Universidad de Sevilla
Dpto. de Ciencias de la Computación
e Inteligencia Artificial

Programación celular: resolución eficiente de problemas numéricos NP-completos

Memoria presentada por
Agustín Riscos Núñez
para optar al grado de Doctor
por la Universidad de Sevilla

Agustín Riscos Núñez

V.º B.º Los Directores de la Tesis

D. Mario de J. Pérez Jiménez

D. Miguel A. Gutiérrez Naranjo

Sevilla, 24 de junio de 2004

A Pilar

Agradecimientos

Me gustaría mencionar aquí a todos los que me han ayudado y acompañado en todas las etapas de mi formación académica y de mi formación como persona, a todos mis amigos y compañeros, y a todos mis profesores. En especial a Alejandro, porque influyó de manera decisiva en mi manera de concebir las Matemáticas, y en mi elección de matricularme en el programa de doctorado de este Departamento.

También quiero agradecer a Mario, no sólo el haber sido uno de los directores de esta tesis, sino también el haberme guiado y ayudado tanto desde que inicié mis estudios de doctorado. Tengo que darle las gracias por haberse ocupado y preocupado por mí, tanto en el ámbito académico/profesional como en el personal.

A Miguel Ángel por su apoyo y su ayuda como director, y también por haberme acogido como a un compañero, durante las horas de trabajo y también fuera de ellas.

A todos los miembros del Departamento de Ciencias de la Computación e Inteligencia Artificial, por saber hacerme un hueco entre ellos, y en especial a Toñi, por compartir conmigo todos los días.

A mi familia, por ser un respaldo constante para mí. Sobre todo a mis padres, por demasiados motivos como para decirlos aquí ("*el margen es demasiado estrecho*").

Y por supuesto a Pilar, porque sin ella no sería como soy.

Índice general

Introducción

Los ordenadores electrónicos actuales no son más que el último eslabón de una larga cadena de esfuerzos del hombre por mecanizar el razonamiento en la medida de lo posible y usar la mejor tecnología disponible para realizar computaciones (en definitiva, por resolver de manera *eficiente* problemas de la vida real).

Pero estos dispositivos computacionales convencionales tienen sus propias limitaciones determinadas por unas leyes físicas inexorables. En 1983, R. Churchhouse establece las limitaciones físicas de la velocidad de cálculo de los procesadores convencionales basados en la manipulación electrónica del silicio, así como las limitaciones propias en lo que respecta a la miniaturización de las componentes físicas de los dispositivos.

En la década de los cincuenta, aparecen las primeras máquinas de cálculo programables de propósito general y, con ellas, surgen los que podríamos denominar informalmente como *modelos de computación práctica*, dando lugar al nacimiento de la *Teoría de la Complejidad*, cuyo objetivo principal es el análisis de la cantidad de recursos computacionales necesarios para resolver un problema abstracto de *forma mecánica*.

La existencia de problemas que para su resolución necesitan una cantidad de recursos que excede las capacidades de cualquier *dispositivo computacional convencional*, independientemente de cuál sea el procedimiento utilizado para resolverlo, nos conduce a la búsqueda de alternativas que nos den la posibilidad de resolver esos mismos problemas de alguna manera *más eficiente*. Es decir, surge la necesidad de encontrar nuevos modelos que de alguna forma nos permitan solucionar instancias de tamaño *relativamente grande*, de problemas *intratables* o *presuntamente intratables*.

Una primera vía para lograr la aceleración de procesos computacionales es permitir que varias operaciones puedan ser realizadas en una misma unidad de tiempo (modelos de *computación paralela*). Sin embargo, esto no es suficiente para abordar la cuestión anterior de manera mínimamente satisfactoria, ya que la aceleración obtenida se ve contrarrestada, en gran medida, por el coste en número de procesadores necesarios y por la complejidad de la red de conexiones entre dichos procesadores.

Otra vía para mejorar el rendimiento de las máquinas consiste en la miniaturización de sus componentes físicas. Hacia finales de la década de los cincuenta, Richard Feynman [18] introduce el concepto teórico de computación a nivel molecular y lo postula como una innovación revolucionaria en la carrera por la miniaturización, aunque tuvieron que pasar muchos años antes de que la comunidad científica se dedicara a explorar debidamente esta nueva vía.

El estudio de modelos *no convencionales* para atacar la resolubilidad algorítmica práctica de problemas importantes de la vida real que son presuntamente intratables, pasa a ser una alternativa a considerar. Se trata de encontrar nuevos dispositivos computacionales que nos permitan mejorar, al menos *cuantitativamente*, la resolución práctica de esos problemas *difíciles*. Una mejora cuantitativa significa ampliar (en cierta medida) el rango de los tamaños de las instancias de esos problemas que pueden ser resueltos en los nuevos dispositivos.

Muchos desarrollos importantes en el marco de la computación práctica se han realizado inspirándose en la Naturaleza ¿Por qué no volver la mirada hacia ella para intentar diseñar nuevos dispositivos computacionales?

La *Computación Natural* es una disciplina inspirada en la estructura y el funcionamiento de los organismos vivos. Tiene como objetivo fundamental la simulación e implementación de los procesos dinámicos que se dan en la Naturaleza y que son susceptibles de ser interpretados como procedimientos de cálculo.

En la actualidad, la Computación Natural consta de varias ramas. Una primera está inspirada en el funcionamiento del *cerebro*. Elementos simples (*neuronas*) conectados de manera adecuada en una red (*sistema nervioso*) pueden llegar a tener una gran potencia computacional. Esta idea ha dado lugar a un modelo informático: las *redes neuronales artificiales*, creadas por W.S. McCulloch y W. Pitts en 1943. Se trata de redes de procesadores simples, conectados por canales de comunicación, que operan solo en sus datos locales y en los datos recibidos por dichos canales. Adicionalmente, estos modelos disponen de un procedimiento de *entrenamiento* y *aprendizaje*.

Otra rama está basada en los procesos de evolución y selección natural como elementos relevantes en el diseño e implementación de sistemas de resolución de problemas mediante ordenadores electrónicos. Un modelo computacional de esta rama son los *Algoritmos Evolutivos*, cuyos orígenes se remontan a finales de la década de los cincuenta en los trabajos de A. Fraser [20] y G.E.P Box [10], y se consolida en 1966 con los trabajos de L. Fogel, A. Owens y H. Walsh [19]. Otro modelo lo constituyen los *algoritmos genéticos*, creados por J.H. Holland en 1975.

Una tercera rama está basada en las propiedades de las moléculas de **ADN** y tiene como precedente el *modelo splicing* desarrollado por T.J. Head en 1987. Su nacimiento

como modelo propio, la *Computación Molecular basada en ADN*, se produjo cuando, en noviembre de 1994, L. Adleman resuelve una instancia concreta del problema del camino hamiltoniano (que, como es bien sabido, es un problema **NP**–completo; es decir, presuntamente intratable) mediante un experimento realizado en el laboratorio, usando moléculas de ADN y operaciones sobre ellas. De esta forma se justifica la posibilidad práctica de codificar y manipular información mediante cadenas de ADN a fin de resolver problemas matemáticos *especialmente difíciles*. Esto es, la computación molecular basada en ADN permite resolver, de manera efectiva, problemas en el laboratorio, operando con las moléculas de ADN *in vitro*. El objetivo es lograr hacerlo *in vivo*, utilizando alguna especie de *hardware biológico*.

La *Computación celular con membranas* es una nueva rama de la Computación Natural, creada Gh. Păun en octubre de 1998 e inspirada en la estructura y el funcionamiento de las células. Es un modelo de computación no determinista, de tipo paralelo y distribuido. Esquemáticamente, podemos considerar un sistema de computación celular con membranas, o sistema P, como un dispositivo constituido por una estructura de membranas que contienen objetos que se desplazan y evolucionan de acuerdo a determinadas reglas especificadas.

Entre esta rama y las tres anteriores existe en la actualidad una diferencia importante en cuanto a su nivel de aplicación e implementación. Por una parte, tanto las redes neuronales artificiales como los algoritmos evolutivos y genéticos han sido implementados en ordenadores electrónicos (*in silico*). Concretamente, se han implementado algoritmos prácticos para la resolución de determinados tipos de problemas en dichos modelos computacionales. La computación molecular basada en ADN ha sido implementada en laboratorios de biología molecular. En cambio los sistemas celulares aún no han sido implementados en medios electrónicos ni bioquímicos.

La memoria que presentamos se sitúa en el marco teórico de la *Computación celular con membranas*. Concretamente, se introducirá formalmente una variante de sistemas de computación celular con membranas que va a ser usada como modelo para atacar la resolución de problemas **NP**–completos: los sistemas P reconocedores con membranas activas.

Contenido de la memoria

Esta memoria está estructurada en capítulos cuyos contenidos pasamos a describir sucintamente.

En el **Capítulo 1** se hace una breve introducción histórica de la *Teoría de la Computabilidad*, analizándose las limitaciones y potencia de los modelos que formalizan el

concepto de *procedimiento mecánico*, así como de la *Teoría de la Complejidad Computacional*, justificándose la necesidad de estudiar nuevos modelos de computación a fin de mejorar la *resolución cuantitativa* de problemas matemáticos *especialmente difíciles*, desde el punto de vista de esta teoría. También se describen brevemente nuevos modelos de computación inspirados en la forma en que la Naturaleza *calcula*.

El **Capítulo 2** está dedicado a la presentación del marco general en que se va a desarrollar esta memoria, la Computación celular con membranas. Concretamente se describen de manera informal los *sistemas* P *de transición*, que es el modelo considerado por Gh. Păun en el artículo fundacional de la disciplina. A continuación, se introducen los *sistemas celulares reconocedores*, que son dispositivos computacionales en el marco de la computación celular con membranas, especialmente adecuados para reconocer lenguajes y, en consecuencia, para resolver problemas de decisión.

Finalmente, se define una clase de complejidad polinomial que proporciona un concepto de *resolubilidad eficiente de problemas* a través de sistemas celulares reconocedores; es decir, un concepto de *tratabilidad* en este nuevo marco no convencional. Esta clase se sitúa por el momento a nivel teórico ya que, como hemos dicho, aún no existen implementaciones en soporte electrónico ni bioquímico.

A menos que **P** = **NP**, para atacar de manera eficiente la resolubilidad de problemas **NP**–completos es necesario disponer de mecanismos que permitan fabricar una cantidad de espacio de tamaño exponencial en tiempo polinomial. Los sistemas celulares con membranas activas son una variante de modelo de computación con membranas que satisface este requisito.

En el **Capítulo 3** se presenta una formalización de estos sistemas celulares que, en su versión de reconocedores, serán usados a lo largo de la memoria para resolver problemas **NP**–completos, *de manera eficiente*, en el marco de la clase de complejidad antes mencionada. En concreto, todas las computaciones de los sistemas diseñados *paran* y, además, lo hacen tras un número de pasos celulares de orden *lineal*.

Los **Capítulos 4, 5** y **6** están dedicados al estudio de soluciones eficientes de tres problemas numéricos **NP**–completos: *Subset Sum*, *Knapsack* y *Partición*, a través de sistemas celulares reconocedores con membranas activas. En todas las soluciones se sigue una misma estructura:

- En primer lugar, se presentan los diseños de las correspondientes familias de sistemas celulares.
- A continuación, se realiza una breve descripción intuitiva del funcionamiento de los mismos.
- Finalmente, se demuestra la verificación formal de las soluciones, de acuerdo

con la definición de clases de complejidad dada en el Capítulo 2.

Además, al final del Capítulo 6 se incluyen una serie de comentarios en la línea de minimizar los recursos que emplean los sistemas utilizados. Se indica que es posible reducir en uno el número de cargas eléctricas distintas que se admiten. Más concretamente, en los sistemas P con membranas activas se considera que las membranas pueden tener carga positiva, negativa o neutra, pero se muestra que basta considerar solo dos cargas para poder obtener soluciones celulares eficientes.

En las soluciones presentadas en los Capítulos 4, 5 y 6 de los problemas antes citados, se ha observado grandes similitudes entre los diferentes diseños. Ello, unido a las enormes dificultades que existen a la hora de diseñar dispositivos computacionales en los modelos de computación orientados a máquinas (como es el caso de los sistemas P), hace que adquiera especial relevancia la búsqueda de *subrutinas* que agrupen conjuntos de reglas de evolución de los sistemas, de tal manera que puedan ser utilizadas a modo de *instrucciones* de un lenguaje de programación celular. Este es el objetivo fundamental del **Capítulo 7**: la idea de un *lenguaje de programación celular* es factible, al menos para cierta clase relevante de problemas **NP**-completos, y podría ser de utilidad a la hora de diseñar y verificar soluciones para nuevos problemas en el futuro.

En el **Capítulo 8** se presenta un simulador de sistemas celulares, escrito en Prolog, que permite realizar simulaciones de las soluciones celulares presentadas en los capítulos anteriores. Se trata de una simulación secuencial, ya que en un ordenador convencional no se puede implementar la creación de un espacio de trabajo de tamaño exponencial en tiempo polinomial que llevan a cabo, a nivel teórico, los sistemas P con membranas activas. Conviene aclarar que el simulador no ha sido diseñado *específicamente* para las familias de sistemas P reconocedores presentadas en esta memoria, sino que ha sido concebido para poder simular cualquier sistema P con membranas activas.

A la hora de analizar soluciones de dispositivos computacionales en el marco de la complejidad computacional, se estudian, básicamente, los recursos en tiempo y/o en espacio utilizados. Pero estas medidas de complejidad pueden resultar claramente insuficientes cuando se trabaja con sistemas celulares con membranas activas que permiten la división de membranas en un paso de transición. En el **Capítulo 9** se estudia la necesidad de profundizar en el estudio de nuevos parámetros que permitan analizar la complejidad de los sistemas P como dispositivos computacionales que resuelven problemas de decisión. Esos parámetros pueden orientar al usuario en la tarea de diseñar *mejores* sistemas que resuelven un mismo problema.

La memoria concluye detallando algunas conclusiones que se pueden extraer de

los resultados obtenidos, y se presentan una serie de objetivos y trabajos futuros que marcan nuevas líneas de investigación en modelos de computación no convencionales, algunas de las cuales han comenzado a desarrollarse en colaboración con algunos miembros del grupo de investigación en Computación Natural de la Universidad de Sevilla.

Aportaciones

A continuación citamos algunas de las aportaciones de esta memoria que consideramos más relevantes.

1. Diseño, análisis y verificación formal de las primeras soluciones celulares eficientes de los problemas numéricos **NP**–completos *Subset Sum*, *Knapsack* y *Partición*, a través de sistemas P reconocedores con membranas activas.

2. Presentación de una nueva aproximación encaminada hacia una metodología que pueda ser útil para la verificación formal de *soluciones celulares* de problemas de decisión.

3. Demostración de que dos cargas eléctricas (sin efectuar cambios de etiquetas en la aplicación de las reglas) son suficientes para proporcionar soluciones celulares eficientes de problemas **NP**–completos en el marco de los sistemas reconocedores con membranas activas.

4. Desarrollo de un primer lenguaje específico de programación celular, mediante la elaboración de subrutinas asociadas a sistemas celulares, si bien con un ámbito restringido de aplicación.

5. Presentación del diseño revisado de un simulador Prolog de sistemas celulares con membranas activas, que sirve de asistente en los procesos de diseño y verificación de sistemas celulares.

6. Programación de módulos auxiliares para el simulador Prolog encargados de generar los conjuntos de reglas y las configuraciones iniciales correspondientes a los diseños presentados en esta memoria.

7. Propuesta de nuevos parámetros que complementan el análisis de la complejidad descriptiva de sistemas celulares reconocedores que resuelven problemas.

Capítulo 1

Computación Natural

En este primer capítulo se presenta una breve introducción histórica a la Teoría de la Computación, es decir a la gestación del concepto de *resolución mecánica* de un problema y a la necesidad de formalizar dicho concepto ante la posibilidad de que existan problemas que no puedan ser resueltos de forma mecánica. Asimismo, se comentarán algunas cuestiones relativas a la Teoría de la Complejidad Computacional, que trata del análisis de los recursos necesarios (básicamente, en tiempo y/o en espacio) para resolver mecánicamente un problema.

La necesidad de resolver problemas usando dispositivos computacionales y la existencia de problemas que necesitan una cantidad *excesiva* de recursos (cuantificados en términos exponenciales) para poder ser resueltos en dichos dispositivos conducen a la búsqueda de vías alternativas, de nuevos modelos de computación inspirados en la forma en que la Naturaleza lleva realizando, desde hace millones de años, una serie de procesos que pueden ser considerados como procedimientos de cálculo, en cierto sentido. La última sección de este capítulo está dedicada a la descripción, breve, de las ramas principales que existen actualmente dentro de la disciplina denominada *Computación Natural*.

1.1. Introducción

Hablar del nacimiento de las Matemáticas, o de cuándo fue la primera vez que el hombre trató de *contar* o *medir* su entorno, es algo prácticamente imposible. La necesidad de supervivencia de la especie, así como la constante aspiración de mejorar la calidad de vida, ha llevado al hombre a plantearse como un problema capital el conocimiento profundo del mundo y de los fenómenos e interacciones que en él se producen. La voluntad de cuantificar, en cierta medida, el mundo que nos rodea y manejar las cantidades obtenidas puede constatarse incluso en el Antiguo Egipto o

en la cultura Babilónica. Sin embargo, el concepto de *computación*, tal y como lo entendemos hoy en día, es relativamente moderno.

Etimológicamente, la palabra **computar** viene del latín *computare*, y tiene la acepción de contar o calcular alguna cosa a través de números, principalmente los años, tiempos y edades (fuente: D.R.A.E.). Sin embargo, actualmente su significado se ha visto modificado, orientándose más bien hacia el mundo de la Informática[1].

Suele ser bastante habitual confundir computación con programación de ordenadores, pero de hecho la teoría de la computación es mucho más genérica. Además, su nacimiento se produce casi dos décadas antes de la aparición de los primeros ordenadores (o *computadores*). A lo largo de este trabajo vamos a considerar que computar es, en esencia, manejar datos o información siguiendo unas reglas que pueden ser consideradas como mecánicas. Se trata, pues, de un proceso dinámico en el que los datos iniciales (la *entrada*) son transformados de acuerdo con unas reglas, especificadas a veces mediante instrucciones, y como consecuencia de transformaciones sucesivas se obtiene, a veces, un resultado expresado mediante nuevos datos (la *salida*).

Esta idea de disponer de unas reglas que indican cómo procesar los datos nos conduce al concepto de *algoritmo*. Un algoritmo suele tomar la forma concreta de un conjunto de instrucciones debidamente secuenciadas que indican unívocamente y paso a paso lo que se debe hacer para obtener la respuesta a cada pregunta formulada. El concepto de algoritmo está en la base de la definición de los llamados *conceptos constructivos*, tales como los de *calculabilidad* de una función, *decidibilidad* de una propiedad o relación, y *generación* de un conjunto.

Históricamente, el primer algoritmo no trivial es debido a Euclides que entre el año 400 y 300 a.C. describió un procedimiento mecánico para hallar el máximo común divisor de dos números enteros arbitrarios. Sin embargo, la palabra *algoritmo* tiene un origen más moderno: etimológicamente debe su nombre al autor persa Abú Jáfar Mohammed ibn al–Khowarizmi, que escribió un texto en el año 825 d.C. en el que recogía una serie de procedimientos mecánicos para el álgebra (en particular una serie de reglas que, secuenciadas en un determinado orden, permitían realizar las operaciones con números decimales).

[1] Por ejemplo, en *www.enciclonet.com* encontramos el término *Computación* con el siguiente significado: Elaboración de informaciones de naturaleza numérica o no, realizada por un operador humano o mecánico, siguiendo indicaciones expresadas en términos detallados y sin ambigüedad (por ejemplo, mediante las instrucciones expresadas en un lenguaje natural o en un programa).

1.2. Teoría de la Computación

Podemos afirmar que el primero en plantear la necesidad de formalizar qué se entiende por procesos (o razonamientos) mecánicos (o automáticos) fue G.W. Leibniz, cuando a finales del siglo XVII formula la necesidad de disponer de un lenguaje universal (*lingua characteristica*) en el que poder expresar cualquier idea, y la necesidad de mecanizar cualquier tipo de razonamiento (*calculus ratiocinator*). Sin embargo, hasta la primera mitad del siglo XX no se lograron avances destacables. El encargado de concretar las tareas pendientes para el siglo que comenzaba fue D. Hilbert, con su famosa lista de 23 problemas presentada en el Congreso Internacional de Matemáticos en París, en 1900. Posteriormente, en el Congreso celebrado en Bolonia en 1928, planteó a la comunidad científica tres cuestiones que marcarían el devenir de la Ciencia:

1. *Completitud de las Matemáticas*: ¿es posible, dado cualquier aserto, encontrar una demostración para éste o para su negación?

2. *Consistencia de las Matemáticas*: ¿se puede garantizar que no existe ningún aserto tal que sea posible encontrar demostraciones tanto de él como de su negación?

3. *Decidibilidad de las Matemáticas*: ¿existe algún procedimiento mecánico que nos permita, dado cualquier aserto, determinar si es demostrable o no?

En 1931, el matemático K. Gödel dio una respuesta negativa a la primera de estas tres cuestiones, considerando un sistema axiomático que contenía la Aritmética elemental y construyendo una proposición verdadera que no se podía probar en dicho sistema. Además, justificó que no era posible dar respuesta a la segunda cuestión en el marco de las propias Matemáticas, demostrando que, si se supone que un sistema sea consistente y que se verifiquen ciertas propiedades *básicas*, entonces el sistema no es capaz de probar una fórmula que exprese su propia consistencia. En cuanto a la tercera cuestión, A. Church y A. Turing, de manera independiente, dieron una respuesta negativa en 1936 al demostrar la indecidibilidad de la lógica de primer orden. Estos tres resultados proporcionaron importantes limitaciones a la hora de formalizar el concepto de procedimiento mecánico.

Modelos de computación

Todo modelo de computación trata de formalizar el concepto intuitivo de *función computable*, o función calculable por medio de un procedimiento mecánico. En todo

modelo se especifica claramente cómo se han de expresar los datos (*sintaxis* del modelo) y también cómo se modifican esos datos; es decir, cómo se representan las reglas y cómo han de aplicarse (*semántica* del modelo).

Los trabajos de K. Gödel, A. Church, S. Kleene y A. Turing entre 1931 y 1936, proporcionaron los primeros modelos de computación, definiendo rigurosamente el concepto de función computable; es decir, función cuyos valores pueden ser calculados de forma mecánica en el modelo. Concretamente, en 1931 K. Gödel define el concepto de relación recursiva e introduce la clase de funciones que denominó *recursivas* (y que hoy se conocen por el nombre de funciones *primitivas recursivas*). Posteriormente, en 1934 el propio K. Gödel extiende la clase anterior a las funciones *general recursivas* (y que hoy se conocen por el nombre de funciones *recursivas*). En 1931, A. Church y S. Kleene desarrollan el concepto de λ-cálculo relacionándolo directamente con el de función computable. En 1936, A. Turing utiliza por primera vez el concepto abstracto de *máquina* para formalizar la noción de algoritmo y, por tanto, el concepto de función computable.

En 1936, A. Church formula su famosa tesis acerca de la equivalencia entre la clase de funciones computables, en sentido intuitivo, y la clase de funciones λ-calculables. Además, en dicho año A. Church también proporciona el primer ejemplo de un problema para el que no existe procedimiento mecánico (en su modelo) que lo resuelva: el problema de la decidibilidad de la lógica de primer orden. Pocos meses después, A. Turing establece el mismo resultado pero en el modelo de las máquinas de Turing. Además, demuestra la equivalencia de los tres modelos de computación antes citados; en el sentido de que todo aquello que es calculable en uno de esos modelos lo es también en cualquiera de los otros dos. Dado que, según la tesis de Church-Turing, en dichos modelos se pueden hallar todas las funciones que sean computables en un sentido intuitivo, a todos los modelos equivalentes a las máquinas de Turing se les llama *universales* (o computacionalmente completos).

Esencialmente, existen tres orientaciones posibles a la hora de formalizar un modelo de computación:

- **modelos orientados a programas**: se define directamente, a través de un lenguaje de instrucciones básicas, el concepto de algoritmo como una sucesión finita de instrucciones que verifique una serie de requisitos sintácticos. A partir de esta noción, una *máquina de cálculo* será cualquier dispositivo capaz de ejecutar los algoritmos de ese modelo, y las *funciones computables* serán aquellas que pueden ser generadas, de manera natural, por los algoritmos del mismo (por ejemplo, el modelo GOTO [16]).

- **modelos orientados a funciones**: se considera un cierto conjunto de fun-

ciones distinguidas como la clase de *funciones computables* del modelo. A partir de este concepto, los *algoritmos* del modelo serán aquellos procedimientos que permitan generar, en algún sentido claramente fijado, las funciones computables; y las *máquinas de cálculo* serán aquellos dispositivos que puedan ejecutar los algoritmos (por ejemplo, los modelos de las funciones recursivas y de las λ-calculables).

- **modelos orientados a máquinas**: se describen sintácticamente los dispositivos o *máquinas* cuya ejecución (definida por una semántica precisa) proporciona las *funciones computables* del modelo (por ejemplo, el modelo de las máquinas de Turing).

1.3. Teoría de la Complejidad computacional

A la hora de resolver un problema en cierto modelo de computación, es importante disponer de unas herramientas que permitan de alguna manera cuantificar, *a priori*, la cantidad de recursos necesarios para ejecutar una *buena solución* en dicho modelo. Más concretamente, si disponemos de una máquina y queremos ejecutar en ella un algoritmo sobre cierto dato de entrada, puede ocurrir que la máquina se colapse y no pueda terminar la computación, o bien que tarde demasiado tiempo, si la cantidad de recursos requerida es demasiado elevada.

Ahora bien, existen problemas resolubles mecánicamente que tienen cierta *dificultad intrínseca* en el sentido de que *cualquier* algoritmo que los resuelva necesite una gran cantidad de recursos.

La *Teoría de la Complejidad Computacional* proporciona herramientas para medir el grado de dificultad inherente a la solución mecánica de problemas abstractos, tanto en términos absolutos (*complejidad intrínseca* de un problema) como en términos comparativos con otros problemas (*clases de complejidad*).

El objetivo fundamental de dicha teoría es la clasificación de problemas en función de la *resolubilidad algorítmica práctica* de los mismos. Para ello, se necesita definir el concepto de *eficiencia* o resolubilidad práctica, capturando la idea intuitiva de resolubilidad a través de ordenadores que existen actualmente. La definición de eficiencia aceptada comúnmente hoy en día es el concepto de *computabilidad en tiempo polinomial*, y fue introducida de manera explícita en la década de los sesenta por A. Cobham [13] y J. Edmonds [17], si bien J. von Neumann [52] había establecido en 1953 una distinción entre algoritmos que se ejecutan en tiempo polinomial y los que se ejecutan en tiempo exponencial. Además, Edmonds llamó *buenos algoritmos* a los que se ejecutan en tiempo polinomial y los consideró como *algoritmos tratables*. Así, un

algoritmo se dirá *eficiente* si la cantidad de recursos necesarios para su ejecución, en el caso peor, está acotada por un polinomio en el tamaño del dato de entrada. De esta manera se fija una frontera entre la resolubilidad algorítmica práctica (*tratabilidad*) y la no resolubilidad algorítmica práctica (*intratabilidad*) de problemas.

1.4. La Naturaleza como fuente de inspiración computacional

Desde los inicios de la Teoría de la Computación hasta nuestros días se han descrito, presentado y estudiado muchos modelos de computación. Unos han gozado de gran aceptación y han sido ampliamente desarrollados en busca de aplicaciones, aunque otros corrieron peor suerte y resultaron no ser fructíferos.

La *Computación Natural* es un campo emergente que estudia modelos abstractos de computación a partir de ideas o paradigmas extraídos de la observación de los procesos que ocurren en la Naturaleza y que son susceptibles de ser interpretados como procedimientos de cálculo.

Muchos científicos se han acercado a este campo desde enfoques distintos, creando una gran diversidad de líneas de investigación tanto en el plano teórico como en el práctico. Comentemos a continuación, de manera breve, algunas de las principales ramas en las que se enmarcan estas líneas.

1.4.1. Redes Neuronales Artificiales

Por antigüedad, tal vez habría que citar en primer lugar las *Redes Neuronales Artificiales* (ideadas por McCulloch y Pitts [33] en 1943), cuyo objetivo principal es mejorar la potencia de cálculo a partir de unidades sencillas conectadas entre sí, imitando el esquema de interconexiones que presentan las neuronas en el cerebro. Cada unidad de cálculo o neurona puede abstraerse hasta reducirse a un dispositivo que recibe una cierta cantidad de datos numéricos (que pueden provenir de otras neuronas) y produce un único valor numérico como salida (que puede ser enviado como entrada a otra neurona). Este modelo ha resultado ser uno de los más efectivos en cuanto a problemas de interpretación de sensores del mundo real, tales como interpretación de escenas visuales (reconocimiento de rostros o de escritura a mano), reconocimiento de lenguaje hablado o aprendizaje de estrategias para el control de robots. Una buena panorámica de aplicaciones prácticas dada por Rumelhart y otros puede ser encontrada en [49].

Como ejemplo para poder captar el alcance de este modelo consideremos algunos datos de neuro-biología. Se estima que el cerebro humano está formado por una

red densamente interconectada con aproximadamente 10^{11} neuronas, en donde cada neurona está conectada, de media, con otras 10^4. La actividad neuronal se activa o se inhibe normalmente a través de las propias interconexiones. Los tiempos de alternancia (o cambio de estado) más rápidos que se han calculado son de 10^{-3} segundos, lo cual es bastante más lento que los 10^{-10} segundos en el caso de un ordenador. Sin embargo, los humanos son capaces de tomar decisiones complejas en tiempos asombrosamente rápidos. Por ejemplo, tan sólo tardamos aproximadamente una décima de segundo en reconocer el rostro de nuestra madre. Nótese que la cadena de activaciones de neuronas que puede tener lugar en ese pequeño intevalo es de tan sólo algunos centenares de pasos, debido a la velocidad de alternancia de cada neurona. Esta aseveración nos lleva a especular que la capacidad de procesamiento de información que tienen los sistemas neuronales biológicos debe basarse en procesos de manipulación altamente paralelos que actúan sobre representaciones que se hallan distribuidas entre múltiples neuronas.

No entraremos aquí en más detalles acerca de cómo las redes neuronales artificiales tratan de implementar o simular esta manera de actuar. Para un estudio de las redes neuronales artificiales aplicadas al aprendizaje automático, véase el capítulo 4 de [34].

1.4.2. Computación Evolutiva y Algoritmos Genéticos

La *Computación evolutiva* está basada en el uso de modelos computacionales de procesos evolutivos como elementos relevantes en el diseño e implementación de sistemas de resolución de problemas mediante ordenadores electrónicos.

Las primeras ideas que hacían uso de la idea de *evolución* en el mundo de la computación fueron dadas por J. von Neumann que postuló que la vida debía estar sustentada por un código que permitiera generar individuos y a la vez les dotase de capacidad reproductiva. Como consecuencia, una máquina o autómata que se pretendiese fuese autorreproductiva debía contener, además de las instrucciones para su formación, aquellas que le permitiesen copiar tales instrucciones a su descendencia.

Pero no sería hasta mediados de la década de los cincuenta cuando se produjo una implementación práctica de las ideas evolutivas para mejorar procesos, en este caso industriales, a través de los trabajos de A. Fraser [20] y G.E.P. Box [10]. La *técnica Box*, denominada *Evolutionary Operation*, consistía en elegir una serie de parámetros que definían un proceso industrial, modificarlos ligeramente en base a una ley predefinida, probar durante un tiempo dichos parámetros y, al final del periodo de pruebas, valorar la combinación por medio de un comité humano. Básicamente, lo que se estaba aplicando era mutación y selección. El proceso fue aplicado con éxito en algunas industrias químicas.

