*, 1899.
- [Sit15] P Systems Oficial Web Site. The P systems web page. <http://ppage.psystems.eu>, 2015. último acceso: 2015-01-01.
- [Sos03] Petr Sosík. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, August 2003.
- [Sos14] Petr Sosík. Active membranes, proteins on membranes, tissue P systems: Complexity-related issues and challenges. In

- Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 40–55. Springer Berlin Heidelberg, 2014.
- [vEB91] Peter van Emde Boas. Machine models and simulations, Handbook of Theoretical Computer Science (Vol. A): algorithms and complexity, 1991.
- [Wic91] Christopher D Wickens. Processing resources and attention. *Multiple-task performance*, pages 3–34, 1991.