A mediados de la década de los 60 se introducen los *algoritmos evolutivos*, tal y como se conocen e investigan hoy en día, en los trabajos de L. Fogel, A. Owens y H. Walsh [19]. Cabe destacar entre ellos los procesos introducidos por Fogel para la obtención de autómatas que fuesen capaces de predecir regularidades en cadenas de símbolos. De esta forma, se consiguen hacer evolucionar autómatas que predecían números primos (de forma más o menos fiable), demostrando el potencial que tenía la evolución como método de búsqueda de soluciones alternativas y novedosas.

Los algoritmos evolutivos trabajan con una población de individuos que evolucionan de acuerdo con unas reglas de selección y de otros operadores tales como *recombinación* y *mutación*. Durante ese proceso, cada individuo va siendo procesado de forma dinámica recibiendo una medida de su *adecuación* o *adaptación* al entorno. Los procesos de selección centran su atención en los individuos altamente adaptados, pudiendo alterar los individuos inadaptados mediante procesos de recombinación o mutación. Estos algorimos, que desde un punto de vista biológico parecen muy simples, proporcionan potentes mecanismos de búsqueda.

Los *algoritmos genéticos*, propiamente dichos, fueron introducidos por J. Holland [24] en 1975 inspirándose también en el proceso observado en la evolución natural de los seres vivos, como culminación de las ideas desarrolladas en el curso *Teoría de sistemas adaptativos*, que impartió en la Universidad de Michigan. Años más tarde, D. Goldberg, actual impulsor de los algoritmos genéticos, comenzó a trabajar con Holland y aplicó esta técnica al diseño de soluciones a problemas industriales. Como consecuencia del éxito que tuvieron las técnicas aplicadas por Goldberg y sus alumnos, en 1985 se celebraría la primera conferencia específica de algoritmos genéticos, *ICGA*, que se sigue celebrando bianualmente en la actualidad.

Se trata de formular problemas de optimización en términos de mejorar una cierta *función de evaluación* sobre una población de *cromosomas*, representados por cadenas de dígitos, y que evolucionan a través de operaciones genéticas conocidas, como son la recombinación o la mutación (véase por ejemplo [9] para un tratamiento genérico). A primera vista, lo que se realiza es un recorrido aleatorio por el espacio de las posibles soluciones, un planteamiento por fuerza bruta. Esto hace que resulte sorprendente el buen resultado que se obtiene con este tipo de estrategias en multitud de aplicaciones. De hecho, no existe una explicación matemática clara a este fenómeno, pero aún así esto aporta un cierto optimismo a la investigación en computación natural: el hecho de que la Naturaleza (la vida) haya usado ciertas herramientas (a nivel abstracto; es decir, operaciones sobre estructuras de datos) que se han ido perfeccionando con la evolución durante un largo periodo de tiempo parece indicar que dichas herramientas pueden sernos útiles también para nuestros modelos de computación.

Tanto en este caso como en el anterior se parte de una idea simple (red de interconexiones y selección natural, respectivamente) para crear modelos teóricos que aporten nuevas filosofías o estrategias más allá de la algorítmica tradicional. En el plano práctico, estos modelos han dado lugar a sus correspondientes adaptaciones de hardware (redes de procesadores) y de software (programación evolutiva).

1.4.3. Computación Molecular

Dentro del marco computacional, existen disciplinas que en lugar de extrapolar ideas o estrategias del mundo que nos rodea tratan de utilizar efectivamente los medios físicos disponibles en la Naturaleza para llevar a cabo las computaciones. Es el caso de la *Computación Molecular*, que considera las moléculas como sustrato sobre el que ejecutar los procesos mecánicos.

La idea de realizar computaciones a nivel molecular fue propuesta por R. Feynman a finales de la década de los cincuenta, y estudiada en 1973 por C. Bennet [5] al analizar la posibilidad de usar el ácido ribonucleico (ARN) como medio físico (reversible) para implementar computaciones, estableciendo un paralelismo entre el funcionamiento de la enzima polimerasa y la cabeza lectora/escritora de una máquina de Turing. Posteriormente [6] el propio Bennet da una descripción teórica de un computador que usa moléculas tipo ARN para almacenar información y usa enzimas imaginarias para catalizar reacciones sobre dichas moléculas.

En 1987, T. Head [23] describió un modelo teórico (el modelo *splicing*) que estaba basado en la manipulación de lenguajes de cadenas de ADN a través de la operación de recombinación.

Sin embargo, no es hasta finales de 1994 cuando tiene lugar, propiamente, la primera implementación de una computación a nivel molecular. L. Adleman [1] realiza un experimento en el laboratorio que permite resolver una instancia del problema del camino hamiltoniano (en su versión dirigida y con dos nodos distinguidos) usando moléculas de ADN y operaciones moleculares sobre ellas. De esta manera, se trata de rentabilizar la capacidad de las moléculas de ADN para implementar el paralelismo masivo y, al mismo tiempo, de usar dichas moléculas como soporte físico de la información, logrando así una extraordinaria densidad de almacenamiento de información (un gramo de ADN en polvo ocupa un volumen aproximado de 1 cm^3 y es capaz de almacenar información equivalente a un billón de CD's convencionales).

Además, el paralelismo en este caso es realmente masivo: billones de moléculas (a modo de "procesadores") se pueden acomodar en una minúscula probeta o tubo de ensayo; más aún, dado que las operaciones bioquímicas son altamente no deterministas, el control de las operaciones debe plantearse desde un nuevo enfoque. Con

este modelo conseguimos un dispositivo de computación muy "barato" en cuanto a unidades de control requeridas y a energía disipada se refiere.

El experimento de Adleman causó un tremendo impacto, sorprendiendo por su extraordinaria simplicidad: todas las operaciones usadas eran operaciones biológicas estándares, habitualmente utilizadas en el laboratorio. Así nació una nueva y excitante área: la *Computación Molecular basada en ADN*.

Otro ejemplo de este espíritu de trasladar las computaciones desde el ordenador al laboratorio lo constituye la *Computación Cuántica*, aunque esta rama aún está en fase inicial, y no existe consenso científico acerca de su inclusión o no dentro de la Computación Natural.

Por último, vamos a ocuparnos de la *Computación Celular con Membranas*, el área más reciente de la Computación Natural, inspirada en la estructura y el funcionamiento de las células como organismos vivos. Este es el marco donde se sitúa el presente trabajo, y por ello nos detendremos algo más en su presentación.

1.4.4. Computación Celular con Membranas

Al buscar en la Naturaleza viva esquemas de comportamiento o de organización que nos puedan inspirar desde el punto de vista computacional, parece razonable centrar nuestra atención en la célula. Es la unidad más pequeña en cuanto a forma de vida se refiere, y la actividad que ocurre en su interior puede ser sin duda considerada como procesamiento de información, en cierto sentido. Esto nos conduce al concepto de Computación Celular con Membranas.

Se trata de un modelo de computación orientado a máquinas; es decir, se trabaja con dispositivos cuya ejecución, al modo de las máquinas de Turing, modifica el contenido de las distintas componentes que lo integran hasta llegar, en su caso, a un estado de parada, en el que el sistema deja de funcionar. En este sentido, dicha ejecución podría decirse que es independiente del usuario puesto que, una vez construido el sistema, no es necesario, en principio, dirigirlo.

En esta línea, para llegar a los *sistemas de membranas* (también conocidos como *sistemas celulares* o *sistemas P*), introducidos a finales de 1998 por Gh. Păun [37], hay que abstraer el complicado mundo interior de la célula (Figura 1.1) simplificándolo hasta reducirlo a un sencillo sistema de compartimentos. Al fin y al cabo, toda célula consta en su interior de varias vesículas, dentro de las cuales tienen lugar reacciones que pueden transformar los compuestos químicos presentes, o bien pueden originar un flujo de materiales de un compartimento a otro, atravesando la membrana que los separa.

Los dispositivos computacionales de este nuevo modelo se denominan *sistemas*

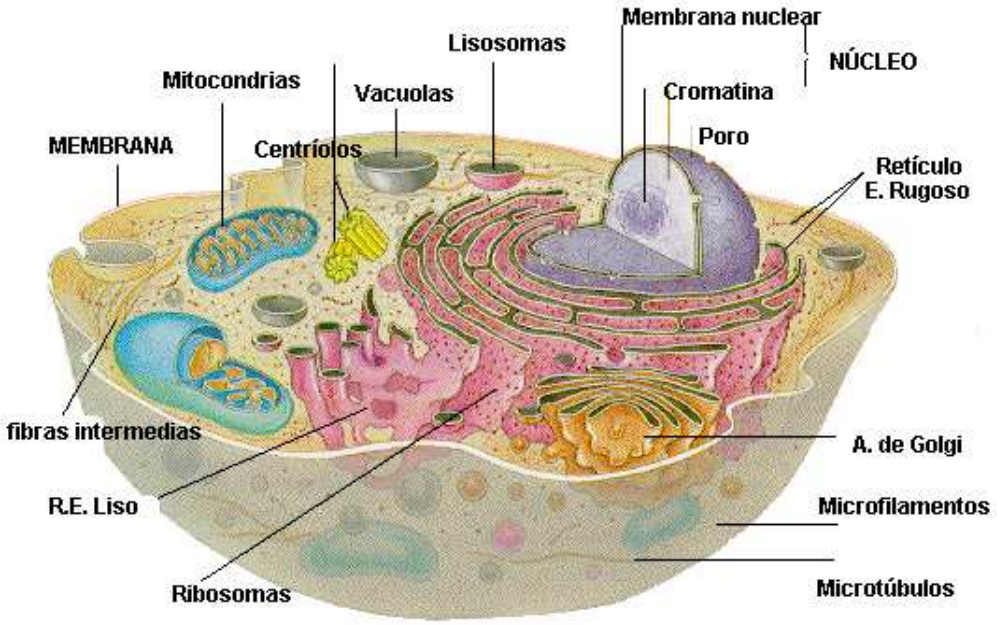

Figura 1.1: *La célula*

P, y tienen como ingrediente fundamental una colección jerarquizada de membranas, todas ellas encerradas en el interior de otra mayor, que hace de la célula un todo separándola del exterior y será por ello llamada *membrana piel*. Dentro de cada membrana podemos encontrar ciertos elementos químicos u *objetos* que pueden aparecer repetidos, pueden moverse atravesando las membranas y pueden ser transformados por la acción de *reglas de evolución*. A continuación, vamos a detallar un poco más estas ideas.

Básicamente, como se ilustra en la Figura 1.2, un sistema P consta de un conjunto de *membranas*, organizado jerárquicamente en una *estructura de membranas*. Existe una *membrana piel* que engloba todas las demás, separando al sistema del *entorno externo*. Las membranas que no contienen otras membranas en su interior se denominan *membranas elementales*. Las *regiones* delimitadas por las membranas (es decir, el espacio acotado por una membrana y las membranas inmediatamente interiores, si existe alguna) pueden contener ciertos *objetos* que, en general, se permite que aparezcan repetidos (y que son abstracciones de los compuestos químicos presentes en las vesículas). Mediante la aplicación de determinadas *reglas de evolución* asociadas a las membranas (que representan las reacciones químicas entre los distintos compuestos), estos objetos pueden transformarse en otros, e incluso pueden pasar de una región a otra adyacente, atravesando la membrana que las separa.

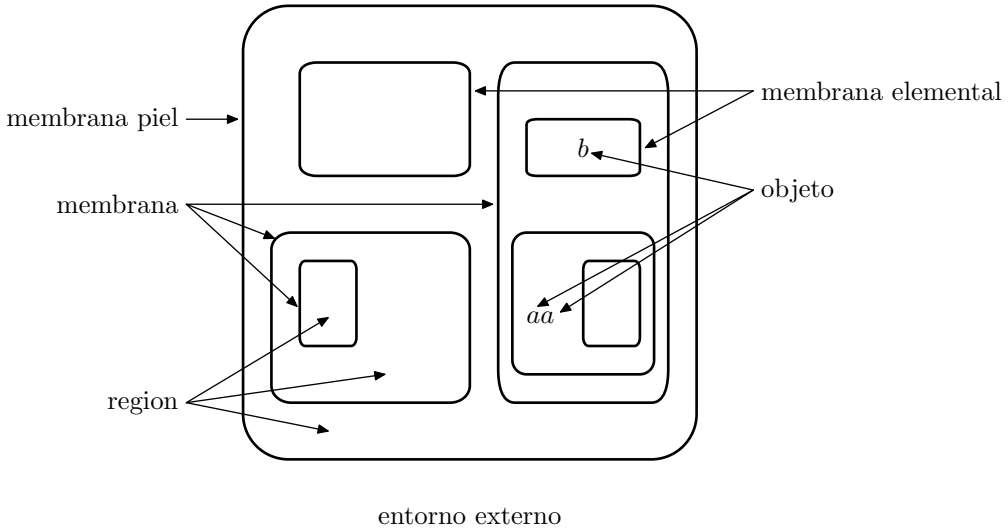

Figura 1.2: *Sistema de computación celular con membranas*

En la versión más simple de este modelo, los sistemas P de transición, se considera que las reglas de evolución están asociadas a las membranas, lo cual representa que en distintas vesículas de una célula pueden tener lugar diferentes tipos de reacciones químicas. Además, se considera que las reglas pueden incorporar *indicadores de destino*, y así los productos de la aplicación de la regla pueden permanecer en la misma región en la que se encontraban los reactivos (*here*), o bien pueden ser enviados a otra región, tanto hacia afuera (*out*), a *la* membrana inmediatamente exterior, como hacia dentro (*in*), a *una* membrana inmediatamente interior. Los casos extremos se dan cuando un objeto en la membrana piel recibe la indicación *out*, con lo cual es expulsado al entorno externo, no pudiendo ser recuperado por el sistema, y cuando en una membrana elemental tenemos una regla que asigna a uno de los productos la indicación *in*, en cuyo caso dicha regla no se puede ejecutar.

Obsérvese que las membranas del sistema actúan tanto de separadores como de canales de comunicación entre regiones, pero son sujetos pasivos en el proceso, la responsabilidad del funcionamiento del sistema recae exclusivamente sobre las reglas de evolución.

Dentro del estudio de los sistemas celulares con membranas existen, básicamente, dos líneas de investigación. La primera se refiere al análisis de la potencia computacional de las distintas variantes consideradas (centradas en la obtención de propiedades *minimales*, en cierto sentido, que permitan asegurar la *completitud computacional* de los modelos). Existen muchos trabajos en esta dirección que demuestran la completitud computacional de variantes que recurren a muy pocas especificaciones. Cabe

destacar entre dichos trabajos el resultado obtenido por O.H. Ibarra en [25], donde se presenta una jerarquía infinita de sistemas celulares (atendiendo a su potencia computacional).

La segunda línea se refiere a la *eficiencia computacional*, es decir, al estudio de variantes que permitan atacar la resolución de problemas **NP**–completos de manera eficiente, en este marco.

Por supuesto, uno de los mayores retos que hay planteados actualmente en el ámbito de la Computación Natural consiste en encontrar una *buena* implementación efectiva de los sistemas celulares con membranas, al menos en su versión básica de transición.

Capítulo 2

Computación Celular con Membranas

El objetivo de este capítulo es la introducción de un nuevo modelo de computación inspirado en la estructura y el funcionamiento de las células de los organismos vivos: el modelo de computación celular con membranas, del que ya se ha hecho una descripción breve en la última sección del capítulo anterior.

El capítulo está estructurado como sigue: en la primera sección se expone brevemente cuál es la idea que el modelo trata de capturar, con vistas a describir, informalmente, los sistemas P de transición introducidos por Gh. Păun en el artículo fundacional de la disciplina [37], así como algunas de las variantes más relevantes del mismo que han ido apareciendo desde entonces. Las Secciones 2.2 y 2.3 están dedicadas a la introducción de los sistemas celulares *con entrada* y los sistemas *con salida externa*, que proporcionan dispositivos computacionales que permiten atacar la resolución de problemas en modelos celulares. En la Sección 2.4 se introduce una variante de sistemas celulares que permiten analizar la aceptación o el reconocimiento de lenguajes. En las dos últimas secciones se estudian los ingredientes necesarios para introducir clases de complejidad en el marco de la computación celular con membranas, y se define una clase de complejidad polinomial asociada a una familia de sistemas celulares reconocedores.

2.1. Sistemas P de transición

En los últimos años el campo de investigación de la *Computación Natural* ha conocido un enorme desarrollo. Los trabajos dentro de este campo estudian la manera de simular el modo en el que la Naturaleza "calcula", aprehendiendo nuevos paradigmas

y modelos de computación a partir de ella. Como se vió con un poco de detalle en el capítulo anterior, son varias las áreas de la Computación Natural que están ya bien establecidas: las *redes neuronales artificiales*, los *algoritmos genéticos*, la *computación molecular basada en ADN* y la *computación celular con membranas*.

La presente memoria se enmarca dentro de esta última disciplina, que se inspira en la estructura y en el funcionamiento de la célula como organismo vivo capaz de procesar y generar información. En efecto, las células están constituidas por diferentes vesículas, delimitadas por membranas. Dentro de dichas vesículas tienen lugar reacciones químicas que provocan no sólo la transformación de los elementos contenidos en ellas, sino también un flujo de dichos elementos entre las diferentes componentes que integran la célula. Estos procesos a nivel celular pueden ser interpretados como procedimientos de cálculo.

A la hora de diseñar un sistema formal que abstraiga este esquema de funcionamiento existen dos caminos a seguir: podemos describir lo más detalladamente posible los procesos que tienen lugar, con la idea de que el sistema nos proporcione un mayor conocimiento acerca de las células; o bien podemos extraer las principales características conocidas que definen una célula, con la esperanza de obtener un nuevo modelo de computación, sencillo pero potente, que nos ayude a resolver problemas que resultan especialmente complicados en otros modelos más clásicos, desde el punto de vista de la complejidad computacional. Este último camino fue el seguido en octubre de 1998 por Gh. Păun al introducir los *sistemas P de transición* [37], creando así una nueva especialidad dentro de la Computación Natural: la *computación celular con membranas*.

La noción de sistema P deriva directamente de la componente fundamental de una célula: la membrana. Nótese que todas las subestructuras internas que componen una célula (incluso la propia célula) están delimitadas por membranas. No obstante, éstas no generan compartimentos estancos, sino que permiten el paso de compuestos químicos a través de ellas, la mayoría de las veces de forma selectiva. Dentro de esos compartimentos es donde, realmente, se producen las reacciones químicas en la célula.

2.1.1. Sintaxis

Veamos a continuación una descripción, detallada aunque informal, de un modelo de computación celular con membranas, siguiendo las ideas originales de Gh. Păun en [37]: los *sistemas P de transición*.

Comenzamos estableciendo el marco dentro del cual se realizarán las computaciones del modelo: una estructura de membranas. Para ello se define por recursión un lenguaje MS sobre el alfabeto $\{[,]\}$ (es decir, cuyos únicos elementos son los símbolos

de corchetes *abierto* y *cerrado*) como sigue:

1. La cadena [ ] pertenece a MS.
2. Si $\mu_1, \ldots, \mu_k \in MS$, entonces $[\,\mu_1 \ldots \mu_k\,] \in MS$.

Sobre este lenguaje se define una relación de equivalencia, $\sim$, que trata de capturar el hecho de que sólo importa la inclusión de un par de corchetes dentro de otro, pero no es relevante la posición a derecha o a izquierda en la cadena de corchetes (por ejemplo $[\,[\,]\ [[\,]]\,] \sim [\,[[\,]]\ [\,]\,]$). Llamaremos *estructuras de membranas* a las clases de equivalencia de los elementos de MS por la relación $\sim$. Es decir, las estructuras de membranas son los elementos del conjunto cociente $MS/\sim$.

Cada par de corchetes coincidentes que aparezca en μ se denomina *membrana*. El par de corchetes externo se denomina *membrana piel* mientras que los pares de corchetes que no contienen otros en su interior se denominan *membranas elementales* (la relación de inclusión entre membranas se define de manera natural para las clases de equivalencia de $MS/\sim$). El *grado* de μ es el número de membranas que contiene.

Dadas dos membranas m_1 y m_2 de μ tales que la segunda está *contenida* en la primera, diremos que dichas membranas son *adyacentes* si no existe ninguna membrana contenida en m_1 y tal que contiene a m_2. En ese caso también diremos que m_1 es la membrana *padre* de m_2 y que m_2 es una membrana *hija* de m_1.

Las estructuras de membranas pueden ser descritas de manera equivalente a través de árboles enraizados tales que cada membrana es un nodo del árbol, la raíz es la membrana piel, las hojas son las membranas elementales y las membranas adyacentes se corresponden con los nodos del árbol tales que uno es el padre del otro (esta descripción basada en árboles se retomará más adelante, en la Sección 2.3).

A continuación especificamos la *sintaxis* del modelo: un sistema P de transición de grado $p \geq 1$ es una tupla

$$\Pi = \big(\Gamma, \mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p), o_\Pi\big)$$

donde:

- Γ es un alfabeto finito (denominado alfabeto de trabajo).
- μ_Π es una estructura de membranas. Las membranas de μ_Π están etiquetadas de forma unívoca usando los números naturales desde 1 hasta p (esto significa, desde luego, que μ_Π contiene exactamente p membranas que, a partir de ahora, identificamos con su etiqueta).
- $\mathcal{M}_i$ $(1 \leq i \leq n)$ es un multiconjunto finito sobre Γ (eventualmente vacío) asociado a la membrana i del sistema.

- R_i $(1 \leq i \leq n)$ es un conjunto finito (eventualmente vacío) de *reglas de evolución* (también conocidas como reglas de transición) asociadas a la membrana i del sistema. Una regla de evolución es un par (u, v), usualmente representado por $u \rightarrow v$, donde u es una cadena sobre Γ y $v = v'$ ó $v = v'\delta$, siendo v' una cadena sobre $\Gamma \times \big(\{here, out\} \cup \{in_j : 1 \leq j \leq p\}\big)$. Los elementos del conjunto $\{here, out\} \cup \{in_j : 1 \leq j \leq p\}$ se denominan *indicadores de destino*, y el símbolo δ es un *marcador de disolución*.
- ρ_i $(1 \leq i \leq n)$ es un orden parcial estricto sobre el conjunto de reglas R_i, especificando una relación de prioridad sobre las reglas de dicho conjunto.
- o_Π es un número natural entre 1 y p que etiqueta una membrana distinguida del sistema (denominada *membrana de salida*).

Nota: Recordemos que un multiconjunto de objetos es un conjunto en el que se admite que un objeto aparezca repetido. Toda cadena sobre un alfabeto determina, de manera natural, un multiconjunto. Por comodidad, en esta memoria representaremos usualmente los multiconjuntos asociados a las membranas de un sistema P como cadenas sobre el alfabeto de trabajo del sistema, Γ.

2.1.2. Semántica

Para describir la *semántica* del modelo, en primer lugar se introduce el concepto de *configuración* de un sistema P, posteriormente se establece el modo en que se aplican las reglas de evolución (lo que permitirá definir un *paso de transición* del sistema) y, finalmente, se presenta la noción de *computación*.

Informalmente, una *configuración* de un sistema P es una descripción instantánea del mismo, incluyendo su estructura de membranas y los contenidos de éstas. Más concretamente, una *configuración* de Π es una tupla $(\mu, M_{i_1}, \ldots, M_{i_q})$, con $q \leq p$, tal que

- μ es la estructura de membranas del sistema, con la salvedad de que es posible que algunas membranas que estuvieran en la estructura inicial ya no aparezcan, porque hayan sido disueltas durante la computación (si bien ése no puede ser el caso de la membrana piel). Es decir, μ es el árbol enraizado con q nodos, cuya raiz coincide con la de μ_Π, y resulta de este último eliminando los nodos j tales que $j \neq i_1, \ldots, i_q$.
- M_{i_j} es un multiconjunto finito sobre Γ, para cada $j = 1, \ldots, q$.

La *configuración inicial* de Π es la tupla $(\mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p)$.

Hablemos ahora de las reglas de evolución. Intuitivamente, el significado de una regla $u \rightarrow v$ es que los objetos de u se transforman en los objetos del alfabeto Γ

que "aparecen" en v, de tal manera que los nuevos objetos pueden dirigirse a otras membranas, en función de lo que marquen los correspondientes indicadores de destino.

Precisemos un poco más cómo se lleva a cabo la aplicación de una regla en una membrana presente en una configuración. Dada una regla $u \rightarrow v$ asociada a una membrana i de una cierta configuración, los objetos de u son eliminados de dicha membrana (por tanto, para poder aplicar dicha regla debe haber suficientes objetos en la membrana i). Entonces, para cada $(a, \alpha) \in v$, se genera un objeto $a \in \Gamma$ que se coloca en una membrana de acuerdo con su indicador de destino; es decir, si $\alpha = here$, entonces el objeto a permanece en la membrana i; si $\alpha = out$, entonces el objeto a es añadido en la membrana padre de la membrana i (o bien el objeto abandona el sistema si la membrana i es la membrana piel); si $\alpha = in_j$, entonces un objeto a es introducido en la membrana j, siempre que esta última sea una membrana hija de la membrana i (en caso contrario la regla no se puede aplicar). Por último, si $\delta \in v$, entonces la membrana i se disuelve; es decir, es eliminada de la estructura de membranas y, además, sus objetos son enviados a la membrana padre. Las reglas de evolución de una membrana disuelta, así como sus relaciones de prioridad, no volverán a intervenir en las sucesivas transiciones del sistema. Conviene aclarar que la semántica del modelo exige que la membrana piel nunca puede ser disuelta. Es decir, en la piel no se puede aplicar nunca una regla del tipo $u \rightarrow v$ tal que $\delta \in v$.

Así pues, para que una regla $r \equiv u \rightarrow v$ asociada a una membrana i sea *aplicable en una cierta configuración*, deben verificarse las condiciones siguientes:

- la membrana i debe pertenecer a dicha configuración.
- el multiconjunto asociado a la membrana i en dicha configuración debe contener al multiconjunto u.
- si en la cadena v hay algún elemento de la forma (a, in_j), entonces la membrana i debe ser la membrana padre de la membrana j.
- si $\delta \in v$, entonces la membrana i no puede ser la membrana piel.
- no existe ninguna regla asociada a la membrana i que cumpla las condiciones anteriores y tenga mayor prioridad que r.

Como ya se ha dicho, los sistemas P no actúan de forma secuencial, sino que son dispositivos paralelos. De hecho, existen dos niveles de paralelismo en un sistema P: por un lado, todas las membranas evolucionan a la vez y, por otro, en un mismo paso pueden aplicarse simultáneamente varias reglas en la misma membrana (en caso de que hubiera más de una que fuese aplicable), o también una misma regla puede aplicarse varias veces.

De acuerdo con lo dicho anteriormente, las prioridades deben ser tenidas en cuenta antes de aplicar una regla, y se interpretarán en *sentido fuerte*; es decir, dichas relaciones prohíben la aplicación de una regla en una cierta configuración si hay alguna otra de mayor prioridad que se pueda aplicar en esa misma configuración. Una posible interpretación o motivación de esta versión se encuentra en el consumo de energía: suponemos que en cada paso de transición se dispone de una cantidad fija de energía para poder aplicar las reglas, de tal manera que las reglas de prioridad superior consumen tal cantidad de energía que no queda suficiente para poder aplicar las reglas de inferior prioridad.

Además, las reglas se aplican de manera *maximal* en cada paso, en el siguiente sentido: tras la aplicación de las reglas, no deben quedar objetos en ninguna membrana que pudieran haber evolucionado por la acción de alguna regla asociada a esa membrana y que sea aplicable.

A continuación vamos a precisar lo que entenderemos por un *paso de transición*. Dadas dos configuraciones, C y C', de Π, decimos que C' se obtiene a partir de C en un paso de transición, y lo notamos por $C \Rightarrow_{\Pi} C'$, si la segunda configuración es el resultado de aplicar sobre la primera, en paralelo (de manera simultánea), y sobre todas las membranas que aparecen en C a la vez, algunas de las reglas de evolución asociadas a dichas membranas, respetando la condición de aplicación de manera maximal en el sentido antes indicado.

Dado que para cada configuración suele existir más de un (multi)conjunto de reglas aplicables, los pasos de transición se ejecutan de manera *no determinista*; es decir, a lo largo de la computación, una configuración del sistema puede tener más de una configuración siguiente. Si no es posible obtener ninguna configuración siguiente, porque ninguna regla es aplicable en ninguna membrana de la configuración, diremos que la configuración es *de parada*.

Una *computación* $\mathcal{C}$ de un sistema celular Π es una sucesión (finita o infinita) de configuraciones $\{C^i\}_{i<r}$, con $r \in (\mathbb{N} - \{0\}) \cup \{\infty\}$, de Π de manera que:

- C^0 es la configuración inicial de Π.
- $C^i \Rightarrow_{\Pi} C^{i+1}$, para cada $i < r - 1$.
- Si la sucesión es finita (es decir, si $r \in \mathbb{N} - \{0\}$), entonces la última configuración de la sucesión, C^{r-1}, es de parada. En ese caso diremos que la computación $\mathcal{C}$ es *de parada* y que ejecuta $r - 1$ pasos. Si $r = \infty$ (es decir, si la sucesión es infinita), entonces diremos que la computación $\mathcal{C}$ *no para*.

Diremos que una computación $\mathcal{C}$ es *exitosa* si es de parada y, además, la membrana de salida, o_{Π}, aparece en la configuración C^{r-1} como una membrana elemental.

Dada una computación exitosa de Π, puede definirse la salida de dicha computación de diversas maneras; por ejemplo, como el tamaño del multiconjunto asociado a la membrana de salida del sistema en su configuración de parada. En este contexto, un sistema P de transición actúa como un dispositivo que *genera* un conjunto de números naturales: el conjunto de las posibles salidas de todas las computaciones exitosas del sistema.

En el siguiente apartado, al introducir las posibles variantes de sistemas celulares que se pueden definir, se incluyen unos comentarios más genéricos acerca de las posibles interpretaciones de la salida de una computación.

2.1.3. Variantes de sistemas P

Tras la introducción de los sistemas P de transición por Gh. Păun se ha producido un gran auge en el estudio de los sistemas de computación celular con membranas, habiendo aparecido muchas variantes que se pueden clasificar, según su orientación, en tres grandes familias. Unas se centran en la resolución de problemas complejos; es decir, se trata de variantes que buscan ser eficientes en el sentido de lograr mayor potencia computacional con menos ingredientes (número de membranas, tipos de reglas, uso de prioridades, etc.). Podemos encontrar resultados de universalidad con una gran diversidad de variantes, ver por ejemplo [7, 21, 26, 29, 30, 36], entre otros.

Otras tienen como finalidad una mayor aproximación al modelo biológico real en el que se inspiran; es decir, tratan de obtener modelos teóricos para la simulación de sucesos que ocurren realmente en la Naturaleza (ver por ejemplo [3, 8], entre otros).

Por último, otras tratan de atacar la resolución de problemas **NP**–completos generando un espacio de trabajo de tamaño exponencial en tiempo polinomial; de hecho, ésta es la orientación con la que se trabaja en esta memoria. Podemos encontrar trabajos en esta dirección en [38, 44, 54].

Ahora bien, independientemente de su orientación, las variantes de sistemas P se pueden enmarcar en tres grandes grupos, según el aspecto de la célula en el que se haga hincapié:

Reproducción: se usan reglas que permiten crear o duplicar membranas.

Membranas como filtros: las reglas usadas únicamente permiten la comunicación entre membranas, pero no la evolución de los objetos.

Intercomunicación: se consideran redes de membranas en lugar de estructuras en forma de árbol (imitando la idea de los tejidos celulares, las redes neuronales o los autómatas celulares).

Por otra parte, una vez definidas las configuraciones y los pasos de transición para un sistema P, se pueden contemplar algunas variantes de tipo "sintáctico":

Objetos: En algunos casos, en lugar de considerar como objetos básicos del modelo un alfabeto de símbolos, se opta por el uso de objetos con estructura interna, como cadenas sobre un determinado alfabeto o incluso objetos en dos dimensiones (arrays o imágenes), ver [27].

Prioridad: Se puede admitir la existencia o no de relaciones de prioridad entre las reglas del sistema.

Cooperación: Se puede admitir la existencia o no de reglas *cooperativas* (es decir, reglas del tipo $u \rightarrow v$ con $|u| > 1$). Una opción intermedia es permitir el uso de *catalizadores*: se trata de considerar un subconjunto de objetos distinguidos $C \subsetneq \Gamma$ de manera que las reglas del sistema sean de la forma $a \rightarrow v$, o bien de la forma $ca \rightarrow cv$, donde $a \notin C$, $c \in C$, y $v \cap C = \emptyset$.

Disolución: Se ha probado que las reglas de disolución no son imprescindibles para obtener la universalidad del modelo. Aún así, dado que parece una herramienta bastante fuerte, muchas veces al hablar de un modelo se explicita si se están considerando reglas de disolución o no.

Además, siempre se puede especificar, para cualquier modelo de computación celular con membranas, cómo se implementa la interacción con el *usuario*:

- Puede existir una *membrana de entrada* en la que se introducen determinados objetos al inicio de una computación, o bien el sistema puede ser *autónomo* y toda la información necesaria ya la tiene en su configuración inicial.

- La salida de una computación puede recogerse en una membrana específica (de acuerdo con la definición dada inicialmente por Gh. Păun), lo que requiere una "intromisión" del usuario en el sistema, o bien puede recogerse en el *entorno externo* del sistema, con lo que el usuario no necesita conocer nada del interior del mismo.

Finalmente, a la hora de fijar qué consideramos como salida de un sistema P encontramos múltiples posibilidades. Así, podemos establecer que dicho sistema genera conjuntos de números naturales, que genera o acepta lenguajes, que computa funciones, o cualquier otro comportamiento que resulte adecuado a nuestros propósitos.

El modelo de computación de los sistemas celulares puede realizar funciones de distinto carácter. En general, existen tres orientaciones:

Carácter generativo: a partir de una configuración inicial se pueden desarrollar distintas computaciones (de manera no determinista) que producirán distintas salidas. Podemos considerar que el sistema *genera* el conjunto formado por las *salidas* de todas las computaciones (que puede interpretarse como *lenguaje generado*).

Carácter computacional: si en la configuración inicial se encuentra codificado un cierto número, n, y se considera el cardinal del multiconjunto de salida como el resultado de la computación, entonces podemos interpretar que el sistema ha "calculado" una cierta función numérica sobre n.

Carácter de decisión: otra opción es considerar que el alfabeto de salida consta de dos objetos especiales, *yes* y *no*, de manera que esos son los únicos objetos que determinan la respuesta, sin importar la presencia en la membrana de salida de otros elementos del alfabeto de trabajo ni su multiplicidad.

En esta memoria estamos interesados en sistemas celulares de este último tipo, ya que se va a trabajar, básicamente, con problemas de decisión.

En las orientaciones de carácter computacional y en la de decisión se hace uso implícitamente del concepto de sistemas celulares con entrada, ya que el resultado de la computación se interpreta en función de cierta información que será codificada al principio del proceso en la configuración inicial. En efecto, en el primer caso decimos que el sistema ha calculado una cierta función $n \mapsto f(n)$, y en el segundo, que el sistema decide una instancia de un problema de decisión, que de nuevo se hallaba "codificada" en cierta forma en la configuración inicial.

2.2. Sistemas P con entrada

Como acabamos de decir, una de las posibilidades a la hora de especificar una variante de sistemas P es considerar que de algún modo son capaces de admitir información del exterior antes de iniciar la computación (es decir, son capaces de recibir una *entrada*).

Definición 2.1 *Un* sistema P con entrada *es una tupla* (Π, Σ, i_Π)*, donde:*

- Π *es un sistema P, con alfabeto de trabajo* Γ*, que consta de* p *membranas, etiquetadas por* $1, \ldots, p$*, y cuyos multiconjuntos iniciales asociados son* $\mathcal{M}_1, \ldots, \mathcal{M}_p$*, respectivamente.*
- Σ *es un alfabeto (de entrada) estrictamente contenido en* Γ*.*
- *Todos los multiconjuntos iniciales son sobre el alfabeto* $\Gamma - \Sigma$*.*

- i_Π *es la etiqueta de una membrana distinguida (de entrada).*

Definición 2.2 *Sea* (Π, Σ, i_Π) *un sistema P con entrada. Sea* Γ *el alfabeto de trabajo de* Π*,* μ *su estructura de membranas y* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *los multiconjuntos iniciales de* Π*. Sea* m *un multiconjunto sobre* Σ*. La* configuración inicial de (Π, Σ, i_Π) con entrada m *es* $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$*.*

Obsérvese que en una configuración inicial, el multiconjunto de entrada m puede ser "detectado" fácilmente a través del alfabeto de entrada Σ, ya que los elementos de dicho alfabeto sólo pueden pertenecer a m, y no a los multiconjuntos iniciales $\mathcal{M}_i$.

Denotaremos por I_Π al conjunto de todas las posibles entradas para el sistema Π. Es decir, I_Π es una colección de multiconjuntos sobre Σ.

Una computación de un sistema P con entrada m, un multiconjunto sobre Σ, se define de manera natural. La única novedad es que la configuración inicial será ahora la configuración inicial del sistema con entrada m; es decir, asociada al multiconjunto de entrada m.

2.3. Sistemas P con salida externa

Como se ha indicado en la sección anterior, una característica interesante que se puede considerar a la hora de clasificar los sistemas P es el lugar donde se "recoge" la salida de sus computaciones. Recuérdese que en los sistemas P de transición, lo habitual es codificar el resultado de una computación a través del contenido de una cierta membrana elemental distinguida (membrana de salida, o_Π). En esta sección vamos a introducir los sistemas P de transición con salida externa, caracterizados por el hecho de que la salida de las computaciones se recoge en el entorno del sistema. Para ello, asociaremos a la estructura de membranas de un sistema una nueva estructura, que denominaremos *estructura con entorno*.

2.3.1. Estructuras de membranas con entorno

Comenzamos recordando qué se entiende por estructura de membranas en el marco de la computación celular.

Definición 2.3 *Una* estructura de membranas *es un árbol enraizado en el que los nodos se denominan* membranas*, la raíz se denomina* piel *y las hojas se denominan* membranas elementales*.*

El grado *de una estructura de membranas es el número de membranas que contiene (es decir, el número de nodos del árbol).*

Por ejemplo, la estructura de membranas de la Figura 1.2 se corresponde con el árbol enraizado que se muestra en la Figura 2.1.

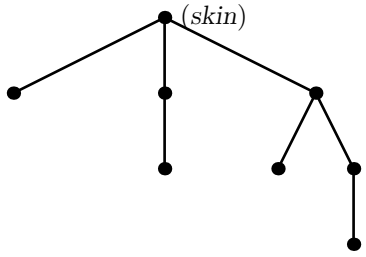

Figura 2.1: *Estructura de membranas*

La membrana piel de una estructura de membranas, a la que nos referiremos genéricamente como *skin*, aísla a la estructura de lo que se conoce como entorno externo de la misma, al que nos referiremos simplemente como *env*. En las variantes de sistemas P que vamos a considerar, la salida de las computaciones se recogerá en dicho entorno. Es por ello que debemos asociarlo de alguna manera a la estructura de membranas.

Definición 2.4 *Sea $\mu = (V(\mu), E(\mu))$ una estructura de membranas, en donde $V(\mu)$ representa el conjunto de nodos del árbol y $E(\mu)$ representa el conjunto de aristas. La* estructura de membranas con entorno externo *asociada a μ es el árbol enraizado $Ext(\mu)$ donde*

- $V(Ext(\mu)) = V(\mu) \cup \{env\}$.
- $E(Ext(\mu)) = E(\mu) \cup \{\{env, skin\}\}$.
- *La raíz del árbol es el nodo env.*

El nuevo nodo se denomina entorno externo *de la estructura μ.*

Obsérvese que lo único que hacemos es añadir un nuevo nodo que representa al entorno externo y que sólo es adyacente a la membrana piel del sistema, mientras que el resto de la estructura de membranas original permanece inalterada. Así, la estructura de membranas con entorno externo asociada a la correspondiente a la Figura 2.1 se muestra en la Figura 2.2.

2.3.2. Sintaxis y Semántica

La sintaxis de un sistema de membranas con salida externa es muy similar a la expuesta en la Subsección 2.1.1 para los sistemas P de transición. Como acabamos de

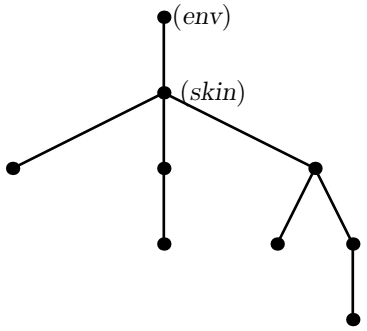

Figura 2.2: *Estructura de membranas con entorno externo*

ver, la estructura de membranas es idéntica salvo por la existencia de un nodo nuevo, *env*, situado en la raíz del árbol. Por tanto, todas las configuraciones del sistema deben incluir la información acerca del contenido del entorno.

Las reglas de transición funcionan exactamente como en el caso estándar, ya que el entorno no cuenta con ninguna regla asociada.

Denotaremos por $C = (\mu, M_{env}, M_{i_1}, \ldots, M_{i_q})$ a las configuraciones de Π, donde $\mu = (V(\mu), E(\mu))$, $V(\mu) = \{i_1, \ldots, i_q\}$, M_{env} es el multiconjunto asociado al entorno externo de μ, y M_{i_j} (con $j = 1, \ldots, q$) es el multiconjunto asociado a la membrana i_j de μ.

A la hora de trabajar con un sistema celular con salida externa, el usuario puede ignorar los procesos internos que ocurren en el sistema, y tener en cuenta solamente los objetos que éste expulse al entorno. Surge entonces una pequeña cuestión: si el usuario es un mero espectador que ve al sistema desde fuera ¿cómo saber si ha parado la computación? y, más aún, ¿cómo determinar el instante de finalización de la misma? Para resolver satisfactoriamente esta cuestión, lo más simple es añadir unos objetos especiales denominados *indicadores de parada*, de tal manera que cuando uno de esos objetos sea expulsado, y sólo entonces, podremos asegurar (desde el exterior) que el sistema ha parado. Desde luego, esta condición complicará un poco el diseño de tales sistemas, pues estarán sujetos a más restricciones.

Diremos que un sistema P de transición con salida externa es *válido* si ninguna computación envía al entorno un objeto indicador de parada en ningún instante, a excepción de las computaciones exitosas, que envían al entorno un único objeto indicador de parada y, además, lo hacen exactamente en el último paso.

En estos sistemas celulares, un usuario situado fuera del sistema puede asegurar que una computación es *exitosa* en el instante en que observa la aparición de algún indicador de parada en el entorno. E incluso analizar el resultado sin preocuparse de lo que haya podido ocurrir en el interior del dispositivo computacional a lo largo de

todo el proceso.

A la hora de interpretar los objetos que son enviados al entorno como salida de la computación, una posiblidad es considerarlos de manera ordenada, según el orden en que van siendo expulsados al entorno. Podemos asociar así a cada computación una cadena de símbolos. En el caso en que varios objetos sean expulsados al entorno en un mismo paso (si la variante de sistemas celulares considerada lo permite), se considerarán todas las posibles permutaciones. De esta manera, podemos definir el *lenguaje generado* por un sistema celular como el conjunto de todas las cadenas asociadas a todas las computaciones exitosas del mismo (si una computación no es exitosa, entonces diremos que no produce ninguna salida).

Normalmente, al diseñar sistemas celulares orientados a la generación de lenguajes, se considera un subconjunto distinguido del alfabeto de trabajo (cuyos elementos se denominarán *terminales*). De tal manera que, a efectos de determinar la cadena asociada a una computación, sólo se toman en consideración los objetos terminales. Cualquier objeto no terminal que sea enviado al entorno durante la computación será ignorado (como si no pudiéramos "leerlo") y, por otra parte, los objetos terminales que permanezcan dentro del sistema cuando la computación pare tampoco serán tenidos en cuenta.

2.4. Sistemas P aceptadores y reconocedores

En esta memoria se usarán los sistemas celulares, básicamente, para atacar la resolución de problemas de decisión. Esto quiere decir que para cada instancia de un tal problema que se quiera resolver, sólo nos interesa conocer una respuesta de tipo binario, de aceptación o rechazo (*Sí* o *No*). Por tanto, el sistema P puede actuar como una especie de *caja negra* a la que el usuario suministra una entrada y de la que recibe una respuesta afirmativa o negativa.

Recordemos que un problema de decisión, X, es un par (I_X, θ_X) donde I_X es un lenguaje sobre un alfabeto finito (cuyos elementos se denominan *instancias*) y θ_X es una función booleana total sobre I_X. Si u es una instancia del problema X tal que $\theta_X(u) = 1$ (resp. $\theta_X(u) = 0$) entonces diremos que la respuesta del problema para esa instancia es *Sí* (resp. *No*).

Por tanto, se puede establecer un paralelismo claro entre la resolución de problemas de decisión y los problemas de reconocimiento de lenguajes: decidir si una instancia de un problema tiene una respuesta afirmativa o negativa es equivalente a decidir si una palabra sobre un cierto alfabeto pertenece o no a un cierto lenguaje sobre dicho alfabeto. En base a esta similitud definiremos una clase especial de sistemas

celulares que usaremos para resolver problemas de decisión, en el marco de la teoría de clases de complejidad.

En la primera sección hemos presentado un marco general para los sistemas celulares. Ahora pasaremos a ver algunas definiciones que van a especificar la clase de sistemas que van a ser utilizados para atacar la resolución de problemas de decisión. Siguiendo la idea de considerar un sistema celular como una *caja negra*, se introduce una variante específica de sistemas P muy apropiada para trabajar con problemas de decisión: los *sistemas reconocedores de lenguajes*. Estos sistemas trabajan de tal manera que cuando se introduce en la membrana de entrada una cadena codificada adecuadamente, el sistema procesa esa cadena y envía un "mensaje" al entorno que representa si dicha cadena pertenece o no a un lenguaje especificado.

Definición 2.5 *Un* sistema P aceptador de lenguajes *es un sistema P con entrada,* (Π, Σ, i_Π)*, y con salida externa, tal que el alfabeto de trabajo tiene dos objetos distinguidos:* Yes *y* No*.*

Pasemos ahora a definir una función *Output* que nos determinará las salidas de las computaciones exitosas de nuestros sistemas celulares. Dada una computación $\mathcal{C} = \{C^i\}_{i<r}$ de un sistema P aceptador de lenguajes, denotaremos por M^i_{env} al contenido del entorno en la configuración C^i.

Definición 2.6 *La* salida *de una computación* $\mathcal{C} = \{C^i\}_{i<r}$*, que notaremos* $Output(\mathcal{C})$*, se define como sigue:*

$$Output(\mathcal{C}) = \begin{cases} Yes, & si\ \mathcal{C}\ es\ de\ parada,\ Yes \in M^{r-1}_{env}\ y\ No \notin M^{r-1}_{env} \\ No, & si\ \mathcal{C}\ es\ de\ parada,\ No \in M^{r-1}_{env}\ y\ Yes \notin M^{r-1}_{env} \\ no\ definida, & en\ otro\ caso \end{cases}$$

Si $\mathcal{C}$ satisface alguna de las dos primeras condiciones, entonces diremos que se trata de una *computación exitosa.*

Definición 2.7 *Diremos que un sistema P aceptador de lenguajes es* válido *si se verifica que en cada computación de parada, y sólo en ellas, se envía al entorno un símbolo* Yes *o bien un símbolo* No *(pero no ambos), y dicho objeto es enviado al exterior en el último paso de la computación.*

Es decir, en este tipo de sistemas celulares los objetos Yes y No actúan de indicadores de parada.

Definición 2.8 *Diremos que una computación $\mathcal{C}$ es de* aceptación *(o de* rechazo*, respectivamente) si un objeto Yes (No, respectivamente) aparece en el entorno externo en la configuración de parada correspondiente a $\mathcal{C}$; es decir, si $Yes = Output(\mathcal{C})$ ($No = Output(\mathcal{C})$, respectivamente).*

Definición 2.9 *Un* sistema P reconocedor de lenguajes *es un sistema P aceptador de lenguajes que es válido y, además, tal que todas sus computaciones paran.*

Estos sistemas reconocedores son una clase de sistemas celulares especialmente apropiados para resolver problemas de decisión.

2.5. Clases de complejidad en sistemas P

A grandes rasgos, estudiar la complejidad computacional de un dispositivo computacional que resuelve un problema consiste en realizar una estimación de los recursos (tiempo, espacio, etc.) que se requieren durante todo el proceso de ejecución del mecanismo, desde la introducción de la instancia del problema hasta la salida final.

Nosotros pretendemos resolver problemas presuntamente intratables (en particular, **NP**-completos) de manera *eficiente*, en el marco de los sistemas celulares con membranas. Es precisamente ese concepto de eficiencia el que nos lleva a la necesidad de considerar clases de complejidad determinadas por problemas de decisión que son resolubles, con una cantidad de recursos similar, por un tipo especial de sistemas de computación celular en tiempo polinomial. Dichas clases de complejidad deben ser cerradas bajo reducibilidad en tiempo polinomial (reducción en sentido clásico), una propiedad que es habitual exigir a las clases de complejidad computacional.

Concretamente, vamos a presentar una clase de complejidad en tiempo polinomial para familias de sistemas P con membrana de entrada, siguiendo las ideas introducidas en [39] y desarrolladas en [43].

Los primeros resultados de "resolubilidad" de problemas **NP**–completos en tiempo polinomial (o incluso lineal) mediante sistemas de computación celular fueron obtenidos utilizando variantes de sistemas P sin membrana de entrada (véanse por ejemplo [38] o [53]). Así pues, en las demostraciones constructivas de dichos resultados se diseñaba *un* sistema para *cada instancia* del problema.

Si se quiere implementar dicha solución en un laboratorio, con este enfoque nos surgiría un grave inconveniente: un sistema P construido para resolver *una* instancia concreta **no** serviría para resolver *cualquier* otra instancia. Este contratiempo puede ser solventado considerando sistemas P con membrana de entrada. En ese caso, un mismo sistema podría resolver un cierto conjunto de instancias del problema (todas

aquellas que tuviesen un tamaño equivalente, en cierto sentido), suministrándole los multiconjuntos de entrada adecuados.

En resumen, en lugar de construir un sistema P asociado a cada instancia concreta, es preferible diseñar una familia de sistemas P tales que cada dispositivo de la familia procese todas las instancias del problema que tienen un cierto tamaño.

A continuación se van a introducir una serie de conceptos básicos necesarios para definir las clases de complejidad. Concretamente, dada una familia de sistemas P, se definen los conceptos de *consistencia* respecto a una clase de sistemas celulares, de *uniformidad por máquinas de Turing* y de *confluencia*. Asimismo se define una herramienta que nos permitirá relacionar un lenguaje con una familia de sistemas celulares, la *codificación polinomial*.

Definición 2.10 *Sea* $\mathbf{\Pi} = \big(\Pi(i)\big)_{i\in I}$ *una familia de sistemas P.*

a) Diremos que la familia $\mathbf{\Pi}$ *es* consistente *respecto a una clase de sistemas celulares,* $\mathcal{D}$*, si se verifica que* $\Pi(i) \in \mathcal{D}$*, para todo* $i \in I$*.*

b) Diremos que la familia $\mathbf{\Pi}$ *es* uniforme *por máquinas de Turing si existe una máquina de Turing determinista de manera que, dado* $i \in I$*, construya el sistema* $\Pi(i)$*. Si esta construcción es además realizada en tiempo polinomial respecto al tamaño de* i*, entonces diremos que la familia* $\mathbf{\Pi}$ *es* polinomialmente uniforme *por máquinas de Turing.*

De acuerdo con la definición general de los sistemas celulares, se tiene que estos modelos de computación son, en general, no deterministas. Por tanto, a priori, no parecen ser una herramienta adecuada para responder a problemas de decisión. Sin embargo, podemos imponer una condición para restringir, en cierto sentido, el no determinismo de las computaciones. Más concretamente, se trabajará en esta memoria con sistemas que sean *confluentes*, en el sentido que se detalla a continuación.

Definición 2.11 *Diremos que un sistema celular,* Π*, es* confluente*, si verifica la siguiente propiedad: todas las computaciones con la misma configuración inicial (y con la misma entrada) devuelven el mismo resultado (tienen la misma salida).*

En concreto, un sistema P aceptador de lenguajes que sea válido es confluente si, o bien todas sus computaciones que tengan la misma configuración inicial son de aceptación, o bien todas ellas son de rechazo. De este modo, para responder a una cierta entrada (es decir, para aceptarla o rechazarla) sólo hay que observar *una* computación asociada a dicho dato de entrada, ya que cualquier otra computación nos dará la misma respuesta.

Ahora bien, dado un lenguaje L y una familia $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$ de sistemas celulares diseñados para decidir la aceptación o rechazo de cadenas de L, es necesario determinar para cada cadena de L cuál es el sistema encargado de procesarla, así como el multiconjunto de entrada que hay que suministrarle para realizar dicha misión. Esto nos conduce al concepto de *codificación polinomial*.

Definición 2.12 *Sea L un lenguaje y sea $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$ una familia de sistemas celulares. Una* codificación polinomial *de L en $\mathbf{\Pi}$ es un par (cod, s) de funciones computables en tiempo polinomial, $cod : L \to \bigcup_{t\in\mathbb{N}} I_{\Pi(t)}$, y $s : L \to \mathbb{N}$ tales que para todo $u \in L$ se tiene que $cod(u) \in I_{\Pi(s(u))}$.*

Es decir, para cada palabra u del lenguaje L, existen un multiconjunto $cod(u)$ y un número $s(u)$ asociado a él, tales que $cod(u)$ es un multiconjunto de entrada para el sistema $\Pi(s(u))$. Obsérvese que todas las palabras $u \in L$ con el mismo valor $s(u)$ tienen asociado el mismo sistema celular; es decir, serán procesadas por el mismo sistema celular (si bien, eventualmente, con distinto multiconjunto de entrada).

Las codificaciones polinomiales son estables bajo reducibilidad en tiempo polinomial, en el siguiente sentido:

Lema 2.1 *Sea $L_1 \subseteq \Sigma_1^*$ y $L_2 \subseteq \Sigma_2^*$ dos lenguajes. Sea $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$ una familia de sistemas celulares. Si $r : \Sigma_1^* \to \Sigma_2^*$ es una reducción en tiempo polinomial de L_1 a L_2, y (cod, s) es una codificación polinomial de L_2 en $\mathbf{\Pi}$, entonces $(cod \circ r, s \circ r)$ es una codificación polinomial de L_1 en $\mathbf{\Pi}$.*

Este resultado se deduce directamente a partir de la definición anterior. Para una prueba detallada, véase [42].

2.6. La clase de complejidad $\mathbf{PMC}_{\mathcal{R}}$

Sea $\mathcal{R}$ una clase de sistemas celulares reconocedores. Seguidamente vamos a introducir la clase de complejidad $\mathbf{PMC}_{\mathcal{R}}$, que englobará a todos los problemas de decisión que son resolubles de manera eficiente a través de sistemas P reconocedores de la clase $\mathcal{R}$.

Definición 2.13 *Diremos que un problema de decisión, $X = (I_X, \theta_X)$, es* resoluble *en tiempo polinomial mediante una familia de sistemas celulares reconocedores de lenguajes, y lo notaremos por $X \in \mathbf{PMC}_{\mathcal{R}}$, si existe una familia de sistemas celulares, $\mathbf{\Pi} = \big(\Pi(t)\big)_{t\in\mathbb{N}}$, que verifica las siguientes propiedades:*

1. *La familia* $\mathbf{\Pi}$ *es consistente, con respecto a la clase* $\mathcal{R}$*; es decir,todos los sistemas de la familia pertenecen a* $\mathcal{R}$*.*

2. *La familia* $\mathbf{\Pi}$ *es polinomialmente uniforme, por máquinas de Turing; es decir, existe una máquina de Turing determinista que permite construir* $\Pi(t)$ *a partir de* t*, en tiempo polinomial.*

3. *Existe una codificación polinomial de* I_X *en* $\mathbf{\Pi}$ *tal que:*

   - *La familia* $\mathbf{\Pi}$ *está polinomialmente acotada, con respecto a* (X, cod, s)*; es decir, existe una función polinomial,* p*, tal que para cada* $u \in I_X$ *todas las computaciones del sistema* $\Pi(s(u))$ *con entrada* $cod(u)$ *son de parada y, además, el número de pasos que efectúan es, a lo sumo,* $p(|u|)$*.*
   - *La familia* $\mathbf{\Pi}$ *es adecuada, con respecto a* (X, cod, s)*; es decir, para cada* $u \in I_X$ *se verifica que si* existe una *computación de aceptación del sistema* $\Pi(s(u))$ *con entrada* $cod(u)$*, entonces* $\theta_X(u) = 1$*.*
   - *La familia* $\mathbf{\Pi}$ *es completa, con respecto a* (X, cod, s)*; es decir, para cada* $u \in I_X$ *se verifica que si* $\theta_X(u) = 1$*, entonces* todas *las computaciones del sistema* $\Pi(s(u))$ *con entrada* $cod(u)$ *son de aceptación.*

Obsérvese que en la definición anterior se ha impuesto implícitamente una condición de *confluencia* a los sistemas P de la familia, en el sentido de la Definición 2.11; esto es, todas las computaciones con la misma configuración inicial y el mismo multiconjunto de entrada deben devolver la misma salida (en este caso Yes o No). Es decir, se ha definido la clase de complejidad $\mathbf{PMC}_{\mathcal{R}}$ en *modo determinista*, en el siguiente sentido: se exige que para *decidir* un multiconjunto de entrada, *todas* las computaciones asociadas del sistema encargado de procesarlo (que es un dispositivo no determinista) deben proporcionar la *misma* salida.

Obsérvese además que, como consecuencia de la definición anterior, la clase de complejidad $\mathbf{PMC}_{\mathcal{R}}$ es cerrada bajo complementario, dado que se utilizan sistemas P reconocedores, en donde todas las computaciones son de parada.

Antes de pasar adelante, nos gustaría aclarar el significado de *tiempo polinomial*. Esa cota para el tiempo se refiere al número de pasos celulares (en donde se usa el paralelismo) que realiza el sistema, pero hemos de tener presente que existe un proceso previo, que podríamos llamar de pre-computación, para obtener $cod(u)$, $s(u)$ y $\Pi(s(u))$ a partir de $u \in I_X$. Este proceso requiere una cantidad polinomial de pasos (de computación convencional y secuencial). Es decir, la resolución de un problema X mediante una familia de sistemas celulares de computación con membrana de entrada consta de dos etapas: en la primera, dada la instancia $u \in I_X$ que se ha de

procesar, necesitamos calcular el número $s(u)$ y el multiconjunto $cod(u)$, y después se ha de construir $\Pi(s(u))$ (computación clásica, pasos secuenciales, *tiempo de precomputación*); en la segunda etapa, se ejecuta el sistema $\Pi(s(u))$ con entrada $cod(u)$ y se recibe la respuesta (computación celular, pasos paralelos, *tiempo de computación*).

Para concluir el capítulo hacemos notar que la clase de complejidad computacional que hemos definido en el marco de los sistemas celulares reconocedores es cerrada bajo reducción en tiempo polinomial.

Proposición 2.1 *Sea $\mathcal{R}$ una clase de sistemas celulares reconocedores. Sean X e Y problemas de decisión tales que X es reducible a Y en tiempo polinomial. Si $Y \in \mathbf{PMC}_{\mathcal{R}}$, entonces $X \in \mathbf{PMC}_{\mathcal{R}}$.*

Es decir, la clase de complejidad $\mathbf{PMC}_{\mathcal{R}}$ es estable bajo reducción en tiempo polinomial. La prueba de este resultado también puede ser encontrada en [42].

En la definición de la clase $\mathbf{PMC}_{\mathcal{R}}$ que hemos presentado, se han usado conceptos relativos a modelos secuenciales convencionales (uniformidad) y a modelos paralelos celulares. Sería interesante dar una definición alternativa de dicha clase en términos exclusivamente de sistemas celulares, introduciendo el concepto de uniformidad a través de estos sistemas. En este contexto, también se podría definir el concepto de reducibilidad mediante sistemas P y demostrar que la clase de complejidad introducida es estable bajo este nuevo tipo de reducción.

Capítulo 3

Sistemas P con membranas activas

Los sistemas P de transición presentan un paralelismo a dos niveles: por un lado, las reglas asociadas a una membrana pueden aplicarse simultáneamente y por otro, en todas las membranas del sistema se realizan operaciones al mismo tiempo.

En este capítulo se introduce una nueva variante de sistemas celulares de computación, los sistemas P con membranas activas [38], que presentan un paralelismo mejorado, en cierto sentido, y que permiten, además, la construcción de un espacio de tamaño exponencial en tiempo polinomial. De esta manera surge la posibilidad de atacar la resolubilidad eficiente de problemas **NP**–completos en el marco de estos nuevos dispositivos computacionales, intercambiando la complejidad en tiempo por complejidad en espacio.

Para la formalización de dicha variante seguiremos la línea desarrollada en [47] y [51]. Describiremos para ello los requisitos que deben verificar los componentes de un sistema celular de computación con división de membranas, formalizando la sintaxis (Subsección 3.3.1) y la semántica (Subsección 3.3.2) de dicho modelo.

3.1. Introducción

En los sistemas P de transición, presentados en el capítulo anterior, el número de membranas puede permanecer invariable o reducirse durante la computación (debido a la posible disolución de alguna membrana). Una alternativa natural consiste en permitir que dicho número pueda también aumentar (lo que está justificado desde el punto de vista biológico, por ejemplo, mediante la creación o división de membranas). Esta idea fue explorada por Gh. Păun [38] dando origen a un nuevo modelo: los

sistemas celulares con membranas activas.

Teniendo presente que se ha demostrado que los sistemas P de transición son un modelo de computación universal (es decir, con la misma potencia computacional que las máquinas de Turing), esta variante no pretende incrementar su potencia computacional, sino que trata de fortalecer la eficiencia computacional (en tiempo) de estos modelos. En efecto, al introducir la división de membranas se produce una aceleración significativa en las computaciones (hasta el punto que en [53] se prueba que si $\mathbf{P} \neq \mathbf{NP}$, entonces un sistema P determinista sin división de membranas no es capaz de resolver ningún problema **NP**–completo en tiempo polinomial). Esta aceleración puede resultar ser especialmente relevante en los casos en que estemos tratando con problemas de la vida real. Por ejemplo, en [26] se muestra un algoritmo de desencriptación para romper el sistema DES (*Data Encryption Standard*) en tiempo lineal usando membranas activas.

3.2. Creación de membranas

En la línea de las ideas comentadas, cabe contemplar la posibilidad de diseñar una variante de sistemas celulares que incluya reglas para la creación de membranas. La creación de membranas es un proceso habitual en Biología; de hecho, es frecuente que se utilice la creación de vesículas para el transporte de sustancias dentro de la célula, y también se ha descrito experimentalmente la aparición, más o menos espontánea, de membranas en el interior de compartimentos especialmente grandes, con el objeto de agrupar cierta clase de reactivos y de facilitar que éstos entren en contacto.

En relación con los sistemas celulares se considera la posibilidad de crear nuevas membranas por la acción de ciertos objetos. En este sentido, se pueden introducir reglas en las que un objeto produce, en el interior de la membrana a la que pertenece, una nueva membrana con un cierto contenido.

Más concretamente, la variante de sistemas celulares con creación de membranas utiliza todos los ingredientes de los sistemas P de transición con la novedad de admitir entre sus reglas algunas del tipo $a \rightarrow [v]_i$, donde a es un objeto del alfabeto de trabajo, v es un multiconjunto de objetos, e i es la etiqueta de una membrana. Si en cierta membrana j existe un objeto a y la regla citada está asociada a dicha membrana, entonces la aplicación de la regla produce el siguiente efecto: se elimina el objeto a de la membrana j y, a la vez, se crea una nueva membrana (hija de la membrana j) con etiqueta i, y cuyo contenido es el multiconjunto de objetos v.

De esta manera, estos sistemas celulares pueden fabricar un número exponencial de membranas en tiempo polinomial y, en consecuencia, son dispositivos computacionales

apropiados para resolver problemas **NP**-completos en tiempo polinomial. Para una descripción más detallada de esta variante, junto con algunos ejemplos de aplicación, ver [39].

3.3. División de membranas

Una de las funciones más importantes de la célula es su capacidad de reproducción, dando lugar a dos nuevas *copias* de sí misma mediante división celular. Este fenómeno biológico recibe el nombre de *mitosis* y, en realidad, consiste en una compleja serie de fases parciales (*interfase*, *profase*, *metafase*, *anafase*, *telofase* y *citocinesis*).

Este proceso permite obtener una cantidad exponencial de células idénticas en tiempo lineal; es decir, en n pasos se pueden generar 2^n copias de una célula, siendo n arbitrariamente grande en un marco teórico ideal (en principio, las únicas restricciones serían de espacio libre y de nutrientes disponibles).

Por supuesto, al diseñar una regla teórica que permita la división de membranas, toda la complejidad del suceso biológico quedará obviada, reduciéndose a lo más básico: de tener una sola membrana pasamos a tener en el paso siguiente dos copias de la misma.

Cabe mencionar aquí que existen otras posibilidades para buscar inspiración en cuanto a la replicación de membranas. Es decir, se podrían formalizar otras vías de reproducción celular: por gemación y por esporulación, en las que una membrana puede dar origen a *varias* nuevas membranas en un solo paso.

A continuación, presentamos una definición formal de los sistemas P con membranas activas que usan división de membranas, tal y como se usarán en esta memoria. Consideraremos únicamente división binaria para membranas elementales, conservándose las etiquetas en el proceso de división, y se usarán reglas sin cooperación y sin relaciones de prioridad entre ellas. Además, dado que se trabajará con sistemas reconocedores de lenguajes (Sección 2.4), y éstos son sistemas celulares con entrada y con salida externa, se utilizará la estructura con entorno, introducida en la Definición 2.4.

En adelante, cuando hablemos de sistemas P con membranas activas, sobreentenderemos que nos referimos a los sistemas celulares con 2-división de membranas elementales.

3.3.1. Sintaxis de los sistemas P con membranas activas

Definición 3.1 *Un* sistema P con membranas activas *es una tupla*

$$\Pi = (\Gamma, H, \mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p, R)$$

donde:

- Γ *es un alfabeto finito (el alfabeto de trabajo).*
- H *es un conjunto finito de etiquetas.*
- μ_Π *es una estructura de membranas de grado* p*, con etiquetas del conjunto* H*. Las membranas poseen carga eléctrica positiva* $(+)$*, negativa* $(-)$ *o neutra* (0)*.*
- $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *son multiconjuntos sobre* Γ*, que expresan los contenidos iniciales de las membranas de* μ*.*
- R *es un conjunto finito de* reglas de evolución*, que pueden ser de cada uno de los tipos siguientes:*

  (*a*) $[a \to v]_l^\alpha$*, donde* $a \in \Gamma$*,* $v \in \Gamma^*$*,* $\alpha \in \{0, +, -\}$ (*reglas de evolución de objetos*)*. Es una regla interna que no modifica nada fuera de la membrana* l*, ni tampoco la carga eléctrica de ésta. Su ejecución produce la sustitución de un objeto* a *por un multiconjunto* v*, dentro de una membrana etiquetada por* l *y con carga* α*.*

  (*b*) $[a]_l^\alpha \to b[\ ]_l^\beta$*, donde* $a, b \in \Gamma$*,* $\alpha, \beta \in \{0, +, -\}$ (*reglas de comunicación*)*. Un objeto* a *puede abandonar una membrana etiquetada por* l *y con carga* α *pasando a la membrana padre, pudiendo transformarse en un nuevo objeto* b *en el proceso. Al mismo tiempo, la ejecución de la regla puede producir un cambio en la carga de la membrana (de* α *a* β*), pero conservándose la etiqueta.*

  (*c*) $a[\ ]_l^\alpha \to [b]_l^\beta$*, donde* $a, b \in \Gamma$*,* $\alpha, \beta \in \{0, +, -\}$ (*reglas de comunicación*)*. Un objeto* a *puede penetrar en una membrana etiquetada por* l *y con carga* α *desde la región inmediatamente superior* (*es decir, desde la membrana padre*)*, pudiendo transformarse en uno nuevo* b *en el proceso y, al mismo tiempo, cambiar la carga de la membrana atravesada (de* α *a* β*), pero conservándose la etiqueta.*

  (*d*) $[a]_l^\alpha \to b$*, donde* $a, b \in \Gamma$*,* $\alpha \in \{0, +, -\}$*,* $l \neq skin$ (*reglas de disolución*)*. Su ejecución produce la transformación de un objeto* a *dentro de una membrana etiquetada por* l *y con carga* α *en otro nuevo* b*. Además, simultáneamente, la membrana es disuelta* (*la piel no se puede disolver*)*.*

(*e*) $[a]_l^\alpha \rightarrow [b]_l^\beta [c]_l^\gamma$, *donde* $a, b, c \in \Gamma$, $\alpha, \beta, \gamma \in \{0, +, -\}$, $l \neq$ *skin* (*reglas de 2-división para membranas elementales*). *Un objeto a situado en una membrana etiquetada por* l *y con carga* α *puede provocar la división de dicha membrana en otras dos, con la misma etiqueta pero, eventualmente, con distintas cargas eléctricas, de tal manera que el objeto a se tranforma de manera independiente en cada una de ellas en nuevos objetos b y c (aparte de esos objetos, los contenidos de las membranas resultantes son idénticos). Este tipo de reglas no pueden ser ejecutadas en la piel.*

Obsérvese que las reglas del sistema están asociadas a etiquetas (es decir, la regla $[a \rightarrow v]_l^\alpha$ está asociada a la etiqueta $l \in H$) y no, propiamente, a las regiones del sistema. Nótese también que las reglas del tipo (*e*) permiten la existencia de varias membranas en el sistema con la misma etiqueta.

Para cada etiqueta $l \in H$, denotaremos por R_l el conjunto de reglas asociadas a la etiqueta l. Por tanto, $R = \bigcup\{R_l : l \in H\}$.
Las reglas se aplicarán de acuerdo con los siguientes principios (semántica informal de los sistemas P con membranas activas):

- Las reglas se usan como es habitual en el marco de la computación con membranas; esto es, de manera paralela y maximal. En cada paso, un objeto en una membrana *sólo* puede participar en la aplicación de *una* regla (elegida de manera no determinista en caso de que hubiera varias posibilidades), pero todo objeto que pueda evolucionar mediante alguna regla, debe hacerlo.

- Si una membrana es disuelta, su contenido (multiconjunto y membranas interiores) pasan a formar parte de la membrana (no disuelta) inmediatamente exterior (esto es, del primer antecesor no disuelto).

- Todos los objetos que no aparecen especificados en ninguna de las reglas que se aplican permanecerán inalterados.

- Una regla de división puede ejecutarse sobre una membrana y, simultáneamente y en el mismo paso, reglas de evolución de objetos se pueden aplicar sobre el contenido de la membrana. En este caso, podemos imaginar que "en primer lugar" evolucionan internamente los objetos y "posteriormente" se produce el proceso de división de la membrana, introduciendo copias de los resultados de dichas evoluciones en las dos membranas creadas (sobreentendiendo claramente que todo lo anterior se produce en un único paso de computación).

- Las reglas asociadas a una etiqueta i se pueden usar para todas las membranas que tengan dicha etiqueta. En el mismo paso se pueden aplicar reglas distintas a

membranas distintas que tengan la misma etiqueta, pero una membrana puede ser objeto de, *a lo sumo*, una regla de entre las que no son de tipo (a).

- La membrana piel no se puede dividir nunca, aunque, como cualquier otra membrana, puede estar polarizada; es decir, cargada eléctricamente.

Obsérvese que en la descripción de esta nueva variante no se admite la cooperación entre objetos para disparar una regla, ni prioridad entre reglas. Además, la carga eléctrica de las membranas puede modificarse por la acción de ciertas reglas, pero no la etiqueta.

3.3.2. Semántica de los sistemas P con membranas activas

A continuación se trata de describir formalmente cómo evoluciona un sistema P con membranas activas, según los multiconjuntos de objetos que se hallen en sus membranas y según los conjuntos de reglas de evolución asociados a las correspondientes etiquetas. Para ello, seguiremos el proceso habitual para definir la semántica de un modelo de computación. Es decir, en primer lugar definiremos qué es una *configuración* o *descripción instantánea* del sistema y, posteriormente, precisaremos qué se entiende por un *paso de computación o transición* de una configuración a otra. Lo que da origen, de manera natural, al concepto de *computación*.

Definición 3.2 *Sea* $\Pi = (\Gamma, H, \mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p, R)$ *un sistema P con membranas activas. Una* configuración *de* Π *es una tupla* $C = \big(Ext(\mu), L, Ch, M\big)$ *que verifica las siguientes condiciones:*

- $\mu = (V(\mu), E(\mu))$ *es una estructura de membranas.*
- *La raíz de* μ *coincide con la raíz de* μ_Π *(estructura de membranas inicial del sistema* Π*).*
- $Ext(\mu)$ *es la estructura de membranas con entorno asociada a la estructura* μ*.*
- L *es una aplicación (no necesariamente inyectiva) cuyo dominio es* $V(\mu)$ *y cuyo rango está contenido en* H*, que asocia una etiqueta (en inglés,* label*) a cada membrana de* μ*.*
- Ch *es una aplicación cuyo dominio es* $V(\mu)$ *y que toma valores en* $\{+, -, 0\}$*, que asocia una carga (en inglés,* charge*) eléctrica (positiva, negativa o neutra) a cada membrana de* μ*.*

- *M es una aplicación cuyo dominio es $V(Ext(\mu))$ y cuyo rango está contenido en $\mathbf{M}(\Gamma)$, que asocia un multiconjunto de objetos sobre Γ (eventualmente vacío) a cada membrana de μ, así como también al entorno externo.*

Usualmente, y a efectos de simplificar la notación, en lugar de dar la aplicación M que a cada membrana le asigna su contenido, se indicará directamente la lista de multiconjuntos asociados a las membranas. Además, la estructura de membranas se representará usualmente haciendo uso de la notación con corchetes (introducida en la Sección 2.1), de manera que se incluya la etiqueta y la carga de cada membrana (es decir, se incluye directamente en μ la información de las funciones L y Ch), como se indica a continuación:

1. Si $V(\mu) = \{i_1\}$, $L(i_1) = l$ y $Ch(i_1) = c$, entonces representaremos $\mu \equiv [\,]_l^c$.

2. Si $i_1 \in V(\mu)$ es la raíz de μ, etiquetada por $L(i_1) = l$ y con carga $Ch(i_1) = c$, y si $\mu_1, \ldots, \mu_k$ son los subárboles principales de cada hijo del nodo i_1, entonces notaremos $\mu \equiv [\mu_1 \ldots \mu_k]_l^c$, donde $\mu_1, \ldots, \mu_k$ están representadas usando la notación con corchetes, etiquetas y cargas.

Por ejemplo, si en una cierta configuración C de un sistema P con membranas activas hay q membranas presentes (supongamos que $V(\mu) = \{i_1, \ldots, i_q\}$ es el conjunto de membranas del sistema), entonces podemos denotar $C = (\mu, M_{env}, M_{i_1}, \ldots, M_{i_q})$ a la configuración de dicho sistema, donde μ es la estructura de membranas expresada mediante la notación con corchetes, etiquetas y cargas, $M_{env} = M(env)$ es el multiconjunto asociado al entorno externo de μ, y $M_{i_j} = M(i_j)$ son los multiconjuntos asociados a las membranas i_j de μ, para todo $j = 1, \ldots, q$.

Nótese que la notación con corchetes incluyendo las etiquetas como subíndices y las cargas eléctricas como superíndices ya se utilizó al presentar los tipos de reglas de los sistemas P con membranas activas en la subsección anterior.

La configuración inicial de un sistema P con membranas activas se define idénticamente a como se hizo para los sistemas de transición en la Subsección 2.1.2: es la descripción del sistema (estructura de membranas y multiconjuntos asociados a las mismas) en el instante inicial, antes de comenzar la computación, sobreentendiendo que inicialmente el entorno está vacío. Es decir, la configuración inicial de Π es $(\mu_\Pi, \emptyset, \mathcal{M}_1, \ldots, \mathcal{M}_p)$.

Recordemos que en el caso de los sistemas con entrada existen infinitas configuraciones iniciales, una por cada multiconjunto de entrada. Asimismo, recordemos que en los sistemas con salida externa también se incluye información acerca del contenido del entorno, tal y como se explicó en la Sección 2.3.

A continuación, antes de definir un paso de transición en un sistema P con membranas activas, vamos a determinar cuándo una regla (o un multiconjunto de reglas) es *aplicable* en una configuración, según los principios expuestos en la Sección 3.3.1.

Definición 3.3 *Diremos que una regla r, asociada a la etiqueta l, es* aplicable en la configuración C en la membrana $i \in V(\mu)$, *y lo denotaremos por $r \in ApR(C, i)$, si $L(i) = l$ y, además, se verifica una de las siguientes situaciones:*

- *Caso 1:* $r = [a \to v]_l^\alpha$ *(regla de tipo* (a)*)*
  - *La membrana i tiene carga eléctrica α; es decir, $Ch(i) = \alpha$.*
  - *La membrana i contiene el objeto necesario para activar la regla; es decir, $a \in M_i$ (multiconjunto de objetos asociado a la membrana i).*
- *Caso 2:* $r = a[\ ]_l^\alpha \to [b]_l^\beta$ *(reglas de tipo* (b)*)*
  - *La membrana i no es la membrana piel.*
  - *La membrana i tiene carga eléctrica α; es decir, $Ch(i) = \alpha$.*
  - *La membrana padre de la membrana i contiene el objeto necesario para activar la regla; es decir, $a \in M_j$ siendo $j \in V(\mu)$ la membrana padre de i.*
- *Caso 3:* $r = [a]_l^\alpha \to [\ ]_l^\beta b$ *(reglas de tipo* (c)*)*
  - *Idéntico al caso 1.*
- *Caso 4:* $r = [a]_l^\alpha \to b$ *(reglas de tipo* (d)*)*
  - *La membrana i no es la membrana piel.*
  - *La membrana i tiene carga eléctrica α; es decir, $Ch(i) = \alpha$.*
  - *La membrana i contiene el objeto necesario para activar la regla; es decir, $a \in M_i$.*
- *Caso 5:* $r = [a]_l^\alpha \to [b]_l^\beta [c]_l^\gamma$ *(reglas de tipo* (e)*)*
  - *La membrana i no es la membrana piel.*
  - *La membrana i es una membrana elemental.*
  - *La membrana i tiene carga eléctrica α; es decir, $Ch(i) = \alpha$.*
  - *La membrana i contiene el objeto necesario para activar la regla; es decir, $a \in M_i$.*

Pasemos ahora a caracterizar las colecciones de reglas que se pueden aplicar al mismo tiempo (en un sólo paso) sobre una configuración, de acuerdo con las ideas expuestas en [38], que establecen que la evolución del sistema se hará de manera paralela y maximal. También es necesario tener en cuenta que las reglas se aplican según los principios enumerados en la sección anterior, resumidos en las siguientes premisas: se pueden aplicar tantas reglas de tipo (a) como sea posible en *una* misma membrana, pero sólo una (a lo sumo) de entre las del resto de los tipos.

Definición 3.4 *Sea* $\Pi = (\Gamma, H, \mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p, R)$ *un sistema P con membranas activas. Sea* $C = \big(Ext(\mu), L, Ch, M\big)$ *una configuración de* Π*. Diremos que un multiconjunto* m *de pares* (r, i)*, donde* $r \in R$ *e* $i \in V(\mu)$*, es un* multiconjunto aplicable de reglas en la configuración C*, y lo denotaremos por* $m \in ApM(C)$*, si verifica las condiciones siguientes:*

- *Contiene al menos una regla; es decir,* $m \neq \emptyset$*.*
- *Todas las reglas contenidas en el multiconjunto son aplicables en* C*, y en su correspondiente membrana asociada; es decir,*

$$\forall (r, i) \in m \ \big(r \in ApR(C, i)\big)$$

- *En el multiconjunto* m *hay, para cada membrana, a lo sumo una regla asociada a ella de entre los tipos* (b)*–*(e)*; es decir, no es posible que existan en* m *dos pares distintos* (r, i) *y* (r', i) *tales que* r *y* r' *no sean ninguna de tipo* (a)*. Dicho de otro modo,*

$$\forall (r, i) \in m \, \forall (r', i) \in m \ \big(r \wedge r' \textit{de tipo } (b)\text{–}(e) \rightarrow r = r'\big)$$

- *Todas las reglas de* m *pueden ser aplicadas simultáneamente; es decir, además de darse las condiciones de cargas correspondientes, hay elementos suficientes para activarlas todas.*
- *Las reglas se aplican de manera maximal; es decir,*

$$\neg \exists m' \big(m' \in ApM(C) \wedge m \subset m'\big)$$

Los distintos tipos de reglas de un sistema P con membranas activas se pueden clasificar en dos grupos, dependiendo del alcance de su aplicación: por un lado, las reglas de tipo (a), (b) o (c) sólo causan la evolución y/o el desplazamiento de un objeto (pudiendo alterar la carga de la membrana); por otro lado, la aplicación de una regla de tipo (d) o (e) modifica la estructura de membranas original.

Debido a esto, y aunque en el modelo se considera que todas las reglas se aplican simultáneamente, a efectos de formalización, dividiremos la ejecución de un multiconjunto aplicable de reglas en tres *micro–pasos*: en primer lugar se tienen en cuenta las reglas de los tipos (a), (b) y (c); en segundo lugar se ejecutan las reglas de división de membranas (esto es, las reglas de tipo (e)); finalmente, se disuelven las membranas que indiquen las reglas de tipo (d).

Así, el paso de una configuración C_1 a otra C_2 se hará por etapas, considerando "configuraciones intermedias" C_1', C_1'' y C_1''' (siendo $C_1''' = C_2$, como veremos en la Definición 3.10). Conviene insistir de nuevo en que este proceso de fragmentación es sólo a efectos de formalización, pero en el modelo no se consideran esas etapas intermedias; es decir, la ejecución de un multiconjunto de reglas aplicable en una configuración corresponderá (en su caso) a *un* paso de computación.

Definición 3.5 *Sea* $C = \big(Ext(\mu), L, Ch, M\big)$ *una configuración de* Π *y sea* m *un multiconjunto aplicable de reglas en* C*. Se definen las aplicaciones* L'_m *y* Ch'_m *sobre* $V(\mu)$ *y la aplicación* M'_m *sobre* $V(Ext(\mu))$ *como sigue:*

- *Las etiquetas no varían; es decir,* $L'_m(i) = L(i)$*, para todo* $i \in V(\mu)$*.*
- $Ch'_m(i) = \begin{cases} \beta, & \text{si existe } (r,i) \in m \text{ tal que } r \in R_{L(i)},\ r \text{ es de tipo } (b) \text{ o } (c), \text{ y se indica dicho cambio de carga en la regla} \\ Ch(i), & \text{en otro caso} \end{cases}$
- *Para las membranas internas* $(i \in V(\mu),\ i \neq skin)$*, el nuevo multiconjunto asociado,* $M'_m(i)$*, se calcula así:*

  1. *Eliminando del multiconjunto* $M(i)$ *los objetos que consumen las reglas de* m *de tipo* (a) *o* (c) *asociadas a* i*, junto con los objetos que consumen las reglas de* m *de tipo* (b) *asociadas a las membranas hijas de* i *en* μ*.*
  2. *Añadiendo al multiconjunto* $M(i)$ *los resultados de la aplicación de las reglas de* m *de tipo* (a) *asociadas a* i*, junto con los objetos que son enviados a la membrana* i*, en su caso, desde su membrana padre (regla de tipo* (b)*) o/y desde sus hijas (reglas de tipo* (c)*).*

- *Para la membrana piel* $(i = skin)$*, se procede igual que en el caso anterior, pero teniendo presente que ahora no aparecerán nuevos objetos en la piel procedentes del entorno.*

- *Por último, para calcular el nuevo multiconjunto del entorno,* $M'_m(env)$*, sólo hay que comprobar si algún objeto ha salido de la piel en este paso (es decir,*

hay que comprobar si había algún par $(r, skin) \in m$ *con* r *de tipo* (c)*) y, en tal caso, añadirlo al anterior multiconjunto.*

Pasemos ahora a estudiar la siguiente etapa intermedia; esto es, analicemos las divisiones de membranas que resultan de la aplicación de las reglas de m de tipo (e). Para ello, se introduce una nueva notación para referirnos al conjunto de membranas afectadas por el proceso de división.

Sea m un multiconjunto aplicable de reglas en la configuración C. El conjunto de *membranas de* C *que son divididas* por la aplicación de m se denotará por $Div(m, C)$.

La nueva estructura de membranas y sus correspondientes multiconjuntos se obtienen como se indica a continuación.

Definición 3.6 *Sea* m *un multiconjunto aplicable de reglas en la configuración* C*. La estructura de membranas* μ''_m*, las aplicaciones* L''_m *y* Ch''_m *sobre* $V(\mu''_m)$ *y la aplicación* M''_m *sobre* $V(Ext(\mu''_m))$ *se definen como sigue:*

- $V(\mu''_m)$ *se obtiene a partir de* $V(\mu)$ *eliminando los nodos de* $Div(m, C)$ *y reponiendo, por cada nodo eliminado (que será una hoja de* μ*), un nuevo par de nodos que se incorporan a la estructura como hijos de la membrana padre de* i *(recordemos que sólo se pueden dividir membranas elementales, luego* i *no puede contener ninguna membrana en su interior).*

- *Las membranas a las que no afecta ninguna regla de división (es decir, todo* $i \in V(\mu)$ *tal que* $i \notin Div(m, C)$*) permanecen inalteradas. Es decir,*

$$L''_m(i) = L'_m(i), \quad Ch''_m(i) = Ch'_m(i), \quad M''_m(i) = M'_m(i)$$

- *Sea* $i \in Div(m, C)$ *una membrana que se divide por la ejecución de una regla* $r \equiv [a]^{\alpha}_{l} \rightarrow [b]^{\beta}_{l}[c]^{\gamma}_{l}$ *(de tipo* (e)*) tal que* $(r, i) \in m$*, y sean* i^1 *e* i^2 *los nodos añadidos en lugar de* i*. Entonces,*

$$\begin{aligned} L''_m(i^1) &= L'_m(i) & \qquad L''_m(i^2) &= L'_m(i) \\ Ch''_m(i^1) &= \beta & \qquad Ch''_m(i^2) &= \gamma \\ M''_m(i^1) &= M'_m(i) - a + b & \qquad M''_m(i^2) &= M'_m(i) - a + c \end{aligned}$$

- *En esta etapa no se modifica el entorno externo; es decir,* $M''_m(env) = M'_m(env)$*.*

Por último, formalicemos el proceso de disolución de membranas. Para ello se introduce una notación para referirnos al conjunto de membranas afectadas por el proceso de disolución. Sea m un multiconjunto aplicable de reglas en la configuración

C. El conjunto de *membranas de C que son disueltas* por la aplicación de m se denotará por $Dis(m, C)$.

Ahora podemos determinar la nueva estructura de membranas.

Definición 3.7 *Sea m un multiconjunto aplicable de reglas en la configuración C. La estructura de membranas μ'''_m se define como sigue:*

- *$V(\mu'''_m)$ se obtiene a partir de $V(\mu''_m)$ eliminando los nodos de $Dis(m, C)$.*
- *Las aristas de la nueva estructura μ'''_m se reajustan convenientemente, desde las hojas a la raíz, de tal manera que:*
  1. *Si una membrana elemental es disuelta, se elimina la arista que la unía con su membrana padre.*
  2. *Las membranas $j \notin Dis(m, C)$ hijas de una membrana $i \in Dis(m, C)$ que se disuelve, pasan a ser hijas del primer antecesor de i que no se disuelva (que existe ya que la raíz nunca se disuelve).*

Definamos, siguiendo las ideas de ese reajuste, una función auxiliar $\overline{ft}_{\mu''_m}$ que nos devuelve, para cada membrana de C, el antecesor más próximo que no es disuelto por la ejecución de m.

$$\overline{ft}_{\mu''_m}(j) = \begin{cases} \textit{no definida}, & \text{si } j = \textit{skin} \\ ft_{\mu''_m}(j), & \text{si } ft_{\mu''_m}(j) \notin Dis(m, C) \\ \overline{ft}_{\mu''_m}(ft_{\mu''_m}(j)), & \text{en cualquier otro caso} \end{cases}$$

Donde $ft_{\mu''_m}(j)$ es la membrana padre de la membrana j en la estructura μ''_m.

Teniendo presente que cuando una membrana se disuelve el multiconjunto de objetos que contiene pasa a la membrana padre, y dado que en un mismo paso puede ser disuelta más de una membrana, necesitamos aclarar cuál es el conjunto de membranas que *donan* su contenido a cada membrana $i \in V(\mu''')$.

Definición 3.8 *Sea m un multiconjunto aplicable de reglas en la configuración C. Para cada membrana $i \in V(\mu'''_m)$ (en particular, $i \notin Dis(m, C)$) el conjunto de* donantes de *i se define como sigue:*

$$Don(i, m, C) = \{j \in V(\mu''_m) : j \in Dis(m, C) \wedge \overline{ft}_{\mu''_m}(j) = i\}$$

Ahora, ya estamos en condiciones de formalizar cómo se lleva a cabo la última etapa intermedia, aclarando la manera de recolocar los objetos tras las disoluciones.

Definición 3.9 *Sea m un multiconjunto aplicable de reglas en la configuración C. Las aplicaciones L'''_m y Ch'''_m sobre $V(\mu'''_m)$ y la aplicación M'''_m sobre $V(Ext(\mu'''_m))$ se definen como sigue:*

- *Las etiquetas y las cargas de las membranas no se modifican en esta etapa:* $L'''_m(i) = L''_m(i)$ *y* $Ch'''_m(i) = Ch''_m(i)$.
- *Dado que la piel no se puede disolver, el contenido del entorno no varía:* $M'''_m(env) = M''_m(env)$.
- *Para toda membrana $i \in V(\mu'''_m)$, se tiene que:*
$$M'''_m(i) = M''_m(i) + \sum_{j \in Don(i,m,C)} M''_m(j).$$

Finalmente, utilizando las etapas intermedias que acabamos de describir, definimos formalmente el concepto de paso de transición para los sistemas P con membranas activas.

Definición 3.10 *Dadas dos configuraciones de un sistema P con membranas activas, $C_1 = \big(Ext(\mu_1), L_1, Ch_1, M_1\big)$ y $C_2 = \big(Ext(\mu_2), L_2, Ch_2, M_2\big)$, diremos que C_2 se obtiene en un paso de transición a partir de C_1, y lo notaremos por $C_1 \Rightarrow_{\Pi} C_2$, si y sólo si existe un multiconjunto m aplicable de reglas en la configuración C_1 tal que:*

$$\mu_2 = {\mu_1}'''_m,\ L_2 = {L_1}'''_m,\ Ch_2 = {Ch_1}'''_m,\ \text{y}\ M_2 = {M_1}'''_m.$$

3.3.3. La clase $\mathcal{AM}$

Recordemos que en la Definición 2.9 se introducían los sistemas celulares reconocedores. Dicha clase de sistemas es especialmente adecuada para resolver problemas de decisión, ya que, dada una entrada, el sistema la acepta o la rechaza (enviando Yes o No al entorno) de manera "fiable", en el marco de las clases de complejidad introducidas en la Definición 2.13. Es decir, aunque el sistema sea un dispositivo no determinista, no es posible obtener respuestas distintas en computaciones diferentes que correspondan al mismo multiconjunto de entrada (los sistemas reconocedores son *confluentes*, en ese sentido).

Las definiciones presentadas en la Sección 2.4 están redactadas en un marco general, pero en el presente trabajo nos ceñiremos al modelo presentado en este capítulo, los sistemas P con membranas activas que usan cargas eléctricas y 2-división de membranas elementales.

Notaremos por $\mathcal{AM}$ a la clase de los sistemas P reconocedores con membranas activas.

El motivo de considerar esta variante concreta de sistemas celulares ya se avanzó en la Introducción del capítulo: además de ser sistemas especialmente adaptados a los problemas de decisión, su capacidad de generar una cantidad exponencial de espacio de trabajo en tiempo polinomial proporciona una aceleración muy significativa a la hora de resolver problemas **NP**–completos.

El proceso de resolución eficiente de dichos problemas en el marco de los sistemas celulares reconocedores con membranas activas, se puede esquematizar como sigue:

1. Inicialmente se dispone de una cantidad polinomial de membranas en el sistema (en función del tamaño del dato de entrada del problema). Mediante la división sucesiva de algunas de ellas se construye una cantidad exponencial de membranas.

2. Con un proceso paralelo, que actúa conjuntamente con la división de membranas, se genera una "biblioteca" de datos de tamaño exponencial. Al mismo tiempo hay un contador que mide el "tiempo" como el número de pasos que ya se han dado y que puede ser útil a efectos de sincronización.

3. Una vez que la biblioteca de datos esté completamente generada, se comprueba si existe alguna solución afirmativa al problema (chequeando todas las posibilidades en paralelo).

4. Se envía un mensaje al entorno para comunicar si la solución del problema es afirmativa o negativa. Para ello hace falta un "mensajero" o intérprete capaz de procesar los resultados de la etapa anterior.

El número de pasos de la fase de división de membranas suele ser de orden lineal, y las tareas de chequeo y de envío de respuesta al exterior podrían ser también lineales, aunque a veces se puede conseguir que sea incluso un número constante de pasos. La Figura 3.1 describe gráficamente el esquema anteriormente descrito.

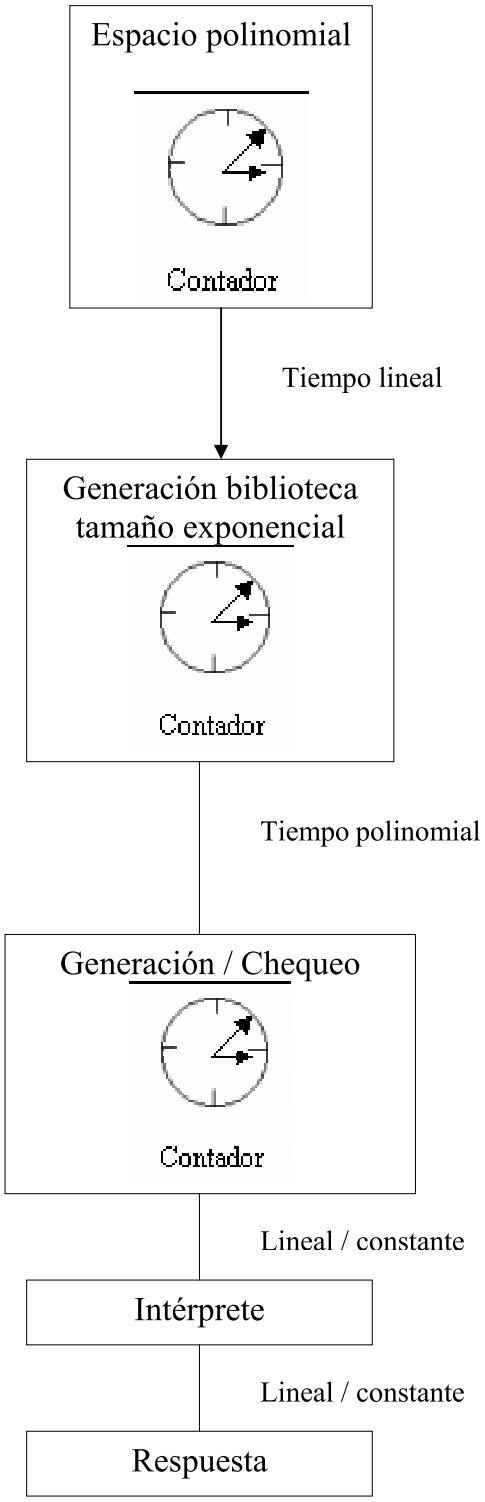

Figura 3.1: Sistema de $\mathcal{AM}$ resolviendo un problema **NP**–completo

Capítulo 4

El problema Subset Sum

En este capítulo se presenta una solución eficiente para el problema Subset Sum en el marco de la computación celular con membranas. En la resolución de este problema numérico **NP**–completo se han utilizado sistemas reconocedores que usan membranas activas, pero sin usar reglas de disolución de membranas. La potencia computacional de la división de membranas elementales, junto con la introducción de algunos contadores específicos, nos ha permitido diseñar una familia de sistemas celulares que resuelve el problema Subset Sum en tiempo (de computación celular, paralelo) lineal, en función del tamaño del dato de entrada (que, recordemos, es codificado en los sistemas celulares a través de multiconjuntos y, por tanto, está representado en forma 1-aria). Sin embargo, se requiere un tiempo (de computación clásica, secuencial) polinomial antes de iniciar propiamente el proceso de computación celular, en concepto de recursos pre-computados.

El capítulo está estructurado como sigue. Tras una breve introducción, se presenta en la Sección 4.2 el diseño de una solución celular para el problema Subset Sum, mediante una familia de sistemas P reconocedores; en la siguiente Sección se describe informalmente el desarrollo de una computación para un sistema genérico de dicha familia (distinguiendo el comportamiento durante las distintas fases de la evolución y la transición entre ellas). Finalmente, en la Sección 4.4, se establece la verificación formal de la solución antes detallada.

4.1. Introducción

Hasta hace poco, los problemas numéricos han recibido poca atención dentro de la comunidad de investigadores en computación celular con membranas. La naturaleza discreta de los sistemas celulares, en los que se trabaja con multiconjuntos, hace muy difícil la representación de números reales, o racionales (magnitudes *contínuas*). En

cuanto al tratamiento de números enteros, una posibilidad es seguir la representación binaria, como se muestra en [4], donde se presenta una implementación de las operaciones aritméticas básicas en sistemas celulares. Otra posibilidad es utilizar una representación 1-aria, donde los números están codificados en el sistema como la multiplicidad de ciertos objetos.

En este trabajo no sólo iniciamos un camino en esa última dirección atacando varios problemas numéricos **NP**–completos, sino que pretendemos alcanzar, a través del estudio de las soluciones presentadas y de las similitudes entre los diseños de las mismas, algún tipo de esquemas "algorítmicos" generales que puedan ser de utilidad en el futuro, a modo de subrutinas, para la resolución de nuevos problemas numéricos.

El diseño presentado en este capítulo proporciona una especie de esquema algorítmico que puede ser adaptado para resolver otros problemas *numéricos* **NP**–completos, como se verá en los capítulos siguientes, donde se abordarán el *problema de la Mochila* y el *problema de la Partición*. También se analizará la posibilidad de extraer características comunes a las distintas soluciones que se describen, con vistas a abordar otros problemas.

Las soluciones que se presentan en este y en los siguientes capítulos difieren de otras soluciones celulares a problemas **NP**–completos dadas por C. Zandron [53], Gh. Păun [38], y otros en el siguiente sentido: en esta memoria se diseña una familia de sistemas P con entrada asociada al problema que se intenta resolver, de manera que *todas* las instancias de dicho problema que tengan el *mismo tamaño* (según un cierto criterio prefijado y computable en tiempo polinomial) son procesadas por el *mismo* sistema (al que se le suministra una entrada apropiada, que dependerá de la instancia concreta y que será computable en tiempo polinomial). Por el contrario, en las soluciones presentadas por los autores antes citados, se asocia a *cada* instancia del problema *un* sistema que se encargará de su procesamiento.

Más concretamente, supongamos que hemos decidido la aceptación o rechazo de una cierta instancia de un problema. Para decidir la aceptación o rechazo de otra instancia del *mismo tamaño* en las soluciones propuestas hasta ahora, habría que modificar, de alguna forma, el sistema celular utilizado para la primera instancia (afectando posiblemente a las reglas de evolución y/o a la estructura de membranas). En las soluciones que se proponen en esta memoria, dicha modificación sería mínima, en tanto en cuanto se reduce a alterar el multiconjunto de entrada, pero no propiamente el "esqueleto" del sistema.

4.2. Una solución celular del problema Subset Sum

El problema Subset Sum puede enunciarse como sigue:

> *Dado un conjunto finito* $A = \{a_1, \ldots, a_n\}$*, donde cada elemento* a_i *tiene asociado un peso,* w_i*, y dada una constante* $k \in \mathbb{N}$*, determinar si existe un subconjunto* $B \subseteq A$ *tal que* $w(B) = k$*.*

Usaremos una tupla $(n, (w_1, \ldots, w_n), k)$ para representar una instancia genérica del problema, siendo n el tamaño del conjunto $A = \{a_1, \ldots, a_n\}$, $(w_1, \ldots, w_n)$ los pesos de los elementos de A, y k la constante que se proporciona como dato de entrada en el problema. Se puede definir de manera natural una función aditiva w sobre A que corresponda a los datos de la instancia.

La solución que se propone está basada en la implementación de un algoritmo de fuerza bruta, en el marco de los sistemas P reconocedores de lenguajes (Definición 2.9) con membranas activas. Para comprender mejor el diseño de la familia que resuelve el problema, estructuraremos su resolución en varias etapas:

- *Fase de generación*: se llevan a cabo divisiones de membranas elementales hasta obtener una membrana específica (y sólo una) para cada subconjunto de A.
- *Fase de cálculo*: en cada membrana se calcula el peso del subconjunto asociado.
- *Fase de chequeo*: se comprueba en cada membrana si el peso del subconjunto asociado coincide o no con la constante k de entrada.
- *Fase de respuesta*: se envía la respuesta al entorno de acuerdo con el resultado de la fase de chequeo.

Como se verá en la siguiente sección, la ejecución de estas fases no se lleva a cabo de manera secuencial y sincronizada. Es decir, etapas correspondientes a fases distintas pueden ser ejecutadas de manera simultánea pero independiente en distintas membranas.

A continuación vamos a construir una familia $\mathbf{\Pi}$ de sistemas P reconocedores de lenguajes que usan membranas activas y que *resuelve* (según la Definición 2.13) el problema Subset Sum en tiempo *lineal*.

En primer lugar, vamos a considerar una función computable a la que se le asignará el papel de *función tamaño*. Consideremos la función s definida sobre el conjunto de instancias del problema Subset Sum, I_{SubS}, como sigue: $s(u) = ((n+k)(n+k+1)/2) + n$, para cada instancia u=$(n, (w_1, \ldots, w_n), k)$.

De este modo, el tamaño definido codifica los dos parámetros clave de la instancia: n y k. El resto de la información (es decir, los pesos de los elementos) es suministrada al sistema a través del multiconjunto de entrada.

Recordemos que la función $\langle m, n\rangle = ((m+n)(m+n+1)/2)+m$ es polinómica, primitiva recursiva y biyectiva de $\mathbb{N}^2$ en $\mathbb{N}$ (su inversa es también polinómica). Así pues, la función s es computable en tiempo polinomial.

Para cada $(n,k) \in \mathbb{N}^2$ consideramos el sistema $(\Pi(\langle n,k\rangle), \Sigma(n,k), i(n,k))$, siendo $\Sigma(n,k) = \{x_1, \ldots, x_n\}$ el alfabeto de entrada, $i(n,k) = e$ la etiqueta de la membrana de entrada, y donde el sistema $\Pi(\langle n,k\rangle)$ viene dado por la tupla $(\Gamma(n,k), \{e,s\}, \mu, \mathcal{M}_s, \mathcal{M}_e, R)$, que describimos a continuación:

- Alfabeto: $\Gamma(n,k) = \Sigma(n,k) \cup \{\bar{a}_0, \bar{a}, a_0, a, d_1, e_0, \ldots, e_n, q, q_0, \ldots, q_{2k+1}, z_0, \ldots, z_{2n+2k+2}, Yes, No_0, No, \#\}$.
- Estructura de membranas: $\mu = [\ [\ ]_e^0\ ]_s^0$.
- Multiconjuntos iniciales: $\mathcal{M}_s = z_0$; $\mathcal{M}_e = e_0\, \bar{a}^k$.
- El conjunto de reglas de evolución, R, consta de las siguientes reglas:

  (*a*) $[e_i]_e^0 \to [q]_e^- [e_i]_e^+$, para $i = 0, \ldots, n$.
  $[e_i]_e^+ \to [e_{i+1}]_e^0 [e_{i+1}]_e^+$, para $i = 0, \ldots, n-1$.
  El objetivo de estas reglas es generar una (única) membrana para cada subconjunto de A. Cuando un objeto e_i $(i < n)$ está presente en una membrana y ésta tiene carga neutra, se selecciona el elemento a_i para el subconjunto asociado y se divide la membrana. En una de las membranas que se crean aparece el objeto q, y esto va a significar que dicha membrana abandona la fase de generación; es decir, ya no se añadirán más objetos al subconjunto asociado y la membrana no se volverá a dividir. Pero la otra membrana creada, en la que aparece el objeto e_{i+1}, continuará en la fase de generación para generar membranas correspondientes a los subconjuntos que se pueden obtener añadiendo al actual otros elementos de A, de índice mayor o igual a $i+1$.

  (*b*) $[x_0 \to \bar{a}_0]_e^0$; $[x_0 \to \lambda]_e^+$; $[x_i \to x_{i-1}]_e^+$, para $i = 1, \ldots, n$.
  Asociado a cada instancia del problema, se añadirá a la membrana de entrada un multiconjunto de objetos que la codifique, de tal manera que las multiplicidades de los objetos x_j (con $1 \le j \le n$) representan los pesos de los correspondientes elementos de A: para cada $a_j \in A$, se introducen w_j copias del objeto x_j. A medida que se van añadiendo elementos al subconjunto asociado a la membrana (como se expuso antes), estas tres reglas se encargan de ir calculando el peso

de dicho subconjunto.

(*c*) $[q \to q_0]_e^-$; $\quad [\bar{a}_0 \to a_0]_e^-$; $\quad [\bar{a} \to a]_e^-$.

La aparición de los objetos q_0, a_0 y a marca el inicio de la fase de chequeo. La multiplicidad del objeto a_0 codifica el peso del subconjunto asociado, y la constante k está codificada a través del número de copias del objeto a.

(*d*) $[a_0]_e^- \to [\ ]_e^0\#$; $\quad [a]_e^0 \to [\ ]_e^-\#$.

Comparamos el número de apariciones de los objetos a y a_0 enviándolos fuera de la membrana uno a uno y cambiando la carga eléctrica a cada paso. Se forma así una especie de *bucle de chequeo* de tal manera que en cada vuelta del mismo, tras aplicar las dos reglas, la situación es idéntica a la inicial pero con una pareja menos de objetos.

(*e*) $[q_{2j} \to q_{2j+1}]_e^-$, para $j = 0, \ldots, k$.

$[q_{2j+1} \to q_{2j+2}]_e^0$, para $j = 0, \ldots, k-1$.

Los objetos q_i actúan de contadores en la fase de chequeo, controlando el número de vueltas que da el *bucle de chequeo* descrito anteriormente.

(*f*) $[q_{2k+1}]_e^- \to [\ ]_e^0 Yes$; $\quad [q_{2k+1}]_e^0 \to [\ ]_e^0\#$.

$[q_{2j+1}]_e^- \to [\ ]_e^-\#$, para $j = 0, \ldots, k-1$.

Estas reglas hacen uso de la información que proporciona el contador para procesar los distintos resultados que pueden obtenerse en la fase de chequeo: que haya el mismo número de objetos a_0 y objetos a, o exceso de los primeros, o exceso de los últimos.

(*g*) $[z_i \to z_{i+1}]_s^0$, para $i = 0, \ldots, 2n + 2k + 1$; $\quad [z_{2n+2k+2} \to d_1 No_0]_s^0$.

Los objetos z_i actúan de contadores dentro de la membrana piel. Su misión consiste en controlar las fases de chequeo de todas las membranas internas; de tal manera que cuando todas ellas hayan terminado sus chequeos, y sólo entonces, se producen (en la misma región) dos objetos especiales, d_1 y No_0.

(*h*) $[d_1]_s^0 \to [\ ]_s^+ d_1$; $\quad [No_0 \to No]_s^+$; $\quad [Yes]_s^+ \to [\ ]_s^0 Yes$; $\quad [No]_s^+ \to [\ ]_s^0 No$.

Estas reglas describen el proceso de respuesta del sistema, que consta de dos pasos. En primer lugar el objeto d_1 altera la carga de la membrana piel, quedando ésta cargada positivamente. En esas condiciones, si hubiera algún objeto Yes en la piel, tendría que ser expulsado al entorno en el paso siguiente, y si no es así, se generará un objeto No para ser enviado fuera como respuesta. Obsérvese que no se producen conflictos, ya que en el primer paso el objeto No aún no

está en la piel (la regla $[No_0 \rightarrow No]_s^+$ necesita carga positiva para aplicarse) y, por tanto, los objetos Yes (si existe alguno) tienen prioridad, en cierto sentido, para salir.

Es fácil probar que la familia de sistemas P que se acaba de definir está formada exclusivamente por sistemas *deterministas*, sin más que estudiar las cabezas de las reglas (junto con las etiquetas asociadas y las condiciones de carga eléctrica) y comprobar que no existen dos reglas que puedan afectar a un mismo objeto en cualquier membrana de una configuración dada. Más aún, probaremos en la Sección 4.4 que la familia $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$ presentada en esta sección resuelve el problema Subset Sum en tiempo *lineal*, de acuerdo con la Definición 2.13.

4.3. Seguimiento informal de la computación

Recordemos que para resolver un problema mediante una familia de sistemas celulares vamos a asociar a cada instancia del problema un multiconjunto (de entrada) y un número ("tamaño" de la instancia) de manera que a la hora de resolver una instancia concreta, se considera el sistema P de la familia que corresponda al tamaño de la instancia, se le introduce el multiconjunto apropiado como entrada, y entonces comienza la computación.

En nuestro caso, dada una instancia $u = (n, (w_1, \ldots, w_n), k)$ del problema Subset Sum, consideraremos la función tamaño introducida en la sección anterior, $s(u) = \langle n, k\rangle$, y consideraremos asimismo una función *cod*, que asigna a la instancia u un multiconjunto sobre el alfabeto de entrada, que para la instancia considerada va a ser $cod(u) = x_1^{w_1} \ldots x_n^{w_n}$, de tal manera que el procesamiento de la instancia u (es decir, decidir si se acepta o se rechaza dicha instancia) se realiza a través del análisis de las computaciones del sistema $\Pi(s(u))$ con entrada $cod(u)$. A continuación describiremos informalmente el desarrollo de las mismas.

En primer lugar vamos a analizar la *fase de generación*. Para comprender mejor el desarrollo de esta fase, se define de manera recursiva el concepto de subconjunto *asociado* a una *membrana interna* (las etiquetadas por e), como sigue:

Definición 4.1 *El* subconjunto asociado a una membrana interna *es:*

- *El conjunto vacío, si la membrana interna es la inicial.*
- *Si en un instante de la ejecución de la fase de generación existe un objeto e_i en una membrana etiquetada por e y con carga neutra, entonces el elemento a_i* se añade *al correspondiente* subconjunto asociado *a dicha membrana.*

Obsérvese que una vez terminada la fase de generación en una membrana interna, el subconjunto asociado a dicha membrana ya no será modificado. También hablaremos, por tanto, de membranas asociadas a subconjuntos.

Los objetos encargados de llevar a cabo la fase de generación son los símbolos e_i. El objetivo es generar una única membrana para cada subconjunto de A, y esto se hará a través de un orden lexicográfico, en cierto modo: si un elemento a_j ya ha sido añadido al subconjunto, entonces no se añadirá al mismo, posteriormente, ningún elemento $a_{j'}$ con $j' < j$.

Al aplicar una regla del tipo $[e_i]_e^0 \rightarrow [q]_e^-[e_i]_e^+$, las dos membranas creadas heredan el subconjunto asociado de la membrana original. La membrana donde aparece el objeto q abandona la fase de generación, y ya no se volverá a dividir en lo que resta de computación (dichas membranas aparecen en la Figura 4.1 señaladas con un círculo). En el siguiente paso, después de haber obtenido carga negativa, se aplican las reglas de (c), renombrando los objetos para preparar la tercera fase. Este es un momento significativo, porque el subconjunto asociado de la membrana ya no variará durante la computación. Por ello, se puede considerar que a partir de ese momento la membrana asume el papel de membrana de trabajo encargada de chequear su subconjunto asociado. Para formalizar esta idea introducimos la definición de membrana relevante.

Definición 4.2 *Toda membrana de una configuración que contiene al objeto q_0 y está polarizada negativamente, así como todas las sucesivas membranas obtenidas a partir de ella por transiciones del sistema, se denominarán* membranas relevantes.

Por otra parte, la otra membrana obtenida al aplicar una regla del tipo $[e_i]_e^0 \rightarrow [q]_e^-[e_i]_e^+$ (en la que aparece un objeto e_i y que tiene carga positiva), seguirá dividiéndose por la aplicación de la regla $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0[e_{i+1}]_e^+$, con objeto de producir membranas asociadas a nuevos subconjuntos, obtenidos añadiendo al subconjunto actual objetos de índice $i+1$ o superior. Obsérvese que en el caso $i = n$ la membrana se queda bloqueada, puesto que no quedan más elementos que añadir (esta circunstancia la hemos señalado con un rombo en la Figura 4.1).

El objetivo de las divisiones que se llevan a cabo al inicio de la computación es obtener una única membrana relevante para cada subconjunto de A. Es decir, en total se obtendrán exactamente 2^n membranas relevantes, todas ellas distintas entre sí.

La *fase de cálculo de pesos* se ejecuta en paralelo con la fase de generación. Veamos a continuación cómo funciona esta fase.

Para hallar el peso de un subconjunto, hay que sumar los pesos de todos sus elementos. El sistema está diseñado de tal manera que la suma de los pesos de los elementos se efectúa al mismo tiempo que dichos elementos van siendo añadidos al subconjunto asociado a la membrana. El proceso continúa hasta llegar a una membrana

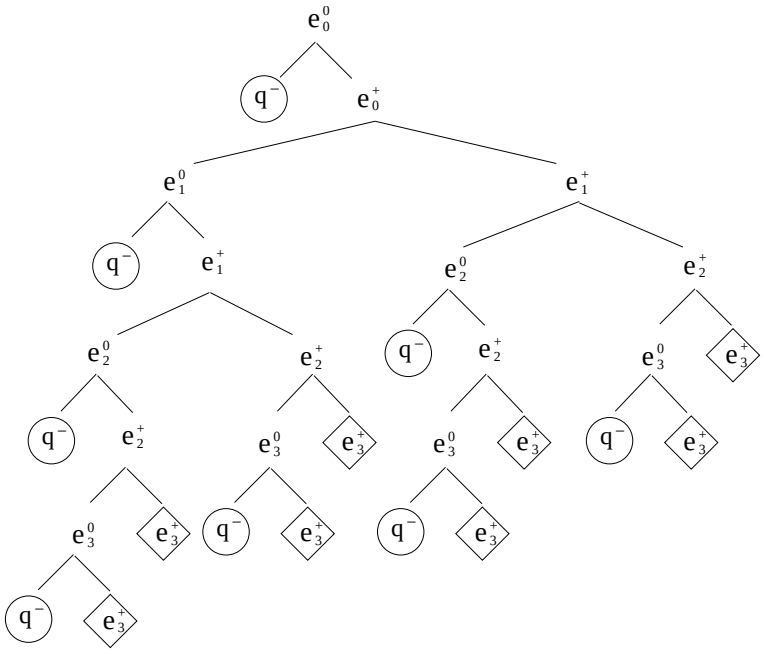

Figura 4.1: Fase de generación para $n = 3$

con carga negativa y, entonces, finalizan en esa membrana las fases de generación y de cálculo. Las reglas de ambas fases están ajustadas de manera que en cada membrana relevante, el número de copias del objeto a_0 coincidirá con el peso del subconjunto asociado.

En efecto, en la siguiente sección se demostrará que en toda membrana con carga positiva (y etiquetada por e) donde aparezca el objeto e_i, la multiplicidad del objeto x_1 será igual al peso del elemento $a_{i+1} \in A$. Por tanto, cuando en el siguiente paso de computación se apliquen las reglas $[x_1 \to x_0]_e^+$, $[x_0 \to \lambda]_e^+$ y $[e_i]_e^+ \to [e_{i+1}]_e^0[e_{i+1}]_e^+$, obtendremos en cada membrana resultante de la división exactamente $w(a_{i+1})$ apariciones de x_0.

Seguidamente, en el caso de la membrana $[e_{i+1}]_e^0$, el elemento a_{i+1} es añadido al subconjunto asociado y en el siguiente paso se producirán $w(a_{i+1})$ nuevas copias de $\bar{a}_0$ en la membrana. La situación es distinta para la otra membrana obtenida en la división ($[e_{i+1}]_e^+$), donde el elemento a_{i+1} no es añadido al subconjunto asociado a la membrana, ni tampoco a los subconjuntos de sus descendientes, así que la información $w(a_{i+1})$ será "borrada" en esta membrana, almacenando en su lugar (como el número de copias del objeto x_0) el valor $w(a_{i+2})$.

De estos comentarios se deduce que, efectivamente, el número de copias del objeto

a_0 coincidirá, en cada membrana relevante, con el peso del subconjunto asociado.

Una vez que se han completado las dos primeras fases, se desarrolla la *fase de chequeo*. Esta fase comienza a ejecutarse en una membrana cuando ésta llega a ser relevante; es decir, esto sucede exactamente un paso después de que aparezca el objeto q y de que la membrana esté cargada negativamente por primera vez en la computación. En ese paso de transición tienen lugar varios renombramientos de objetos (todo ello por la aplicación de las reglas del grupo (c)): $\bar{a}$ y $\bar{a}_0$ pasan a ser a y a_0, respectivamente, y el objeto q se transforma en q_0, con lo que la membrana pasa a ser relevante.

Seguidamente, se comprueba si hay exactamente k objetos a_0 comparando el número de objetos a y de a_0: éstos son contados uno a uno alternativamente, cambiando la carga de la membrana en cada paso, de negativa a neutra y viceversa.

En caso de que la respuesta sea afirmativa, tras $2k$ pasos de la fase de chequeo ya no quedarán más objetos a ni a_0 y la carga de la membrana será negativa. El contador q_j se encarga de contar esos $2k$ pasos, y existen reglas que se ocupan de manejar los casos en los que la respuesta es negativa.

Por último, hay que hacer referencia a los objetos z_j, que están en la membrana piel y se encargan de la *fase de respuesta*. Estos objetos comienzan a evolucionar desde el primer paso de la computación, su tarea es contar $2n + 2k + 2$ pasos de espera por ser éste, precisamente, el número de pasos a realizar, en el caso peor, a fin de que todas las membranas finalicen su fase de chequeo. Una vez que se ha contabilizado ese número de pasos, se puede garantizar que todas las membranas han terminado sus correspondientes fases de chequeo y, por tanto, no queda más que comprobar si alguna de ellas produjo un objeto Yes, y enviar la respuesta adecuada al exterior. Esta comprobación final se lleva a cabo, como ya se comentó en la sección anterior, a través de las reglas del apartado (h).

4.4. Verificación formal

El objetivo de esta sección consiste en demostrar que la familia $\mathbf{\Pi}$ de sistemas celulares presentada en la Sección 4.2 *resuelve* el problema Subset Sum en tiempo polinomial, de acuerdo a la Definición 2.13.

En primer lugar, hay que justificar que la familia definida es $\mathcal{AM}$-consistente; es decir, que todos los sistemas de la familia son sistemas celulares reconocedores de lenguajes que usan membranas activas. De la construcción realizada (por el tipo de reglas y por el alfabeto de trabajo) se sigue directamente que se trata de una familia de sistemas aceptadores de lenguajes con membranas activas. Para probar que los

sistemas de $\mathbf{\Pi}$ son reconocedores de lenguajes basta comprobar que todos son válidos (Definición 2.7) y que todas sus computaciones paran. A lo largo de esta sección veremos esto junto con las restantes condiciones.

4.4.1. Uniformidad polinomial de la familia

A continuación, veamos que la familia $\mathbf{\Pi} = \{\Pi(t) : t \in \mathbb{N}\}$ definida en la Sección 4.2 es polinomialmente uniforme por máquinas de Turing. Es decir, vamos a probar que existe una máquina de Turing determinista que permite, dado $t \in \mathbb{N}$, construir $\Pi(t)$ en tiempo polinomial con respecto al tamaño de t.

Recordemos en primer lugar que para la presentación de los esquemas de reglas que definen a los sistemas de la familia se ha introducido una biyección entre $\mathbb{N}^2$ y $\mathbb{N}$, de manera que el índice, $t \in \mathbb{N}$, de cada elemento de la familia codifica dos únicos valores: n y k. Se puede observar que las reglas de evolución de los sistemas de la familia están definidas de manera recursiva a partir de dichos valores. Además, los recursos necesarios para construir un elemento de la familia son de orden lineal con respecto a los mismos:

- tamaño del alfabeto: $4n + 4k + 17 \in \Theta(n + k)$,
- número inicial de membranas: $2 \in \Theta(1)$,
- número inicial de objetos: $|\mathcal{M}_e| + |\mathcal{M}_s| = k + 2 \in \Theta(k)$,
- suma de las longitudes de las reglas: $35n + 27k + 110 \in \Theta(n + k)$.

Por tanto una máquina de Turing determinista puede construir $\Pi(\langle n, k\rangle)$ en tiempo polinomial con respecto a n y k.

Además, conviene tener presente que cada instancia $u = (n, (w_1, \ldots, w_n), k)$ es introducida en la configuración inicial de su sistema celular asociado mediante un multiconjunto de entrada (es decir, en una representación 1-aria) y, por tanto, se tiene que $|u| \in O(w_1 + \cdots + w_n + k)$.

Antes de proseguir, recordemos que las funciones *cod* y s se han definido en la Sección 4.3 para una instancia $u = (n, (w_1, \ldots, w_n), k)$ del problema Subset Sum como sigue: $cod(u) = x_1^{w_1} \cdots x_n^{w_n}$, y $s(u) = \langle n, k\rangle$, respectivamente. Ambas funciones son computables en tiempo polinomial; más aún, el par (cod, s) es una codificación polinomial de I_{SubS} en $\mathbf{\Pi}$ (ver Definición 2.12), ya que para cada instancia u del problema Subset Sum se tiene que $cod(u)$ es un multiconjunto de entrada del sistema $\Pi(s(u))$.

Para establecer la verificación formal de la familia de sistemas celulares respecto al problema Subset Sum, siguiendo las indicaciones de la Definición 2.13, nos resta probar que todos los sistemas de la familia están polinomialmente acotados y, además,

que son adecuados y completos con respecto a $(SubS, cod, s)$.

4.4.2. Acotación polinomial de la familia

Para cerciorarnos de que el sistema $\Pi(s(u))$ con entrada $cod(u)$ está polinomialmente (de hecho, linealmente) acotado, basta encontrar el instante en que la computación para o, al menos, una cota superior del mismo. Como veremos a continuación, el número de pasos de las computaciones de los sistemas de la familia puede acotarse siempre por una función *lineal*. Sin embargo, conviene mencionar que la cantidad de recursos pre-computados para cada instancia u es polinomial en el tamaño de la instancia, puesto que $cod(u)$ y $s(u)$ necesitan ser calculados y $\Pi(s(u))$ necesita ser construido.

Proposición 4.1 *La familia* $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$*, definida en la Sección 4.2, está polinomialmente acotada respecto a* $(SubS, cod, s)$.

Demostración. Sea $u = (n, (w_1, \ldots, w_n), k)$ una instancia del problema Subset Sum. Analizaremos lo que ocurre durante la computación del sistema encargado de procesar dicha instancia, $\Pi(s(u))$ con entrada $cod(u)$, para encontrar el instante de parada, o al menos una cota superior del mismo.

En primer lugar estudiemos el proceso de división de membranas que ocurre en la *fase de generación*. Obsérvese que una condición necesaria y suficiente para que una membrana se divida es que tenga carga neutra y, además, contenga un objeto e_j con $j \in \{0, 1, \ldots, n\}$, o bien que su carga sea positiva y contenga un objeto e_j con $j \in \{0, 1, \ldots, n-1\}$ (ver reglas del apartado (a)).

En el primer caso se obtendrán una membrana cargada negativamente, donde el objeto e_j es reemplazado por q, y otra membrana con carga positiva, en la que el objeto e_j permanece inalterado (esta membrana se dividirá de nuevo si $j < n$). En el segundo caso también se obtendrán dos nuevas membranas, esta vez con cargas positiva y neutra, respectivamente. En ambas el objeto e_j que produjo la división es reemplazado por un objeto e_{j+1}.

Dado que el conjunto de índices es finito, éstos no pueden aumentar de manera indefinida. Por tanto, la fase de generación (o división de membranas) es un proceso finito. De hecho, teniendo presente que tras dos divisiones consecutivas el índice del objeto e_j aumenta en una unidad (si no es reemplazado por q), se puede deducir que tras $2n + 1$ pasos no tendrán lugar más divisiones.

Pasemos a examinar ahora la *fase de cálculo*. Esta fase tiene lugar durante el proceso de añadir elementos al subconjunto asociado a cada membrana. Si se sigue un razonamiento análogo al anterior, considerando esta vez los objetos x_i (introducidos

en el multiconjunto de entrada $cod(u)$) y observando cómo van desapareciendo o cómo sus índices disminuyen en una unidad por efecto de las reglas del apartado (b), entonces llegamos a la conclusión de que esta fase también da lugar a un proceso finito.

Cuando, como resultado de la aplicación de una regla de división, se genera una membrana etiquetada por e y cargada negativamente, entonces finalizan las dos primeras fases en dicha membrana. Nótese que ya no es aplicable en dicha membrana ninguna de las reglas de los apartados (a) o (b), pero las reglas de (c) sí tienen la condición de carga negativa. Estas son reglas de renombramiento, y se aplican simultáneamente en un sólo paso (en función de la disponibilidad de objetos $\bar{a}$ y $\bar{a}_0$).

En el instante siguiente aparece el objeto q_0, con lo que la membrana llega a ser relevante y comienza en ella la *fase de chequeo*. Sea $B = \{a_{i_1}, \ldots, a_{i_r}\} \subseteq A$ el subconjunto asociado a una membrana relevante, y notemos $w_B = w(B)$. El multiconjunto presente en la membrana en el momento en que pasa a ser relevante será $q_0 a^k a_0^{w_B} x_1^{w_{i_r+1}} \cdots x_{n-i_r}^{w_n}$. Entonces, partiendo de dicha situación, las reglas $[a_0]_e^- \to [\ ]_e^0\#$ y $[a]_e^0 \to [\ ]_e^-\#$ (apartado (d)) serán aplicadas alternativamente hasta que se agote alguno de los objetos a ó a_0. Es claro que este proceso constituye un ciclo finito, ya que cada vez que se completa una vuelta hay un par de objetos menos en la membrana, y no se crea ninguno nuevo.

De manera simultánea al proceso anterior, el contador q_i va evolucionando en la membrana (según las reglas del apartado (e)), y nos proporciona información acerca del paso en el que finaliza el ciclo antes descrito (basta ver el índice que tenga el contador en ese instante). Entre las reglas de (f) sólo una será aplicable en cada membrana, dependiendo de si había un mayor número de objetos a, o bien si había más objetos a_0, o bien si había exactamente la misma cantidad de ambos. Una vez más, dado que disponemos de un número finito de índices para q_i y que éstos van aumentando una unidad en cada paso, de nuevo concluimos que esta fase da lugar a un proceso finito. De hecho, veamos que se verifica el siguiente resultado.

Lema 4.1 *El número máximo de pasos para la fase de chequeo es* $2k + 2$*, y dicha cota se alcanza si* $w(A) \geq k$*.*

Demostración. En primer lugar se aplica k veces el bucle

> "*aplicar* $[a_0]_e^- \to [\ ]_e^0\#$ *junto con* $[q_{2j} \to q_{2j+1}]_e^-$*,*
> *y después aplicar* $[a]_e^0 \to [\ ]_e^-\#$ *junto con* $[q_{2j+1} \to q_{2j+2}]_e^0$ ".

Este proceso ejecuta un total de $2k$ pasos.

Después se ejecutan 2 pasos más, en el primero se aplica la regla $[q_{2k} \to q_{2k+1}]_e^-$ (y puede que también se aplique $[a_0]_e^- \to [\ ]_e^0\#$), y en el segundo se aplica la regla

$[q_{2k+1}]_e^- \rightarrow [\ ]_e^0 Yes$ (o bien la regla $[q_{2k+1}]_e^0 \rightarrow [\ ]_e^0 \#$, si la carga de la membrana es neutra). La cota de $2k+2$ pasos no siempre se alcanza, ya que en los casos en los que el peso del subconjunto asociado es menor que k, los objetos a_0 se agotan antes de poder a aplicar k veces el bucle de comparación.

□

Esto nos lleva a la *fase de respuesta*. En esta fase entra en juego en la membrana piel un nuevo contador, z_j, para garantizar que la respuesta sólo será enviada al entorno una vez que todos los procesos internos en el sistema han terminado. El contador evoluciona según indican las reglas del apartado (g), y se encarga de contabilizar el momento en que termina su fase de chequeo la última de las membranas internas (relevantes).

Lema 4.2 *Cuando la membrana piel obtiene carga positiva, todos los procesos internos han terminado (es decir, ninguna regla es aplicable en las membranas etiquetadas por e).*

Demostración. La membrana piel consigue carga positiva mediante la aplicación de la regla $[d_1]_s^0 \rightarrow [\ ]_s^+ d_1$. Ahora bien, el objeto d_1 aparece tras $2n+2k+3$ pasos de computación: primero el contador z_i evoluciona desde z_0 hasta $z_{2n+2k+2}$ (esto consume $2n+2k+2$ pasos), y luego se aplica la regla $[z_{2n+2k+2} \rightarrow No_0 d_1]_s^0$ (un paso más).

Para probar que tras $2n+2k+4$ pasos de computación (cuando la membrana piel obtiene carga positiva), todos los procesos internos han terminado, basta ver que en ese instante la membrana relevante asociada al conjunto A ya ha terminado su fase de chequeo, puesto que dicha membrana ejecuta un número de pasos máximo en sus tres primeras fases.

Por un lado, la membrana relevante asociada al conjunto A es la que aparece en último lugar, ya que su fase de generación requiere un mayor número de pasos (hay que añadir todos los elementos de A al subconjunto asociado). Dicha membrana se obtiene tras la ejecución de $2n+2$ pasos.

- En primer lugar, para $i = 1, \ldots, n$, se aplica el bucle

  > "*aplicar* $[e_{i-1}]_e^0 \rightarrow [q]_e^- [e_{i-1}]_e^+$ *(la membrana con carga negativa sale del bucle) y luego aplicar* $[e_{i-1}]_e^+ \rightarrow [e_i]_e^0 [e_i]_e^+$ *(sólo nos interesa la membrana con carga neutra, en la otra no se añade* a_i *al subconjunto asociado)*"

  Este proceso se ejecuta en $2n$ pasos.

- Después se ejecutan 2 pasos más, en el primero se lleva a cabo la última división, mediante la regla $[e_n]_e^0 \rightarrow [q]_e^-[e_n]_e^+$, y en el siguiente se realiza el renombramiento del que se habló anteriormente, mediante la regla $[q \rightarrow q_0]_e^-$ y las demás del apartado (c).

Por otro lado, la fase de chequeo de la membrana asociada al conjunto A ejecutará, a lo sumo, $2k+2$ pasos (Lema 4.1). Así que, en total, las fases de generación, cálculo y chequeo de la membrana relevante asociada al conjunto A finalizarán, a lo sumo, tras $2n+2k+4$ pasos de computación.

□

Así pues, si la respuesta es afirmativa, un objeto Yes es enviado al entorno en el paso $2n+2k+5$: la regla $[Yes]_s^+ \rightarrow [\ ]_s^0 Yes$ se aplica un paso después del instante en que la piel recibe carga positiva. En caso de que la respuesta sea negativa, un objeto No es enviado al entorno en el paso $2n+2k+6$: después del instante en que la piel recibe carga positiva se realizan dos pasos más, primero se aplica la regla $[No_0 \rightarrow No]_s^+$ y en el siguiente paso $[No]_s^+ \rightarrow [\ ]_s^0 No$.

Por tanto, existe una cota polinomial (de hecho, *lineal*) para el número de pasos de la computación, respecto a n y k (recuérdese que la codificación del dato de entrada del problema está hecha en forma 1-aria).

□

4.4.3. Adecuación y completitud de la familia

Para probar la adecuación y completitud de la familia $\mathbf{\Pi}$, respecto a $(SubS, cod, s)$, vamos a probar que dada una instancia, u, del problema Subset Sum, el sistema $\Pi(s(u))$ con entrada $cod(u)$ expulsa un objeto Yes si y sólo si la respuesta al problema para la instancia u considerada es afirmativa, y que en caso contrario expulsa un objeto No.

Proposición 4.2 *La familia $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$, definida en la Sección 4.2, es adecuada y completa respecto a $(SubS, cod, s)$.*

Demostración. La prueba se hará estableciendo el correcto funcionamiento de cada fase por separado. En primer lugar, centrémonos en la *fase de generación*. Es necesario probar que a lo largo de la computación se generan membranas (relevantes) de trabajo para todos los subconjuntos de A.

Lema 4.3 (fase de generación) *Dado un subconjunto $B \subseteq A$, en algún momento de la computación aparece una membrana relevante asociada a dicho subconjunto.*

Demostración. Sea $B = \{a_{i_1}, \ldots, a_{i_r}\}$ un subconjunto de A (con $i_1 < i_2 < \cdots < i_r \leq n$). Entonces existe una (única) membrana relevante asociada a él; de hecho, la sucesión de membranas que, a través de la evolución del sistema, conduce a dicha membrana relevante, es la siguiente:

$$[e_0]_e^0 \Rightarrow [e_0]_e^+ \Rightarrow [e_1]_e^+ \Rightarrow \cdots \Rightarrow [e_{i_1-1}]_e^+ \Rightarrow [e_{i_1}]_e^0 \Rightarrow [e_{i_1}]_e^+ \Rightarrow \cdots \Rightarrow$$
$$\Rightarrow [e_{i_2-1}]_e^+ \Rightarrow [e_{i_2}]_e^0 \Rightarrow \cdots \Rightarrow [e_{i_r}]_e^0 \Rightarrow [q]_e^- \Rightarrow [q_0]_e^-$$

Como se puede observar en la definición de subconjunto asociado a una membrana, los cambios de carga eléctrica a lo largo de la computación tienen una semántica asociada, de manera que el subconjunto asociado a una membrana codifica de alguna forma su *historia*. Según esto, la sucesión antes descrita conduce precisamente a la membrana relevante asociada al subconjunto B.

□

A continuación, veamos que el número de ocurrencias de a_0 en dicha membrana es exactamente $w(B)$; es decir, veamos que la fase de cálculo funciona debidamente. Para ello probaremos primero un lema auxiliar.

Lema 4.4 *Supongamos que una membrana etiquetada por e y cargada positivamente contiene un objeto e_i (con $0 \leq i \leq n-1$). Entonces, para cada j tal que $1 \leq j \leq n-i$, la multiplicidad del objeto x_j coincide con el peso del elemento $a_{i+j} \in A$.*

Demostración. Por inducción sobre i.

Para el caso base, $i = 0$, se sabe que el multiconjunto de la membrana etiquetada por e, en la configuración inicial, es $e_0\bar{a}^k x_1^{w_1} \cdots x_n^{w_n}$. Entonces se aplica la regla $[e_0]_e^0 \to [q]_e^-[e_0]_e^+$ (de hecho, esa es la única regla que se puede aplicar sobre ese multiconjunto de objetos) y se obtienen dos membranas: una de ellas con carga negativa, y otra con carga positiva que contiene al multiconjunto de objetos $e_0\bar{a}^k x_1^{w_1} \cdots x_n^{w_n}$. En ese instante, para cada j tal que $1 \leq j \leq n$, la multiplicidad del objeto x_j es exactamente el peso del elemento $a_j \in A$.

Para establecer el paso inductivo, supongamos que el resultado es cierto para $i < n-1$. Consideremos una membrana con carga positiva y etiquetada por e que contenga un objeto e_{i+1}. Distinguiremos dos casos:

1. Supongamos, en primer lugar, que la membrana considerada ha sido obtenida a partir de la aplicación de la regla $[e_i]_e^+ \to [e_{i+1}]_e^0[e_{i+1}]_e^+$. Por hipótesis de inducción se sabe que las multiplicidades de los objetos x_j en la membrana original (en la que aparece e_i) coinciden exactamente con los pesos de los elementos $a_{i+j} \in A$, para todo j tal que $1 \leq j \leq n-i$. Obsérvese que junto con la regla de división también se han aplicado en ese paso las reglas $[x_i \to x_{i-1}]_e^+$, para

$i = 1, \ldots, n$, y $[x_0 \rightarrow \lambda]_e^+$. Por tanto, las multiplicidades de los objetos x_j en nuestra membrana serán iguales a las multiplicidades de los objetos x_{j+1} en la membrana original, para $1 \leq j \leq n$. Así pues, de la hipótesis de inducción se deduce que las multiplicidades de los objetos x_j coinciden exactamente con los pesos de los elementos $a_{i+j+1} \in A$. Esto completa la prueba, en el caso considerado.

2. Supongamos ahora que la membrana considerada se ha creado mediante la aplicación de la regla $[e_j]_e^0 \rightarrow [q]_e^-[e_j]_e^+$, para $j = i+1$. Se puede seguir un razonamiento similar al anterior para la membrana original (en este caso la membrana original es $[e_{i+1}]_e^0$) ya que ésta ha sido obtenida a su vez como resultado de la aplicación de la regla $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0[e_{i+1}]_e^+$. Se concluye entonces que las multiplicidades de los objetos x_j en la membrana original coinciden exactamente con los pesos de los elementos $a_{i+j+1} \in A$, para $j = 1, \ldots, n-i-1$. Por último, téngase presente que la multiplicidad de dichos objetos x_j no varía en el último paso, ya que la única regla que se podría aplicar conjuntamente con la regla de división sería $[x_0 \rightarrow \bar{a}_0]_e^0$, pero ésta no afecta a los objetos x_j con $j \geq 1$.

□

Lema 4.5 (fase de cálculo) *En cada membrana relevante la multiplicidad del objeto a_0 es exactamente el peso del subconjunto asociado a dicha membrana.*

Demostración. Consideremos una membrana relevante arbitraria. Supongamos que el subconjunto asociado es $B = \{a_{i_1}, \ldots, a_{i_r}\}$, con $i_1 < i_2 < \cdots < i_r$, y $r \leq n$. Entonces, tal y como se comentó anteriormente, no hay ni habrá ninguna otra membrana relevante en ningún momento de la computación que represente al mismo subconjunto.

Todas las membranas de la sucesión que conduce a la membrana considerada tienen carga positiva excepto algunas que tienen carga neutra y contienen un objeto e_{i_l}, una para cada elemento $a_{i_l} \in B$ (con $1 \leq l \leq r$).

Aplicando el lema anterior, para $i = i_l - 1$ (con $1 \leq l \leq r$), para $j = 1$, y a todas las membranas con carga positiva, obtenemos que en cada una de ellas, como debe estar presente un objeto e_{i_l-1}, la multiplicidad del objeto x_1 será exactamente w_{i_l}; es decir, el peso del elemento $a_{i_l} \in A$. En el siguiente paso se aplican las reglas de (b) y deducimos por tanto que, para cada l con $1 \leq l \leq r$, en la membrana con carga neutra en la que aparezca e_{i_l} la multiplicidad del objeto x_0 es w_{i_l}. Por tanto, en el siguiente paso w_{i_l} copias del objeto $\bar{a}_0$ serán añadidas al multiconjunto de la membrana.

Esto es válido para todo i_l tal que $1 \leq l \leq r$; esto es, para todos los elementos del conjunto B. Concluimos, por tanto, que en la membrana con carga negativa en la que

aparece el objeto q, la multiplicidad de $\bar{a}_0$ es la suma de los pesos de los elementos de B; es decir, el peso del conjunto B.

Por último, dado que las reglas del apartado (c) se aplican en el último paso de la sucesión, la multiplicidad del objeto a_0 en la membrana relevante es exactamente $w(B)$.

□

A continuación, pasemos a analizar la fase de chequeo.

Lema 4.6 (fase de chequeo) *Una membrana relevante, asociada a un subconjunto $B \subseteq A$, superará con éxito la fase de chequeo (mandando a continuación un objeto Yes a la piel) si y sólo si se verifica $w_B \leq k$.*

Demostración. Consideremos una membrana relevante cualquiera, y sea $B = \{a_{i_1}, \ldots, a_{i_r}\}$ su subconjunto asociado. Sea $w(B) = w_B$ el peso de dicho subconjunto. Entonces el multiconjunto asociado a dicha membrana es $q_0 a_0^{w_B} a^k x_1^{w_{i_r+1}} \cdots x_{n-i_r}^{w_n}$.

Así pues, se tienen w_B y k ocurrencias de los objetos a_0 y a, respectivamente. El objeto q_0 también está presente en la membrana y pudiera haber asimismo algún objeto x_j, con $j > 0$. Ahora bien, objetos de este tipo no evolucionan más en el resto de la computación, porque las reglas de evolución que les afectan son todas con condición de carga positiva, y la membrana no volverá a tener dicha carga.

Distinguiremos tres casos.

1. Supongamos que $w_B < k$. Al estar la membrana cargada negativamente, en el primer paso sólo se pueden aplicar las reglas $[a_0]_e^- \to [\ ]_e^0\#$ y $[q_0 \to q_1]_e^-$. Entonces, dado que la carga pasa a ser neutra en el siguiente paso, las reglas $[a]_e^0 \to [\ ]_e^-\#$ y $[q_1 \to q_2]_e^0$ serán las que se apliquen a continuación. Esto completa la primera vuelta del *bucle de chequeo*.

   El bucle continuará iterándose hasta agotar los objetos a_0 (recordemos que estamos en el caso $w_B < k$); es decir, llegaremos a una situación en que la membrana tendrá carga negativa y su contenido será el multiconjunto $q_{2w_B} b^{k-w_B} x_1^{w_{i_r+1}} \cdots x_{n-i_r}^{w_n}$. Entonces se aplica la regla $[q_{2w_B} \to q_{2w_B+1}]_e^-$, pero esta vez no habrá ningún objeto a_0 que cambie la carga y, por tanto, se aplicará seguidamente la regla de "rechazo" $[q_{2w_B+1}]_e^- \to [\ ]_e^-\#$.

   A partir de entonces, no se aplicará ninguna regla más en esta membrana, porque los objetos restantes son únicamente a y x_j, y no existen reglas de evolución para ellos con condición de carga negativa.

2. Supongamos ahora que $w_B = k$. En este caso, tras k iteraciones del bucle de chequeo ya no quedarán objetos a ni a_0, y el índice del contador q_i será $i = 2k$.

En el siguiente paso sólo se aplicará la regla $[q_{2k} \to q_{2k+1}]_e^-$, y esto nos lleva a una respuesta afirmativa: $[q_{2k+1}]_e^- \to [\ ]_e^- Yes$.

Entonces no se aplicará ninguna regla más en esta membrana, porque los objetos restantes son únicamente x_j, y la membrana tiene carga negativa.

3. Por último, supongamos que $w_B > k$. Entonces en dicha membrana habrá más objetos a_0 que a. En este caso el bucle de chequeo también realizará k iteraciones, pero al término de las mismas el multiconjunto de objetos presentes en la membrana será $q_{2k} a_0^{w_B - k} x_1^{w_{i_r+1}} \cdots x_{n-i_r}^{w_n}$. Por tanto, esta vez la regla $[q_{2k} \to q_{2k+1}]_e^-$ se aplicará junto con $[a_0]_e^- \to [\ ]_e^0 \#$, y en el siguiente paso la regla de "rechazo" $[q_{2k+1}]_e^0 \to [\ ]_e^0 \#$ da por terminada la fase, porque los objetos restantes son únicamente a_0 y x_j, con $j > 0$, y no existe ninguna regla de evolución para ellos con condición de carga neutra.

□

Para finalizar, veamos que la *fase de respuesta* es adecuada. Es importante resaltar que no se enviará ninguna respuesta al entorno mientras que la membrana piel tenga carga neutra. El objeto d_1 (encargado de cambiar la carga de la piel de neutra a positiva) es, por tanto, imprescindible para poder obtener alguna salida, y el objeto d_1 evoluciona a partir del contador z_j de la membrana piel. Ya se ha dicho que el propósito de este contador es contabilizar el momento en que terminan todos los procesos internos del sistema. También se ha dicho que la última membrana relevante en aparecer es la asociada al subconjunto total, $B = A$, y que ésta aparece en el paso $2n + 2$.

Su fase de chequeo tardará, a lo sumo, $2k + 2$ pasos en el caso peor (cuando se verifica $w_B \geq k$), así que el sistema espera $2n + 2k + 2$ pasos antes de liberar en la piel el objeto d_1 mediante la regla $[z_{2n+2k+2} \to d_1 No_0]_s^0$.

Cuando el objeto d_1 es expulsado al entorno, estamos seguros de que todos los procesos internos han terminado. Por tanto, es el momento de buscar en la piel si hay algún objeto Yes presente. Para ello, el objeto d_1, al salir al entorno, deja la piel del sistema cargada positivamente. En ese momento se aplicará la regla $[No_0 \to No]_s^+$ y, además, en caso de que haya algún objeto Yes presente en la piel (es decir, si la fase de chequeo ha finalizado con resultado afirmativo en alguna membrana elemental), la regla $[Yes]_s^+ \to [\ ]_s^0 Yes$ se aplicará y el sistema parará, ya que no puede ser enviado al entorno el objeto No, por tener la piel, de nuevo, carga neutra. En caso de que no hubiera ningún objetos Yes presente, eso significaría que la instancia que se está considerando tiene solución negativa. De hecho, si no se ha expulsado ningún objeto Yes, entonces la carga de la piel sigue siendo positiva en el siguiente paso y se

aplicará la regla $[No]_s^+ \rightarrow [\ ]_s^0 No$.

Resumiendo, hemos probado que:

1. Para todo subconjunto de A existe una (única) membrana relevante asociada a él que aparece durante la computación.

2. En cada membrana relevante, el peso del subconjunto asociado es calculado correctamente y es comparado con k.

3. El objeto Yes es expulsado al entorno si y sólo si la fase de chequeo termina con resultado afirmativo en alguna de las membranas elementales. En caso contrario, es expulsado al entorno un objeto No.

Por tanto, quedan probadas la adecuación y completitud de la familia de sistemas celulares definida en la Sección 4.2, respecto a $(SubS, cod, s)$.

□

De todo lo expuesto en esta sección, y de acuerdo con la Definición 2.13, se deduce el siguiente resultado:

Teorema 1 *a)* $SubS \in \mathbf{PMC}_{\mathcal{AM}}$.

b) $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

Demostración. Para demostrar *b*), basta hacer las siguientes observaciones: el problema Subset Sum es **NP**–completo, $SubS \in \mathbf{PMC}_{\mathcal{AM}}$ y la clase $\mathbf{PMC}_{\mathcal{AM}}$ es estable bajo reducción polinomial y cerrada bajo complementario.

□

Capítulo 5

El problema Knapsack

En este capítulo se presenta una solución eficiente, en el marco de la computación celular con membranas, para el problema Knapsack (*problema de la Mochila*) en su versión de decisión 0/1 acotada. Se trata de un problema que juega un papel importante en los sistemas de encriptación.

Al igual que en el capítulo anterior, en la resolución de este problema numérico **NP**–completo se han utilizado sistemas P reconocedores con membranas activas, sin permitir el cambio de etiquetas en la aplicación de una regla y, además, sin usar reglas de disolución de membranas.

La potencia computacional de la división de membranas elementales, junto con la introducción de algunos contadores específicos, nos ha permitido diseñar una familia de sistemas celulares que resuelve el problema Knapsack en tiempo (de computación celular, paralelo) lineal, en función del tamaño del dato de entrada (que, recordemos, es codificado en los sistemas celulares a través de multiconjuntos y, por tanto, está representado de forma 1-aria). Sin embargo, se requiere un tiempo (de computación clásica, secuencial) polinomial antes de iniciar propiamente el proceso de computación celular, en concepto de recursos pre-computados.

El capítulo está estructurado como sigue. En primer lugar se introduce el problema Knapsack y se comentan algunas variantes interesantes del mismo. A continuación, se presenta en la Sección 5.2 el diseño de una solución celular para dicho problema, mediante una familia de sistemas P reconocedores, siguiendo la línea desarrollada en la solución del problema Subset Sum presentada en el capítulo anterior. En la Sección 5.3 se describe informalmente el desarrollo de una computación para un sistema genérico de la familia (distinguiendo el comportamiento durante las distintas fases de la evolución y la transición entre ellas). Finalmente, en la Sección 5.4, se establece la verificación formal de la solución presentada.

5.1. Introducción

Como ya se ha comentado, en esta memoria no se busca solamente resolver una serie de problemas numéricos **NP**–completos, sino que se trata de encontrar similitudes entre los diseños de las distintas soluciones celulares con objeto de aprehender métodos o subrutinas que nos puedan ser de utilidad para afrontar la resolución de nuevos problemas en el futuro.

En esta línea, presentamos en este capítulo el diseño de una familia de sistemas P que resuelve otro problema numérico **NP**–completo, el *problema Knapsack*, así como el análisis de los recursos utilizados y la verificación formal de dicha solución, en el marco de las clases de complejidad presentadas en el Capítulo 2.

El problema Knapsack es un problema muy conocido en teoría de la complejidad, en criptografía y en matemáticas aplicadas. Se enuncia normalmente en forma de problema de optimización: *dada una serie de elementos con distintos tamaños y valores, encontrar el conjunto de objetos que tenga valor máximo pero que quepan en una mochila de volumen prefijado.*

Más formalmente:

> *Se tienen una mochila de capacidad $k > 0$ y N objetos. Cada objeto tiene asignado un valor, $v_i > 0$, y un peso, $w_i > 0$. Encontrar una selección de objetos ($\delta_i = 1$ si el objeto se selecciona, 0 si no) que quepa en la mochila (es decir, tal que $\sum_{i=1}^{N} \delta_i w_i \leq k$) y tal que su valor total (es decir, $\sum_{i=1}^{N} \delta_i v_i$) sea máximo.*

Nota: esta versión se suele denominar versión 0/1 o binaria del problema de la Mochila, dado que cada objeto puede ser seleccionado (1), o no (0). También se utiliza la expresión problema de la Mochila acotado (*Bounded Knapsack Problem*) porque hay una cantidad limitada de objetos. La versión que introducimos en este capítulo es la de decisión, que sigue siendo un problema **NP**–completo: *dada otra constante, $c > 0$, determinar si existe alguna selección de objetos que quepa en la mochila y cuyo valor sea mayor o igual que c.*

En la vida real, el problema Knapsack está presente cada vez que se planea efectuar una inversión. Con un presupuesto fijado, hay que seleccionar qué bienes se van a comprar, buscando lograr la mayor rentabilidad (el máximo valor posible para el coste considerado). La imagen de un excursionista que trata de llenar su bolsa de viaje (que tiene una capacidad acotada) con los objetos "más útiles" es la que da nombre al problema.

La formulación más habitual es la versión 0/1, donde un objeto se añade a la mochila o no, sin posibilidades intermedias. Esta restricción hace que el problema sea

difícil, pues si se permite dividir indefinidamente los objetos (versión fraccionaria), entonces existe un sencillo algoritmo voraz que encuentra la selección óptima, y que puede ser descrito informalmente como sigue:

a) se calcula para cada objeto el "precio por kilo", y se coloca la máxima cantidad que quepa del objeto más caro,

b) repetir la operación con el siguiente más caro, hasta que no quede más espacio en la mochila.

Además de la distinción entre versión binaria y versión fraccionaria, vamos a comentar algunos casos particulares del problema Knapsack:

- *Todos los objetos tienen el mismo valor, o todos tienen el mismo peso.* En esta situación especial, el problema se simplifica enormemente, convirtiéndose en un simple problema de ordenación. Si todos los objetos valen lo mismo, el valor máximo se alcanza con el máximo número de objetos seleccionados. Basta entonces ordenar los objetos por peso de menor a mayor y añadirlos en ese orden hasta que la capacidad de la mochila se alcance. Se procede análogamente para el caso de objetos con el mismo peso y distintos valores.

- *Todos los objetos tienen el mismo "precio por kilo".* En este caso, el problema es equivalente a tratar de minimizar el espacio libre en la mochila, ignorando el valor de los objetos. Desafortunadamente, incluso esta versión restringida del problema resulta ser **NP**–completa.

- *Todos los pesos son cantidades relativamente pequeñas.* Cuando tanto los pesos de los objetos como la capacidad, k, de la mochila son números enteros, existe un algoritmo eficiente pseudo–polinomial, diseñado utilizando la técnica de programación dinámica, que encuentra una solución en tiempo $O(nk)$ y espacio $O(k)$. Estas cotas pueden ser más o menos aceptables según sea k. Por ejemplo, la complejidad del algoritmo puede ser asumible para $k \leq 10^3$, pero no serlo para k del orden de 10^7.

- *Disponemos de varias mochilas.* Cuando se dispone de varias mochilas, el problema pasa a ser un problema de empaquetamiento (*Bin Packing*). En este caso el nuevo problema es *sustancialmente* más difícil que la versión original (si $\mathbf{P} \neq \mathbf{NP}$).

Un caso particular muy especial del problema Knapsack, dentro del apartado de "precio por kilo" constante, es el problema de la Partición (*Integer Partition*), que estudiaremos en el siguiente capítulo:

Se trata de encontrar una partición de los elementos de un conjunto S en dos subconjuntos A y B, de tal manera que $\Sigma_{i\in A}s_i = \Sigma_{i\in B}s_i$ o, en la versión de optimización, que la diferencia entre ambos sea mínima. El problema de la Partición puede enfocarse también como un problema de empaquetamiento con dos mochilas o contenedores iguales, o bien como un problema Knapsack donde la capacidad de la mochila coincida con la mitad del peso total. Por tanto, observamos que estos tres problemas (Partición, Bin Packing y Knapsack) están muy relacionados, siendo todos ellos **NP**–completos.

También cabe destacar, entre las instancias del problema de la Mochila (versión 0/1 de decisión) con "precio por kilo" constante, los casos en los que, para cada elemento, el peso coincide con el valor ($w_i = v_i$), y las dos constantes que aparecen en el enunciado son iguales ($k = c$). En esta situación, el problema se puede reformular, equivalentemente, como sigue: *dado un conjunto de números enteros (los pesos de los objetos) y una constante, k, determinar si existe algún subconjunto tal que su suma sea igual a k*. Este problema es conocido como *problema Subset Sum* (o *problema del subconjunto ponderado*), que fue estudiado en el capítulo anterior. Desde este punto de vista, el Subset Sum es un subproblema del problema de la Mochila. De hecho, en el campo de la criptografía se utiliza normalmente la expresión "Knapsack problem" incluso cuando se trata del problema Subset Sum.

Como ya se ha dicho, muchos problemas relacionados directamente con el problema de la Mochila son resueltos mediante la técnica de programación dinámica, proporcionando algoritmos pseudo–polinomiales, aunque no se conoce ningún algoritmo general en tiempo polinomial. Tanto el problema Knapsack general como el problema Subset Sum son **NP**–duros, y esto ha hecho que se hayan realizado múltiples intentos para diseñar códigos de encriptación de clave pública basados en el la complejidad computacional de los citados problemas.

5.2. Una solución celular del problema Knapsack

El problema Knapsack, en su versión de decisión 0/1 acotada, es el siguiente:

> *Dadas dos constantes $k, c \in \mathbb{N}$, y un conjunto finito $A = \{a_1, \ldots, a_n\}$, donde cada elemento tiene asociado un "peso", $w_i \in \mathbb{N}$, y un "valor", $v_i \in \mathbb{N}$, determinar si existe un subconjunto de A tal que su peso no exceda de k y su valor sea mayor o igual que c.*

Representaremos las instancias del problema mediante tuplas de la forma $(n, (w_1, \ldots, w_n), (v_1, \ldots, v_n), k, c)$, donde n es el tamaño del conjunto A; $(w_1, \ldots, w_n)$ y $(v_1, \ldots, v_n)$ son los *pesos* y los *valores*, respectivamente, de los elementos de A;

y, por último, k y c son las constantes que se proporcionan como dato de entrada en el problema. Se pueden definir de manera natural funciones aditivas w y v que correspondan a los datos de la instancia.

Como en el capítulo anterior, afrontamos la resolución del problema basándonos en la implementación de un algoritmo de fuerza bruta, en el marco de los sistemas P reconocedores de lenguajes (Definición 2.9) con membranas activas. Para comprender mejor el diseño de la familia que resuelve el problema, estructuraremos su resolución en varias etapas:

- *Fase de generación*: se llevan a cabo divisiones de membranas elementales hasta obtener una membrana específica (y sólo una) para cada subconjunto de A.
- *Fase de cálculo*: en cada membrana se calculan el peso y el valor del subconjunto asociado.
- *Fase de chequeo para la función "peso" w*: se comprueba en cada membrana si se verifica $w(B) \leq k$, siendo $B \subseteq A$ el subconjunto asociado a dicha membrana.
- *Fase de chequeo para la función "valor" v*: se comprueba en cada membrana si se verifica $v(B) \geq c$, siendo $B \subseteq A$ el subconjunto asociado a dicha membrana.
- *Fase de respuesta*: se envía la respuesta al entorno de acuerdo con los resultados de ambas fases de chequeo.

Como ocurría en el caso del problema Subset Sum, la ejecución de estas fases no se lleva a cabo de manera secuencial y sincronizada. Es decir, etapas correspondientes a fases distintas pueden ser ejecutadas de manera simultánea, pero independiente, en distintas membranas.

A continuación vamos a construir una familia $\mathbf{\Pi}$ de sistemas P reconocedores con membranas activas que *resuelve* (según la Definición 2.13) el problema Knapsack en tiempo *lineal*.

En primer lugar, vamos a considerar una función computable a la que se le asignará el papel de *función tamaño*. De manera similar al caso del Subset Sum, para el problema Knapsack consideremos la función s definida sobre el conjunto de instancias del problema como sigue: $s(u) = \langle n, k, c\rangle = \langle\langle n, k\rangle, c\rangle$ (recuérdese la biyección $\langle x, y\rangle = (x + y) \cdot (x + y + 1)/2 + x$, presentada en el capítulo anterior), para una instancia u=$(n, (w_1, \ldots, w_n), (v_1, \ldots, v_n), k, c)$.

De este modo, el tamaño de la instancia codifica los tres parámetros clave de la instancia: n, k y c. El resto de la información (es decir, pesos y valores) es suministrada al sistema a través del multiconjunto de entrada.

Nótese que la función $\langle \cdot, \cdot, \cdot \rangle$ es polinómica, primitiva recursiva y biyectiva de $\mathbb{N}^3$ en $\mathbb{N}$ (su inversa es también polinómica), y la función s es computable en tiempo polinomial.

La familia presentada es la siguiente:

$$\mathbf{\Pi} = \{(\Pi(\langle n,k,c\rangle), \Sigma(n,k,c), i(n,k,c)) : (n,k,c) \in \mathbb{N}^3\}$$

Para cada elemento de la familia, el alfabeto de entrada es $\Sigma(n,k,c) = \{x_1,\ldots,x_n,y_1,\ldots,y_n\}$, la membrana de entrada es $i(n,k,c) = e$, y el sistema $\Pi(\langle n,k,c\rangle)$ viene dado por la tupla $(\Gamma(n,k,c), \{e,s\}, \mu, \mathcal{M}_s, \mathcal{M}_e, R)$, que describimos a continuación:

- Alfabeto:

$$\Gamma(n,k,c) = \Sigma(n,k,c) \cup \{a_0, a, \bar{a}_0, \bar{a}, b_0, b, \bar{b}_0, \bar{b}, \hat{b}_0, \hat{b}, d_1, e_0,\ldots,e_n, q_0,\ldots,q_{2k+1}, q, \bar{q}, \bar{q}_0,\ldots,\bar{q}_{2c+1}, Yes, No_0, No, z_0,\ldots,z_{2n+2k+2c+6}, \#\}$$

- Estructura de membranas: $\mu = [\ [\ ]_e^0\ ]_s^0$.

- Multiconjuntos iniciales: $\mathcal{M}_s = z_0$; $\mathcal{M}_e = e_0\,\bar{a}^k\,\bar{b}^c$.

- El conjunto de reglas de evolución, R, consta de las siguientes reglas:

  (*a*) $[e_i]_e^0 \to [q]_e^-[e_i]_e^+$, para $i = 0,\ldots,n$.
  $[e_i]_e^+ \to [e_{i+1}]_e^0[e_{i+1}]_e^+$, para $i = 0,\ldots,n-1$.
  Estos esquemas de reglas son idénticos a los usados para el problema Subset Sum en el capítulo anterior. Se trata de obtener una (única) membrana *asociada* a cada subconjunto (según la Definición 4.1).

  (*b*) $[x_0 \to \bar{a}_0]_e^0$; $[x_0 \to \lambda]_e^+$; $[x_i \to x_{i-1}]_e^+$, para $i = 1,\ldots,n$.
  $[y_0 \to \bar{b}_0]_e^0$; $[y_0 \to \lambda]_e^+$; $[y_i \to y_{i-1}]_e^+$, para $i = 1,\ldots,n$.
  De nuevo en este apartado recurrimos a esquemas de reglas casi idénticos a los del apartado (*b*) para el problema Subset Sum. La diferencia reside en que en este caso en la fase de cálculo intervienen dos funciones: la función peso, w, y la función valor, v. La introducción de los datos en el sistema también se hará para el problema Knapsack a través de un multiconjunto de entrada apropiado: para cada $a_j \in A$, se introducen w_j copias del objeto x_j y v_j copias del objeto y_j en la membrana de entrada (es decir, la multiplicidad de los objetos x_j codificará el *peso* del elemento a_j, y la multiplicidad de los objetos y_j representará el *valor* de dicho elemento).

(*c*) $[q \to \bar{q}q_0]_e^-$; $[\bar{a}_0 \to a_0]_e^-$; $[\bar{a} \to a]_e^-$; $[\bar{b}_0 \to \hat{b}_0]_e^-$; $[\bar{b} \to \hat{b}]_e^-$.

Cuando una membrana recibe carga negativa por primera vez, las dos primeras fases (es decir, las de generación y cálculo) terminan, y son aplicadas estas reglas como transición hacia la fase de chequeo correspondiente a la función peso w. Los objetos $\bar{a}$ y $\bar{b}$, cuyas multiplicidades codifican en el multiconjunto inicial las constantes k y c, respectivamente, así como los objetos $\bar{a}_0$ y $\bar{b}_0$ (que codifican el peso y el valor del subconjunto asociado a la membrana), son renombrados para esta tercera fase, donde las multiplicidades de a y de a_0 serán comparadas. También se genera un objeto q_0, que actuará como contador, y aparecen asimismo objetos $\bar{q}$, $\hat{b}_0$ y $\hat{b}$, aunque éstos permanecerán inactivos hasta que la membrana obtenga carga positiva. Esto ocurrirá sólo si el chequeo se resuelve afirmativamente (ver reglas en (f)), en cuyo caso los objetos $\bar{q}$, $\hat{b}_0$ y $\hat{b}$ se activan para el chequeo correspondiente a v.

(*d*) $[a_0]_e^- \to [\ ]_e^0\#$; $[a]_e^0 \to [\ ]_e^-\#$.

Estas reglas implementan la comparación antes mencionada (esto es, comprueban si se verifica $w(B) \leq k$). De nuevo se reutiliza una técnica ya empleada para el problema Subset Sum: estas reglas funcionan como un ciclo que elimina los objetos a_0 y a, uno a uno alternativamente, cambiando la carga de la membrana en cada paso (ver apartado (d) en la Sección 4.2).

(*e*) $[q_{2j} \to q_{2j+1}]_e^-$, para $j = 0, \ldots, k$.

$[q_{2j+1} \to q_{2j+2}]_e^0$, para $j = 0, \ldots, k-1$.

Las reglas que rigen la evolución del contador q_i, encargado del control del bucle antes descrito, también son idénticas a las utilizadas en el capítulo anterior para el problema Subset Sum, aunque hay que modificar las reglas del final de la fase de chequeo, como se explica en el siguiente apartado.

(*f*) $[q_{2j+1}]_e^- \to [\ ]_e^+\#$, para $j = 0, \ldots, k$.

Si existe un subconjunto $B \subseteq A$ tal que verifique la condición $w(B) \leq k$, entonces su membrana asociada deberá pasar a la siguiente fase, y las asociadas a conjuntos que no verifiquen esa condición se bloquearán. En efecto, dado un subconjunto que verifique la desigualdad citada, habrá menos objetos a_0 que a dentro de la membrana relevante (ver Definición 4.2) asociada al mismo. Entonces el ciclo descrito en (e) parará cuando se agoten los objetos a_0: en ese momento la regla $[q_{2w(B)} \to q_{2w(B)+1}]_e^-$ se aplicará pero no será posible aplicar al mismo tiempo la regla $[a_0]_e^- \to [\ ]_e^0\#$. Por tanto, habrá un objeto $q_{2w(B)+1}$ presente en la membrana y ésta tendrá carga negativa, con lo que una regla de

este apartado (para $j = w(B)$) se aplicará, dando paso a la fase de chequeo para v.

(*g*) $[\bar{q} \to \bar{q}_0]_e^+$; $[\hat{b}_0 \to b_0]_e^+$; $[\hat{b} \to b]_e^+$; $[a \to \lambda]_e^+$.
Estas reglas desempeñan la misma función que las del apartado (*c*): realizan un renombramiento antes de pasar a la siguiente fase (chequeo para v).

(*h*) $[b_0]_e^+ \to [\ ]_e^0\#$; $[b]_e^0 \to [\ ]_e^+\#$.
El bucle de comparación para la función v está diseñado exactamente como el de la función w, pero las cargas eléctricas que están en juego ahora son positiva y neutra.

(*i*) $[\bar{q}_{2j} \to \bar{q}_{2j+1}]_e^+$ para $j = 0, \ldots, c$.
$[\bar{q}_{2j+1} \to \bar{q}_{2j+2}]_e^0$ para $j = 0, \ldots, c-1$.
El contador q_i, descrito por las reglas del apartado (*e*), es ahora reemplazado por uno ligeramente diferente, $\bar{q}_i$. Ambos evolucionan de manera similar, pero el final de la cuarta fase (chequeo para la función v) es distinto al de la tercera (chequeo para la función w), porque ahora el resultado exitoso se da cuando la multiplicidad de b_0 es *mayor o igual* que la de b.

(*j*) $[\bar{q}_{2c+1}]_e^+ \to [\ ]_e^0 Yes$; $[\bar{q}_{2c+1}]_e^0 \to [\ ]_e^0 Yes$.
Como apuntábamos antes, si un subconjunto $B \subseteq A$ verifica la condición $v(B) \geq c$, entonces en la membrana correspondiente a ese subconjunto habrá más (o igual número de) objetos b_0 que b. Esto provoca que las reglas del ciclo de (*h*) puedan aplicarse c veces cada una, tras lo cual el índice de $\bar{q}_i$ será $i = 2c$. Entonces se aplicará la regla $[\bar{q}_{2c} \to \bar{q}_{2c+1}]_e^+$ (y tal vez $[b_0]_e^+ \to [\ ]_e^0\#$ se aplique simultáneamente), tras lo cual alguna de las dos reglas de este apartado finalizará la fase.

(*k*) $[z_i \to z_{i+1}]_s^0$, para $i = 0, \ldots, 2n + 2k + 2c + 5$.
$[z_{2n+2k+2c+6} \to d_1 No_0]_s^0$.
El contador z_i, situado en la piel, cumple la misma función que en el diseño para el problema Subset Sum: evita que la respuesta se envíe al exterior mientras exista alguna membrana interna que esté realizando algún proceso de la fase de chequeo. El rango del índice del contador se ha modificado convenientemente, teniendo en cuenta que ahora existen dos chequeos que realizar (el de la función w y el de la función v).

(*l*) $[d_1]_s^0 \to [\ ]_s^+ d_1$; $[No_0 \to No]_s^+$; $[Yes]_s^+ \to [\ ]_s^0 Yes$; $[No]_s^+ \to [\ ]_s^0 No$.

Por último, se activa el proceso de respuesta. La membrana piel necesita obtener carga positiva antes de permitir que se envíe la respuesta al entorno. El objeto d_1 se ocupa de ello y después, si la respuesta es afirmativa, se enviará un objeto Yes al entorno restaurando la carga eléctrica neutra para la piel. Una vez más se han reutilizado reglas del diseño del capítulo anterior (ver apartado (h) en la Sección 4.2).

Es fácil probar que la familia de sistemas P que se acaba de definir está formada exclusivamente por sistemas deterministas, sin más que estudiar las cabezas de las reglas (junto con las etiquetas asociadas y las condiciones de carga eléctrica) y comprobar que no existen dos reglas que puedan afectar a un mismo objeto en cualquier membrana de una configuración dada. Más aún, probaremos en la Sección 5.4 que la familia $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$ presentada en esta sección resuelve el problema Knapsack en tiempo *lineal*, de acuerdo con la Definición 2.13.

5.3. Seguimiento informal de la computación

Recordemos que para resolver un problema mediante una familia de sistemas celulares vamos a asociar a cada instancia del problema un multiconjunto (de entrada) y un número ("tamaño" de la instancia) de manera que a la hora de resolver una instancia concreta, se considera el sistema P de la familia que corresponda al tamaño de la instancia, se le introduce el multiconjunto apropiado como entrada, y entonces comienza la computación.

En nuestro caso, dada una instancia $u = (n, (w_1, \ldots, w_n), (v_1, \ldots v_n), k, c)$ del problema Knapsack, consideraremos la función tamaño introducida en la sección anterior, $s(u) = \langle n, k, c\rangle$, y consideraremos asimismo una función *cod*, que asigna a la instancia u el multiconjunto $cod(u) = x_1^{w_1} \ldots x_n^{w_n} y_1^{v_1} \ldots y_n^{v_n}$, de tal manera que el procesamiento de la instancia u (es decir, decidir si se acepta o se rechaza dicha instancia) se realiza a través del análisis de las computaciones del sistema $\Pi(s(u))$ con entrada $cod(u)$. A continuación describiremos informalmente el desarrollo de las mismas.

En el primer paso de computación se aplica la regla $[e_i]_e^0 \to [q]_e^-[e_i]_e^+$, para $i = 0$. A partir de ahí las *fases de generación* y *de cálculo* se desarrollan en paralelo, siguiendo las indicaciones de las reglas de (a) y (b). Estas dos fases no finalizan en una membrana mientras haya un objeto e_j (con $0 \leq j \leq n$) en su interior. Lo que se pretende es obtener una única membrana de trabajo para cada subconjunto de A, pero, como se verá a continuación, éstas no son generadas a la par (se puede comprobar que

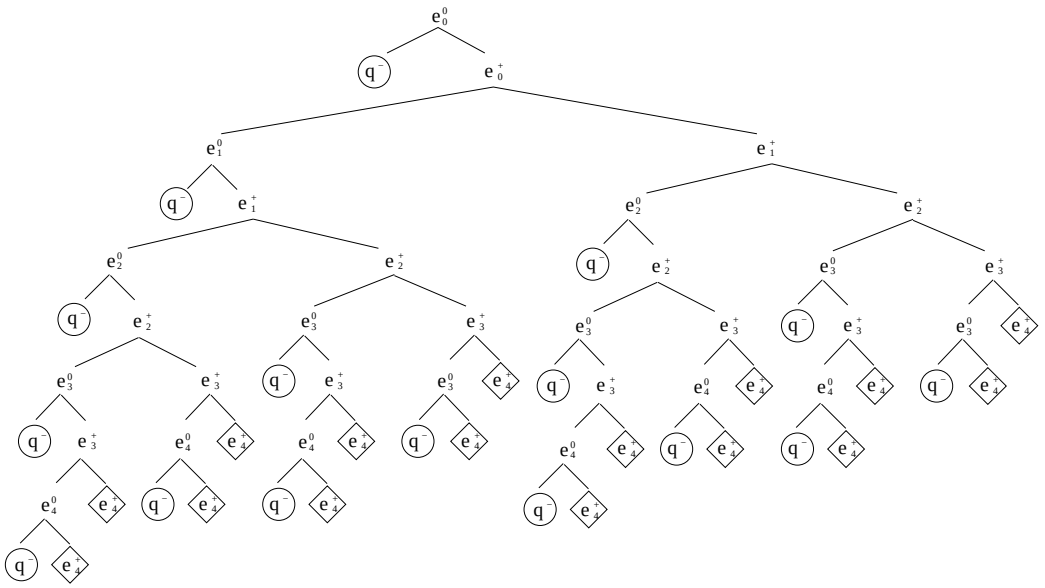

Figura 5.1: Fase de generación para $n = 4$

la membrana correspondiente al subconjunto $\{a_{i_1}, \ldots, a_{i_r}\}$ se obtendrá en el paso $(i_r + r + 2)$-ésimo).

En este capítulo utilizaremos la expresión *subconjunto asociado* en el mismo sentido que en el capítulo anterior; es decir, se considera que cada membrana tiene asociado un subconjunto de A, que irá siendo ampliado durante la fase de generación en función de la carga de la membrana en cada momento y del índice del objeto e_i que se encuentre en la membrana.

Como ya se ha dicho, las dos primeras fases se llevan a cabo en paralelo. De hecho, tan sólo hay un desfase de un paso de computación entre el instante en que se decide añadir un elemento de A al subconjunto asociado y el instante en que se actualizan el nuevo peso y el nuevo valor de dicho subconjunto. Por ejemplo, para a_1: tras dos pasos de computación se puede observar que en la configuración del sistema aparecen tres membranas internas, una de las cuales tiene carga neutra y contiene al objeto e_1 (ver Figura 5.1). Entonces, el elemento a_1 es añadido al subconjunto asociado. Se puede probar que en ese momento hay w_1 copias de x_0 y v_1 copias de y_0 en la membrana. Por tanto, en el siguiente paso, al mismo tiempo que la membrana se divide, las reglas de (b) generan w_1 objetos $\bar{a}_0$ y v_1 objetos $\bar{b}_0$. Dichas reglas también modificarán los índices de todos los objetos x_i e y_i (con $i > 0$) cuando haya carga positiva, de manera que w_2 copias de x_0 y v_2 copias de y_0 estarán listas en la membranas que se obtengan dos pasos después.

Cuando se aplica una regla del tipo $[e_i]_e^0 \rightarrow [q]_e^-[e_i]_e^+$, las dos nuevas membranas que se obtienen fruto de esa división se comportarán de manera totalmente distinta. Por un lado, la membrana con carga negativa (dichas membranas están marcadas con un círculo en la Figura 5.1) da por terminadas las dos primeras fases y, en el siguiente paso, se aplicarán en ella las reglas de (c), renombrando los objetos para preparar la tercera fase.

Éste es un momento significativo, por lo que convendremos en llamar membranas *relevantes* a aquellas que tengan carga negativa y contengan un objeto q_0, así como a todas aquellas obtenidas a partir de éstas mediante transiciones del sistema (como ya se hizo en el capítulo anterior). Una membrana relevante ya no se dividirá más durante la computación, y su subconjunto asociado ya no será modificado. Por otro lado, la membrana cargada positivamente proseguirá con las fases de generación y cálculo. Mediante sucesivas divisiones dará lugar a membranas asociadas con subconjuntos obtenidos añadiendo nuevos elementos al actual, de índice $i+1$ o superior. Obsérvese que para el caso $i = n$, la membrana no puede continuar la fase de generación, al no haber reglas para el objeto e_n con condición de carga positiva (véanse las membranas rodeadas por un rombo en la Figura 5.1). Tiene sentido que estas membranas queden bloqueadas, ya que no es posible añadir elementos de índice mayor que n.

Consecuentemente, dado que los índices de los objetos e_i nunca decrecen, se observa que las membranas relevantes se generan siguiendo una especie de "orden lexicográfico", en el siguiente sentido: si el j-ésimo elemento de A ya ha sido añadido al subconjunto asociado de cierta membrana, entonces ningún elemento con índice menor que j será añadido, posteriormente, al subconjunto asociado a dicha membrana ni a los subconjuntos de ninguna de las membranas que resulten de divisiones sucesivas.

El objetivo de las reglas que aparecen en (d) es comparar las multiplicidades de los objetos a_0 y a (esto es, llevar a cabo la *fase de chequeo para la función* w). Nótese que para evitar que las reglas de comparación del apartado (h) interfieran en esta fase, los objetos $\bar{b}$ y $\bar{b}_0$ son renombrados como $\hat{b}$ y $\hat{b}_0$, respectivamente, y permanecen inactivos durante toda esta fase. Recordemos que el contador q_i es quien controla el resultado de la comparación. Expliquemos brevemente cómo funciona el ciclo de chequeo.

Sea B un subconjunto con un cierto peso w_B y un cierto valor v_B. Entonces la evolución de la membrana relevante asociada a él, a lo largo de la tercera fase, está descrita en la Tabla 2.

Multiconjunto	Carga	Paridad de q_i
$q_0 a_0^{w_B} a^k \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	PAR
$q_1 a_0^{w_B-1} a^k \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	IMPAR
$q_2 a_0^{w_B-1} a^{k-1} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	PAR
$\vdots$	$\vdots$	$\vdots$
$q_{2j} \ a_0^{w_B-j} a^{k-j} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	PAR
$q_{2j+1} \ a_0^{w_B-(j+1)} a^{k-j} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	IMPAR
$\vdots$	$\vdots$	$\vdots$

Tabla 2

Obsérvese que el índice de q_i coincide con la cantidad total de copias de objetos a y a_0 que ya han sido eliminados durante la comparación. Si $B = \{a_{i_1}, \ldots, a_{i_r}\}$ con $i_r \neq n$, entonces en el multiconjunto habrá además algunos objetos x_j e y_j, para $1 \leq j \leq n - i_r$, pero son irrelevantes para esta fase y, por tanto, se omitirán.

Si el número de objetos a_0 es menor o igual que el número de objetos a (es decir, si $w_B \leq k$) entonces el resultado de esta fase es exitoso y podemos pasar a la siguiente comparación. Para ello, las reglas que aparecen en el apartado (g) se ocupan de preparar, con un renombramiento apropiado, los objetos cuyas multiplicidades serán comparadas. Este proceso está descrito en la Tabla 3.

Multiconjunto	Carga	Paridad de q_i	reglas
$q_{2w_B-1} \ a^{k-w_B+1} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	IMPAR	d, e
$q_{2w_B} \ a^{k-w_B} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	PAR	e
$q_{2w_B+1} \ a^{k-w_B} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	IMPAR	f
$a^{k-w_B} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$+$		g
$\bar{q}_0 b_0^{v_B} b^c$	$+$		h, i

Tabla 3

Si el número de objetos a_0 es mayor que k, entonces cada vez que se aplique la regla $[q_{2j} \to q_{2j+1}]_e^-$ (esto es, para $j = 0, \ldots, k$), la regla $[a_0]_e^- \to [\ ]_e^0\#$ también se aplicará. Por tanto, no se puede llegar a una situación en la que el índice del contador q_i sea un número impar y la carga sea negativa. Esto significa que las reglas del apartado (f) nunca se podrán aplicar y, más aún, la membrana se *bloqueará* (no evolucionará más durante la computación). Para más detalles, ver la Tabla 4.

Multiconjunto	Carga	Paridad de q_i
$\vdots$	$\vdots$	$\vdots$
$q_{2k-1}\ a_0^{w_B-k} a\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	IMPAR
$q_{2k}\ a_0^{w_B-k}\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	PAR
$q_{2k+1}\ a_0^{w_B-(k+1)}\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	IMPAR

Tabla 4

Supongamos que la tercera fase se ha completado con éxito en una membrana. Esto quiere decir que dicha membrana codifica un subconjunto $B \subseteq A$ tal que $w(B) \leq k$. Pues bien, tras la aplicación de la regla de (f) correspondiente a $j = w(B)$ (esto es, la regla $[q_{2w(B)+1}]_e^- \rightarrow [\ ]_e^+\#$), dicha membrana pasa a tener carga positiva. Como consecuencia de ello, se aplican las reglas que aparecen en (g). Es decir, se realiza un paso de transición, renombrando los objetos $\hat{b}_0$ y $\hat{b}$ como b_0 y b, respectivamente, a fin de permitir la comparación de sus multiplicidades (es decir, se prepara la *fase de chequeo para la función* v). También se inicializa el contador $\bar{q}_i$ (a $\bar{q}_0$) que comienza a evolucionar controlando el resultado de la comparación.

La cuarta fase se desarrolla en cada membrana de manera muy similar a la tercera. Las reglas que aparecen en (h) e (i) se corresponden con las reglas que aparecen en (d) y (e), respectivamente, pero el final de la fase es distinto (véase Tabla 5). La diferencia estriba en las reglas del apartado (j). En esta fase, el chequeo tiene éxito si el número de objetos b_0 es mayor o igual que el número de objetos b (es decir, si $v(B) \geq c$) y así, para poder pasar a la siguiente fase, las dos reglas que aparecen en (h) deben aplicarse c veces cada una (y, consecuentemente, las reglas que aparecen en (i) se aplican el mismo número de veces). Esta condición está garantizada por las cabezas de las reglas de (j), que exigen que el índice de $\bar{q}_i$ debe ser $i = 2c + 1$. Para más detalles, ver la Tabla 5.

Multiconjunto	Carga	Paridad de q_i	reglas
$\bar{q}_0 b_0^{v_B} b^c$	$+$	PAR	h, i
$\bar{q}_1 b_0^{v_B-1} b^c$	0	IMPAR	h, i
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\bar{q}_{2c-1} b_0^{v_B-c} b$	0	IMPAR	h, i
$\bar{q}_{2c} b_0^{v_B-c}$	$+$	PAR	h, i
$\bar{q}_{2c+1} b_0^{v_B-(c+1)}$ (si $v_B > c$)	0	IMPAR	j
o $\bar{q}_{2c+1}$ (si $v_B = c$)	$+$	IMPAR	j
$b_0^{v_B-(c+1)}$ o bien $\emptyset$	$+$		

Tabla 5

Por último, las reglas que aparecen en (k) y en (l) están asociadas a la membrana piel y se encargan de la fase de respuesta. El contador z_i, descrito en el apartado (k), espera durante $2n + 2k + 2c + 7$ pasos ($2n + 2$ pasos para la primera y la segunda fase, luego un paso de transición, después $2k + 1$ pasos para la tercera fase, otro paso de transición y, finalmente, $2c + 2$ para la cuarta). Después de que todos esos pasos se hayan dado, se tiene que todas las membranas internas ya han terminado sus fases de chequeo (o se han quedado bloqueadas) y, por tanto, es el momento de activar el proceso de respuesta.

En ese instante, la membrana piel tendrá carga neutra y aparecerán en ella los objetos d_1 y No_0. Además, en dicha membrana habrá también objetos Yes si y sólo si las dos fases de chequeo han sido exitosas en, al menos, una membrana interna.

Pasemos a la fase de respuesta. En primer lugar, un objeto d_1 es enviado fuera, cambiando la carga de la piel a positiva. Entonces el objeto No_0 evoluciona a un objeto No dentro de la piel (segunda regla que aparece en (l)) y, simultáneamente, si existe algún objeto Yes presente en la membrana piel, será enviado fuera del sistema (al entorno), restaurando la carga neutra de la piel y haciendo que el sistema pare (en particular, se evita cualquier evolución del objeto No).

Si se diera el caso de que ninguna de las membranas tiene éxito en sus dos chequeos, entonces no habrá ningún objeto Yes presente en la piel en el instante en que comience la fase de respuesta. En ese caso, después de que se genere el objeto No, la piel todavía tendrá carga positiva y, por tanto, podrá ser enviado al entorno como respuesta. En ese momento el sistema para.

5.4. Verificación Formal

En esta sección se trata de probar que la familia $\mathbf{\Pi}$ de sistemas P con membranas activas diseñados en la Sección 5.2 proporciona una solución polinomial para el problema *Knapsack*, en el sentido de la Definición 2.13; esto es, $Knapsack \in \mathbf{PMC}_{\mathcal{AM}}$.

En primer lugar, hay que justificar que la familia definida es $\mathcal{AM}$-consistente; es decir, que todos los sistemas de la familia son sistemas celulares reconocedores de lenguajes que usan membranas activas. De la construcción realizada (por el tipo de reglas y por el alfabeto de trabajo) se sigue directamente que se trata de una familia de sistemas aceptadores de lenguajes con membranas activas. Al igual que se hizo en el capítulo anterior, se demostrará que todas las computaciones paran y lo hacen de manera confluente (es decir, que o bien todas las computaciones asociadas a un multiconjunto de entrada son de aceptación o bien todas son de rechazo).

5.4.1. Uniformidad polinomial de la familia

A continuación, veamos que la familia es polinomialmente uniforme por máquinas de Turing. Se puede observar que la definición de la familia está hecha de manera recursiva a partir de las instancias, en particular a partir de los valores de las constantes n, k y c. Además, los recursos necesarios para construir un elemento de la familia son de orden lineal con respecto a dichas constantes:

- tamaño del alfabeto: $5n + 4k + 4c + 28 \in \Theta(n + k + c)$,
- número de membranas: $2 \in \Theta(1)$,
- $|\mathcal{M}_e| + |\mathcal{M}_s| = k + c + 2 \in \Theta(k + c)$,
- suma de las longitudes de todas las reglas:

$$40n + 27k + 20c + 193 \in \Theta(n + k + c).$$

Antes de proseguir, recordemos que las funciones *cod* y s se han definido en la sección anterior para una instancia $u = (n, (w_1, \ldots, w_n), (v_1, \ldots v_n), k, c)$ como sigue: $cod(u) = x_1^{w_1} \ldots x_n^{w_n} y_1^{v_1} \ldots y_n^{v_n}$, y $s(u) = \langle n, k, c \rangle$, respectivamente.

Ambas funciones son totales y polinomialmente computables. Más aún, el par (cod, s) constituye una codificación polinomial del conjunto de instancias del problema *Knapsack* en la familia $\mathbf{\Pi}$ de sistemas P, ya que se verifica que, para cualquier instancia u, el multiconjunto $cod(u)$ es una entrada válida para el sistema $\Pi(s(u))$.

Obsérvese que cada instancia $u = (n, (w_1, \ldots, w_n), (v_1, \ldots v_n), k, c)$ es introducida en la configuración inicial de su sistema celular asociado mediante un multiconjunto de entrada (es decir, en una representación 1-aria) y, por tanto, se tiene que $|u| \in O(w_1 + \cdots + w_n + v_1 + \cdots + v_n + k + c)$.

Para establecer la verificación formal de la familia de sistemas P respecto al problema Knapsack necesitamos probar que todos los sistemas de la familia están polinomialmente acotados y, además, que son adecuados y completos con respecto a $(Knapsack, cod, s)$, de acuerdo con la Definición 2.13.

5.4.2. Acotación polinomial de la familia

Para cerciorarnos de que el sistema $\Pi(s(u))$ con entrada $cod(u)$ está polinomialmente (de hecho, linealmente) acotado, basta encontrar el instante en que la computación para o, al menos, una cota superior del mismo. Como veremos a continuación, el número de pasos de las computaciones de los sistemas de la familia puede acotarse siempre por una función *lineal*. Sin embargo, conviene mencionar que la cantidad de recursos pre-computados para cada instancia u es polinomial en el tamaño de la instancia, puesto que $cod(u)$ y $s(u)$ necesitan ser calculados y $\Pi(s(u))$ necesita ser construido.

En primer lugar vamos a analizar la *fase de generación*. Las reglas que aparecen en (a) están diseñadas de manera que una condición necesaria y suficiente para que una membrana se divida es que ocurra uno de los dos siguientes casos: o bien tiene carga neutra y contiene un objeto e_j (con $j \in \{0, 1, \ldots, n\}$), o bien su carga es positiva y contiene un objeto e_j (con $j \in \{0, 1, \ldots, n-1\}$).

- En el primer caso el resultado de la división es una membrana con carga negativa, donde e_j es reemplazado por q, y una membrana con carga positiva donde permanece el objeto e_j (si $j < n$, entonces esta última membrana estaría en el segundo caso y, por tanto, se dividiría otra vez).

- En el segundo caso, una de las membranas obtenidas tiene carga neutra y la otra positiva, y en ambas el objeto e_j es reemplazado por e_{j+1}.

Dado que el conjunto de índices es finito, las divisiones no pueden seguir produciéndose indefinidamente y, por tanto, la fase de generación es un proceso finito. De hecho, teniendo presente que tras dos divisiones consecutivas el índice del objeto e_j aumenta en una unidad (si no es reemplazado por q), se puede deducir que tras $2n + 1$ pasos no tendrán lugar más divisiones.

Para la *fase de cálculo* se realiza un razonamiento similar. Basta considerar los objetos x_i e y_i y observar cómo sus índices decrecen en cada paso (si $i > 0$) o cómo son eliminados o renombrados (si $i = 0$). Así pues, esta fase también da lugar a un proceso finito.

La tercera fase (*fase de chequeo para la función w*) consiste, básicamente, en un bucle formado por las reglas $[a_0]_e^- \to [\ ]_e^0\#$ y $[a]_e^0 \to [\ ]_e^-\#$. De nuevo podemos afirmar que se trata de un proceso finito, ya que en cada vuelta del bucle se elimina una pareja de objetos. En paralelo a la aplicación de estas dos reglas, el contador q_i va evolucionando, y hace que esta fase termine en, a lo sumo, $2k + 1$ pasos. Si el chequeo termina con éxito (es decir, si $w_B \leq k$, siendo B el subconjunto asociado), la comparación durará $2w_B + 1$ pasos, como se vió en la sección anterior; pero si el chequeo tiene resultado negativo (es decir, si $w_B > k$) entonces la membrana quedará bloqueada tras $2k + 1$ pasos y no pasará a la siguiente fase.

Supongamos que en una membrana se completa con éxito la tercera fase. En el siguiente paso las reglas que aparecen en (g) se aplicarán en dicha membrana, preparando los objetos b y b_0 para que se comparen sus multiplicidades e inicializando el contador $\bar{q}_0$ que controla el resultado. Como ocurría para la función w, la *fase de chequeo para la función v* consiste, básicamente, en la reiteración de un bucle, pero en este caso el número de pasos será de $2c + 2$ para el caso afirmativo (es decir, si $v(B) \geq c$) y en el caso negativo el proceso se bloqueará tras $2v_B + 2$ pasos (si

$v(B) < c$).

En consecuencia, no se aplicará ninguna regla en las membranas internas tras la ejecución de los primeros $2n + 2k + 2c + 7$ pasos.

Sólo queda repasar la *fase de respuesta*. El mecanismo es simple: hay un contador z_j en la piel que evoluciona desde z_0 hasta $z_{2n+2k+2c+6}$, lo que requiere $2n+2k+2c+6$ pasos en total. Entonces, $z_{2n+2k+2c+6}$ evoluciona produciendo dos objetos, d_1 y No_0, y, a continuación, se aplica la regla $[d_1]_s^0 \rightarrow [\ ]_s^+ d_1$, cambiando la polarización de la piel a positiva. En ese momento, si la respuesta a la instancia es afirmativa, entonces el objeto No_0 se transforma en un objeto No dentro de la piel y, simultáneamente, un objeto Yes es enviado al entorno cambiando la carga de la piel a neutra, con lo que el sistema para en el paso $(2n + 2k + 2c + 9)$-ésimo. Si, por el contrario, la respuesta es negativa, entonces el objeto No_0 evoluciona a un objeto No dentro de la piel pero ningún objeto Yes es enviado fuera simultáneamente, por lo que la piel aún tiene carga positiva en el siguiente paso y, consecuentemente, se aplica la regla $[No]_s^+ \rightarrow [\ ]_s^0 No$, provocando la parada del sistema en el paso $(2n+2k+2c+10)$-ésimo. Recuérdese que $|u| \in O(w_1 + \cdots + w_n + v_1 + \cdots + v_n + k + c)$.

5.4.3. Adecuación y completitud de la familia

A continuación vamos a establecer la adecuación y completitud de la familia $\mathbf{\Pi}$, respecto a $(Knapsack, cod, s)$. Es decir, vamos a probar que dada una instancia, u, del problema Knapsack, el sistema P de la familia asociado a dicha instancia, $\Pi(s(u))$, con entrada $cod(u)$ responde Yes si y sólo si la instancia u considerada tiene una respuesta afirmativa, y que en caso contrario expulsa un objeto No.

Proposición 5.1 *La familia $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$, definida en la Sección 5.2, es adecuada y completa respecto a $(Knapsack, cod, s)$.*

Demostración. La prueba se hará estableciendo el correcto funcionamiento de cada fase por separado. En primer lugar, centrémonos en la *fase de generación*. Es necesario probar que a lo largo de la computación se generan membranas (relevantes) de trabajo para todos los subconjuntos de A.

Lema 5.1 (fase de generación) *Dado un subconjunto $B \subseteq A$, en algún momento de la computación aparece una membrana relevante asociada a dicho subconjunto.*

Demostración. Consideremos un subconjunto $B = \{a_{i_1}, \ldots, a_{i_r}\} \subseteq A$ (con $i_1 < i_2 < \cdots < i_r$, y $r \leq n$); entonces debe existir una única membrana relevante que lo codifique. De hecho, ésta es el resultado de la siguiente sucesión de transiciones:

$$[e_0]_e^0 \Rightarrow [e_0]_e^+ \Rightarrow [e_1]_e^+ \Rightarrow \cdots \Rightarrow [e_{i_1-1}]_e^+ \Rightarrow [e_{i_1}]_e^0 \Rightarrow [e_{i_1}]_e^+ \Rightarrow \cdots \Rightarrow$$
$$\Rightarrow [e_{i_2-1}]_e^+ \Rightarrow [e_{i_2}]_e^0 \Rightarrow \cdots \Rightarrow [e_{i_r}]_e^0 \Rightarrow [q]_e^- \Rightarrow [q_0]_e^-.$$

Debido a su definición recursiva, el subconjunto asociado refleja de algún modo la *historia* de la membrana (el subconjunto asociado está formado por los elementos $a_i \in A$ tales que en algún momento de la sucesión la membrana tenía carga neutra y contenía un objeto e_i). A partir de esta consideración, es fácil probar que la sucesión arriba descrita conduce efectivamente a una membrana relevante determinada unívocamente y que tiene a B como subconjunto asociado.

□

A continuación, para asegurarnos de que la segunda fase también funciona correctamente, veremos que las multiplicidades de los objetos a_0 y b_0 en la membrana antes mencionada son $w(B)$ y $v(B)$, respectivamente. Obsérvese que no representa problema alguno añadir los pesos de los elementos de A immediatamente después de que sean seleccionados para el subconjunto asociado, ya que si un elemento a_j pertenece al subconjunto codificado por cierta membrana, entonces también pertenecerá a los subconjuntos codificados por sus descendientes y, por tanto, su peso puede ser añadido ya a la membrana.

Lema 5.2 *En una membrana interna con carga positiva en la que aparezca un objeto e_i, con $0 \leq i \leq n-1$, las multiplicidades de los objetos x_j e y_j se corresponden exactamente con los pesos $w(a_{i+j})$ y los valores $v(a_{i+j})$ de los elementos $a_{i+j} \in A$, respectivamente, para cada j tal que $1 \leq j \leq n-i$.*

Demostración. Por inducción sobre i.

Para el caso base, $i = 0$, observemos que el multiconjunto asociado a la membrana e en la configuración inicial es $e_0 \bar{a}^k \bar{b}^c x_1^{w_1} \cdots x_n^{w_n} y_1^{v_1} \cdots y_n^{v_n}$. Entonces, en el primer paso se aplicará la regla $[e_0]_e^0 \rightarrow [q]_e^- [e_0]_e^+$ (de hecho, es la única regla aplicable en la membrana en esa situación), y el multiconjunto permanecerá inalterado en la membrana cargada positivamente. Así pues, para todo j tal que $1 \leq j \leq n-i$, las multiplicidades de los objetos x_j e y_j se corresponden exactamente con los pesos y los valores de los elementos $a_{i+j} \in A$, como queríamos demostrar.

Para establecer el paso inductivo, supongamos que el resultado es cierto para $i < n-1$. Consideremos una membrana interna cargada positivamente y en la que aparece un objeto e_{i+1}. Distinguiremos dos casos:

1. En primer lugar, supongamos que la membrana se ha obtenido fruto de la aplicación de la regla $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0 [e_{i+1}]_e^+$. Entonces, de la hipótesis inductiva se deduce que las multiplicidades de los objetos x_j e y_j en la membrana original

(en la que aparece e_i) se corresponden exactamente con los pesos y los valores de los elementos $a_{i+j} \in A$, para todo j tal que $1 \leq j \leq n-i$. Obsérvese que también se aplican las reglas que aparecen en (b) (las que tengan condición de carga positiva) y, por tanto, las multiplicidades de los objetos x_j e y_j en la nueva membrana son iguales a las de los objetos x_{j+1} e y_{j+1} en la membrana original, para $0 \leq j \leq n-i-1$. Por consiguiente, concluimos que las multiplicidades de los objetos x_j e y_j se corresponden exactamente con los pesos y los valores de los elementos $a_{i+j+1} \in A$, para $1 \leq j \leq n-(i+1)$. Esto completa la prueba, en el caso considerado.

2. En segundo lugar, supongamos que la membrana que estamos considerando se ha obtenido a través de la regla $[e_{i+1}]_e^0 \rightarrow [q]_e^- [e_{i+1}]_e^+$. Se puede seguir un razonamiento similar al anterior para la membrana original (en este caso la membrana original es $[e_{i+1}]_e^0$) ya que ésta ha sido obtenida a su vez como consecuencia de la división $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0 [e_{i+1}]_e^+$. Llegamos entonces a la conclusión de que las multiplicidades de los objetos x_j e y_j en la membrana $[e_{i+1}]_e^0$ se corresponden exactamente con los pesos y los valores de los elementos $a_{i+j+1} \in A$, para todo j tal que $0 \leq j \leq n-i-1$. Además, las multiplicidades de dichos objetos no cambian en el siguiente paso (cuando se genera la membrana $[e_{i+1}]_e^+$), porque la única regla que es aplicada simultáneamente a la división es $[x_0 \rightarrow \bar{a}_0]_e^0$, y ésta no afecta a los objetos x_j e y_j con $j \geq 1$.

□

Lema 5.3 (fase de cálculo) *En una membrana relevante el número de copias de los objetos a_0 y $\hat{b}_0$ coinciden, respectivamente, con el peso y el valor del subconjunto asociado.*

Demostración. Consideremos una membrana relevante arbitraria cuyo subconjunto asociado es $B = \{a_{i_1}, \ldots, a_{i_r}\}$, con $i_1 < i_2 < \cdots < i_r$, y $r \leq n$. Entonces no existe, a lo largo de la computación, ninguna otra membrana relevante asociada al mismo subconjunto.

En la sucesión de membranas que conduce desde la membrana inicial hasta la membrana relevante considerada, existen $r + i_r + 1$ membranas intermedias. Entre ellas se encuentran algunas que tienen carga neutra y contienen un objeto e_{i_l} (hay una por cada $a_{i_l} \in B$). Además, para todo j con $0 \leq j \leq i_r - 1$, hay una membrana cargada positivamente que contiene al objeto e_j. Por último, también aparece en la sucesión una membrana cargada negativamente que contiene al objeto q.

Si se aplica el Lema anterior, para $i = i_l - 1$ (con $1 \leq l \leq r$) y para $j = 1$, en todas las membranas cargadas positivamente y que contengan al objeto e_{i_l-1}, resulta que en cada una de ellas las multiplicidades de los objetos x_1 e y_1 son exactamente w_{i_l} y v_{i_l} (es decir, el peso y el valor de $a_{i_l} \in A$, respectivamente). Dado que en cada una de ellas se aplicarán las reglas de (b) en el siguiente paso, se deduce que para $1 \leq l \leq r$, en la membrana con carga neutra en la que aparezca e_{i_l} habrá w_{i_l} objetos x_0 y v_{i_l} objetos y_0. Consecuentemente, en el siguiente paso se generarán w_{i_l} copias de $\bar{a}_0$ y v_{i_l} copias de $\bar{b}_0$ en la membrana.

Esto se tiene para todo i_l con $1 \leq l \leq r$; es decir, para todos los elementos de B, por lo que en la membrana cargada negativamente que contiene a q, las multiplicidades de los objetos $\bar{a}_0$ y $\bar{b}_0$ son la suma de los pesos y la suma de los valores de los elementos de B, respectivamente. Es decir, coinciden con el peso y el valor del subconjunto B. Finalmente, tras la aplicación de las reglas que aparecen en (c) en el último paso de la sucesión, se tiene que el número de copias de los objetos a_0 y $\hat{b}_0$ coincide, respectivamente, con el peso y el valor de B.

□

A continuación se estudian las fases de chequeo.

Lema 5.4 (fase de chequeo) *Una membrana relevante, asociada a un subconjunto $B \subseteq A$, superará con éxito las dos fases de chequeo (mandando a continuación un objeto Yes a la piel) si y sólo si se verifican las condiciones $w_B \leq k$ y $v_B \geq c$.*

Demostración. Consideremos una membrana relevante arbitraria cuyo subconjunto asociado es $B = \{a_{i_1}, \ldots, a_{i_r}\}$, con $i_1 < i_2 < \cdots < i_r$, y $r \leq n$. Sean $w(B) = w_B$ y $v(B) = v_B$ el peso y el valor de dicho subconjunto, respectivamente. Entonces el multiconjunto de objetos contenido en la membrana en el instante en que llega a ser relevante es

$$q_0 \bar{q} a_0^{w_B} a^k \hat{b}_0^{v_B} \hat{b}^c x_1^{w_{i_r+1}} \cdots x_{n-i_r}^{w_n} y_1^{v_{i_r+1}} \cdots y_{n-i_r}^{v_n}.$$

Se trata de comparar el número de objetos a con el número de objetos a_0 presentes en la membrana. El objeto q_0 evolucionará en paralelo a esta comparación y controlará el final de esta fase. Los objetos $\hat{b}_0$, $\hat{b}$ y $\bar{q}$ permanecen inactivos durante esta fase, como ya se dijo en la sección anterior. Si algún objeto x_j o y_j, con $j > 0$, aparece en la membrana, también permanecerá inactivo durante esta fase, ya que la membrana no volverá a tener carga positiva hasta que termine la fase (y sólo si el chequeo termina con éxito).

Existen dos posibilidades: o bien $w_B \leq k$ o bien $w_B > k$. En la sección anterior ya se detalló cómo evoluciona el sistema en cada caso. La membrana pasa a la siguiente fase (es decir, no se queda bloqueada) si y sólo si $w_B \leq k$. En caso de terminar

el chequeo con éxito la membrana pasará a tener carga positiva, con lo cual los objetos $\hat{b}_0$, $\hat{b}$ y $\bar{q}$ se reactivan (mediante las reglas que aparecen en (g)). Además, los posibles objetos de tipo x_j o y_j que se hallen en la membrana también evolucionarán como resultado de la aplicación de las reglas que aparecen en (b), aunque no les prestaremos atención porque no condicionan la carga de la membrana ni el resultado del chequeo para la función v. El bucle formado por las dos reglas que aparecen en (h) funciona exactamente igual que el bucle de la tercera fase: su objetivo es comparar las multiplicidades de los objetos b y b_0, y esto se realiza eliminando dichos objetos uno a uno (enviándolos fuera de la membrana) y cambiando la carga en cada paso. Como ya se estudió en la sección anterior, en esta fase el contador $\bar{q}_i$ se comporta de un modo diferente a como lo hacía el contador q_i, ya que esta fase termina con éxito cuando el número de objetos b_0 es mayor que el número de objetos b.

Al terminar la fase de chequeo, un objeto Yes será enviado a la piel si y sólo si el contador $\bar{q}_i$ no se ha quedado bloqueado antes de llegar a $\bar{q}_{2c+1}$. Esto es, si y sólo si la membrana cambia de carga en cada paso durante $2c$ pasos. Esta condición es equivalente a afirmar que en cada uno de los $2c$ primeros pasos de esta fase se aplica una de las reglas que aparecen en (h) y, a su vez, esto es equivalente a exigir que haya, al menos, c copias del objeto b y también del objeto b_0. Ya hemos visto que la multiplicidad del objeto b al inicio de la fase coincide con la constante c, así que lo que se está exigiendo realmente es que la multiplicidad de b_0 (es decir, el valor del subconjunto) sea mayor o igual que c. Por tanto, la fase de chequeo para v está bien diseñada.

□

Hasta ahora, hemos probado que para cada subconjunto, $B \subseteq A$, una membrana relevante asociada a dicho subconjunto aparecerá en algún momento de la computación y contendrá, exactamente, w_B copias de a_0 y v_B copias de $\hat{b}_0$. También se ha demostrado que una membrana relevante superará con éxito las dos fases de chequeo (mandando a continuación un objeto Yes a la piel) si y sólo si se verifican las condiciones $w_B \leq k$ y $v_B \geq c$. Por último, veamos que la *fase de respuesta* también funciona adecuadamente.

Para ello hay que tener presente que cuando los objetos d_1 y No_0, encargados de desencadenar el proceso de respuesta, aparecen en la piel ya han transcurrido suficientes pasos celulares para que todas las membranas hayan completado sus dos fases de chequeo. Es decir, o bien han superado ambas y han enviado un objeto Yes a la piel o bien se han quedado bloqueadas. Basta entonces comentar que el objeto Yes tiene una cierta "prioridad" sobre el objeto No a la hora de ser enviado al entorno.

De hecho, cuando el objeto d_1 sale del sistema y la piel recibe carga positiva, el objeto No aún no ha sido generado. Por tanto, si la instancia tiene solución afirmativa, entonces habrá objetos Yes (al menos uno) presentes en el multiconjunto contenido en la piel, y en el siguiente paso uno de esos objetos Yes será enviado al entorno, mientras que un objeto No es generado dentro de la piel. En caso de que la instancia considerada tuviera una solución negativa, ninguna de las membranas mandará un objeto Yes a la piel (porque ningún subconjunto verifica las dos condiciones $w_B \leq k$ y $v_B \geq c$) y, por lo tanto, después de que se envíe al entorno el objeto d_1 se aplicarán las reglas $[No_0 \rightarrow No]_s^+$ y $[No]_s^+ \rightarrow [\ ]_s^0 No$, con lo que se recibirá un objeto No en el entorno y la computación del sistema parará.

Esto completa la prueba de que la familia de sistemas diseñada *resuelve* el problema Knapsack en tiempo lineal, de acuerdo con la Definición 2.13.

□

De todo lo expuesto en esta sección, y de acuerdo con la Definición 2.13, se deduce el siguiente resultado:

Teorema 2 *a*) $Knapsack \in \mathbf{PMC}_{\mathcal{AM}}$.

b) $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

Demostración. Para demostrar *b*), basta hacer las siguientes observaciones: el problema de la Mochila es **NP**–completo, $Knapsack \in \mathbf{PMC}_{\mathcal{AM}}$ y la clase $\mathbf{PMC}_{\mathcal{AM}}$ es estable bajo reducción polinomial y cerrada bajo complementario.

□

Capítulo 6

El problema de la Partición

En este capítulo se presenta una solución eficiente para el problema de la Partición mediante una familia de sistemas celulares reconocedores con membranas activas. A diferencia de las soluciones de los problemas Subset Sum y Knapsack, presentadas en los capítulos anteriores, en este diseño sí se han usado reglas de disolución de membranas. La potencia computacional de la división de membranas elementales, junto con la introducción de algunos contadores específicos, nos ha permitido diseñar una familia de sistemas celulares que resuelve el problema de la Partición en tiempo (de computación celular, paralelo) lineal, en función del tamaño del dato de entrada (que, recordemos, es codificado en los sistemas celulares a través de multiconjuntos y, por tanto, está representado de forma 1-aria). Sin embargo, antes de iniciar propiamente el proceso de computación celular se requiere un tiempo polinomial (de computación clásica, secuencial) en concepto de recursos pre-computados.

El capítulo está estructurado como sigue. Tras una breve introducción, se presenta en la Sección 6.2 el diseño de una solución celular para el problema de la Partición, en el marco de los sistemas reconocedores de lenguajes, siguiendo la línea desarrollada en los dos capítulos anteriores. En las Secciones 6.3 y 6.4 se describe informalmente el desarrollo de una computación para un sistema genérico de la familia (distinguiendo el comportamiento durante las distintas fases de la evolución y la transición entre ellas) y se establece la verificación formal de la solución antes detallada. En la última sección se analizan algunas posibles modificaciones para mejorar el diseño.

6.1. Introducción

El problema de la Partición se puede enunciar como sigue:

> *Dado un conjunto A con n elementos, donde cada elemento tiene asociado un peso, determinar si existe una partición de A en dos subconjuntos de manera que ambos tengan el mismo peso.*

Ilustramos gráficamente el problema en la Figura 6.1, se trata de distribuir las pesas equitativamente entre los dos platos de la balanza (el número de pesas en cada plato puede no ser el mismo).

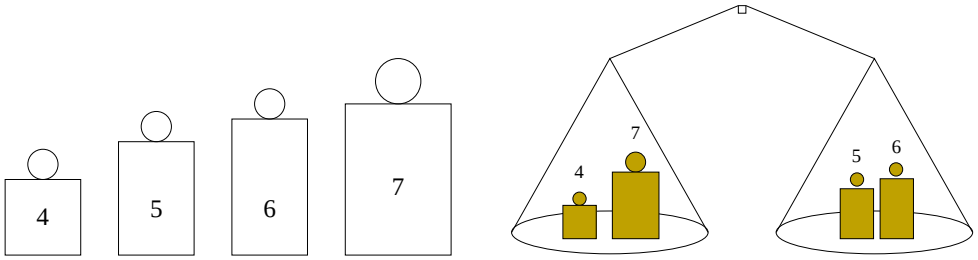

Figura 6.1: Esquema informal del problema de la Partición

Como ya se avanzó en el capítulo anterior, los problemas Subset Sum, Knapsack y Partición están íntimamente ligados. Todos ellos son problemas numéricos **NP**–completos, y las reducciones polinomiales requeridas para pasar de uno a otro son bastante sencillas. Por ejemplo:

- *Reducción del problema de la Partición al problema Subset Sum*: Supongamos que disponemos de un algoritmo que resuelve el problema Subset Sum. Entonces, para resolver el problema de la Partición basta utilizar dicho algoritmo suministrándole como entrada la tupla $(n, (w_1, \ldots, w_n), (\sum_{i=1}^{n} w_i)/2)$. Es decir, además de los pesos de los elementos del conjunto, la constante que hay que alcanzar será la mitad del peso total, $\frac{w(A)}{2}$. Obviamente, si el peso del conjunto es impar, entonces el problema Partición tiene una respuesta negativa.

- *Reducción del problema Subset Sum al problema Knapsack*: Supongamos que disponemos de un algoritmo que resuelve el problema Knapsack. Entonces, para resolver el problema Subset Sum basta utilizar dicho algoritmo suministrándole como entrada la tupla $(n, (w_1, \ldots, w_n), (w_1, \ldots, w_n), k, k)$. Es decir, se considera que todos los elementos tienen su peso igual a su valor y que las dos constantes (capacidad de la mochila y valor mínimo) coinciden.

Uno de los problemas pendientes en Computación Celular es implementar estas reducciones dentro del modelo de los sistemas P, en lugar de necesitar un proceso

previo de computación clásica. Este problema no se abordará directamente en esta memoria, aunque está relacionado con la idea de permitir la reutilización de esquemas de reglas para más de un problema.

Por tanto, a pesar de que exista una reducidibilidad al problema Subset Sum, presentamos en este capítulo el diseño de una solución celular del problema Partición. Se ha elegido presentar este problema en último lugar en esta memoria porque, como se comentará en la siguiente sección, presenta una peculiaridad que incrementa la dificultad del diseño respecto a los otros dos problemas, relacionada con la uniformidad de la familia de sistemas P respecto del "tamaño" de cada instancia.

6.2. Una solución celular del problema de la Partición

El problema de la Partición se puede enunciar como sigue:

> *Dado un conjunto finito* $A = \{a_1, \ldots, a_n\}$*, donde cada elemento tiene asociado un "peso"* $w_i \in \mathbb{N}$*, determinar si existe una partición de* A*,* (B, B^c)*, de manera que* $\sum_{a_i \in B} w_i = \sum_{a_i \notin B} w_i$.

Representaremos las instancias del problema mediante tuplas de la forma $(n, (w_1, \ldots, w_n))$, siendo n el tamaño del conjunto A y $(w_1, \ldots, w_n)$ la lista de los pesos de sus elementos. Podemos definir de manera natural una función aditiva, w, que se corresponda con los datos de la instancia.

La solución que proponemos está basada en la implementación de un algoritmo de fuerza bruta en el marco de los sistemas celulares con membranas activas (en su variante con división binaria, usando cargas eléctricas, sin cooperación y sin prioridad entre las reglas). Para comprender mejor el diseño, conviene considerar el problema estructurado en varias etapas:

- *Fase de generación*: se genera, mediante divisiones, una membrana para cada par (B, B^c), siendo B un subconjunto de A que contenga al elemento a_1 (esta condición se impone para evitar considerar dos veces la misma pareja) y B^c su complementario.

- *Fase de cálculo*: en cada membrana se calcula el peso del subconjunto asociado y el de su complementario.

- *Fase de chequeo*: se comprueba en cada membrana si estos dos pesos coinciden.

- *Fase de respuesta*: se envía la respuesta al entorno, de acuerdo con los resultados de los chequeos.

El diseño de estas fases es muy similar al realizado en los capítulos anteriores para los problemas Subset Sum y Knapsack. Como ocurría entonces, en este caso la ejecución de las fases tampoco se lleva a cabo de manera secuencial y sincronizada. Es decir, etapas correspondientes a fases distintas pueden ser ejecutadas de manera simultánea pero independiente en distintas membranas.

Sin embargo, conviene resaltar algunas peculiaridades del problema de la Partición que condicionan la adaptación de los diseños previos.

En primer lugar, dado que en este caso queremos asociar a cada membrana un subconjunto que contenga al elemento a_1 (y el complementario de dicho subconjunto), resulta que en este caso sólo tenemos 2^{n-1} posibilidades, con lo que podemos simplificar un poco la fase de generación. En cuanto a la fase de cálculo, ahora hay que codificar también el peso del subconjunto complementario, lo cual se puede conseguir modificando ligeramente las reglas usadas para el problema Subset Sum.

Ahora bien, en las fases de chequeo y de respuesta surge una dificultad adicional. En los problemas anteriores se podía calcular una cota superior, en función de k y c, para el número de pasos que iba a tardar la comparación de los pesos de los subconjuntos con dichas constantes. Ahora, en cambio, ese número de pasos depende del peso total del conjunto A considerado en cada instancia, y no interesa que el diseño de las reglas dependa de la instancia. Por ello se ha ideado un nuevo sistema para garantizar que la respuesta se envía al entorno después de que todos los procesos internos han terminado, haciendo posible así la corrección de la respuesta y que ésta sea enviada en el último paso de la computación.

A continuación vamos a construir una familia $\mathbf{\Pi}$ de sistemas celulares reconocedores con membranas activas que *resuelva* (según la Definición 2.13) el problema de la Partición en tiempo *lineal*.

La familia que presentamos aquí es $\mathbf{\Pi} = \{(\Pi(n), \Sigma(n), i(n)) : n \in \mathbb{N}\}$, donde para cada elemento de la familia, el alfabeto de entrada es $\Sigma(n) = \{x_1, \ldots, x_n\}$, la membrana de entrada es $i(n) = e$, y el sistema celular $\Pi(n)$ viene dado por la tupla $(\Gamma(n), \{e, r, s\}, \mu, \mathcal{M}_e, \mathcal{M}_r, \mathcal{M}_s, R)$, que se define como sigue:

- Alfabeto de trabajo:

$$\Gamma(n) = \Sigma(n) \cup \{a_0, a, b, c, d_0, d_1, d_2, e_1, \ldots, e_n, g, g_0, g_1, h_0, h_1, p, p_0, q, q_0, q_1, q_2, q_3, Yes, No, No_0, z_1, \ldots, z_{2n+1}, \#\}$$

- Estructura de membranas: $\mu = [\ [\ ]_e^0\ [\ ]_r^0\ ]_s^0$.

- Multiconjuntos iniciales: $\mathcal{M}_e = e_1\, g_1$; $\mathcal{M}_r = b\, h_0$ y $\mathcal{M}_s = z_1$

- El conjunto de reglas de evolución, R, consta de las siguientes:

  (*a*) $[e_i]_e^0 \to [q]_e^-[e_i]_e^+$, para $i = 1, \ldots, n$.
  $[e_i]_e^+ \to [e_{i+1}]_e^0[e_{i+1}]_e^+$, para $i = 1, \ldots, n-1$.
  El objetivo de estas reglas es generar una membrana para cada par (B, B^c), siendo B un subconjunto de A. La carga de la membrana en cada instante y el índice del objeto e_i marcarán si un elemento de A se añade al primer subconjunto del par (es decir, a B) o bien si dicho elemento pasa a formar parte del segundo subconjunto. Además, para evitar que una misma pareja se considere dos veces, imponemos la condición de que $a_1 \in B$.

  (*b*) $[x_1 \to a_0]_e^0$; $\quad [x_1 \to p_0]_e^+$.
  $[x_i \to x_{i-1}]_e^+$, para $i = 2, \ldots, n$.
  $[x_i \to p]_e^-$, para $i = 2, \ldots, n$.
  Las reglas de la fase de cálculo usan la técnica de rotación de índices igual que en los dos capítulos anteriores, con los oportunos retoques para el peso del complementario del subconjunto asociado.

  (*c*) $[q \to q_0]_e^-$; $\quad [p_0 \to p]_e^-$; $\quad [a_0 \to a]_e^-$; $\quad [g_1]_e^- \to [\ ]_e^- g_0$.
  Al igual que ocurría en los dos capítulos anteriores, cuando terminan en una membrana las dos primeras fases (es decir, generación y cálculo) se produce un renombramiento de objetos para la siguiente fase. Además, se envía desde cada membrana a la piel un objeto g_0 que entrará en juego en la fase de respuesta.

  A partir del siguiente paso, diremos que la membrana es *relevante* (ver Definición 4.2).

  (*d*) $[e_n]_e^+ \to \#$; $\quad [a_0 \to \#]_s^0$; $\quad [p_0 \to \#]_s^0$; $\quad [g_1 \to \#]_s^0$.
  Estas reglas realizan una tarea de "limpieza" disolviendo las membranas que no nos son útiles y eliminando los objetos que, como resultado de dichas disoluciones, son liberados en la membrana piel. Más adelante se verá que es importante que sólo queden 2^{n-1} membranas etiquetadas por e en el sistema.

  (*e*) $[a]_e^- \to [\ ]_e^0\#$; $\quad [p]_e^0 \to [\ ]_e^-\#$.
  El peso del subconjunto se compara con el de su complementario mediante el bucle creado por estas dos reglas, como ocurría en los dos capítulos anteriores.

  (*f*) $[q_0 \to q_1]_e^-$; $\quad [q_1 \to q_0]_e^0$.
  En los casos del problema Subset Sum y del problema Knapsack, se usaba un

contador q_i cuyo índice variaba en un rango que dependía de las constantes k y c de dichos problemas. En este caso tenemos un marcador con tan sólo dos índices.

(*g*) $[q_0]^0_e \to [\ ]^+_e No_0$.

Si un subconjunto $B \subseteq A$ verifica $w(B) > w(B^c)$, entonces dentro de la membrana relevante asociada al par (B, B^c) habrá un menor número de objetos p que de objetos a. Esto hace que se detenga el bucle descrito por las reglas de (e) cuando se agoten los objetos p. En efecto, en ese instante la regla $[q_1 \to q_0]^-_e$ se aplicará, pero sin un objeto p cambiando la carga simultáneamente (mediante la regla $[p]^0_e \to [\ ]^-_e\#$). Por tanto, la membrana seguirá teniendo carga neutra y, además, contendrá un objeto q_0, con lo que la regla (g) se encarga de terminar la fase de chequeo con resultado negativo.

(*h*) $[q_1 \to q_2 c]^-_e$; $\quad [c]^-_e \to [\ ]^0_e\#$; $\quad [q_2 \to q_3]^0_e$.
$[q_3]^0_e \to [\ ]^+_e Yes$; $\quad [q_3]^-_e \to [\ ]^+_e No_0$.

Si, por el contrario, se verifica $w(B) \leq w(B^c)$, entonces los objetos a se agotarán antes que los objetos p. Es importante diferenciar los casos en los que la multiplicidad de p es estrictamente mayor que la de a de los casos en los que ambas multiplicidades coinciden. Por ello, el objeto c vuelve a darle carga neutra a la membrana y, entonces, el objeto q_3 comprueba si se ha aplicado o no la regla $[p]^0_e \to [\ ]^-_e\#$.

(*i*) $[p \to \#]^+_e$; $\quad [a \to \#]^+_e$.

Si después de terminar la fase de chequeo aún quedan objetos p u objetos a en la membrana, serán eliminados (para evitar futuras interacciones con la carga de la membrana).

(*j*) $[z_i \to z_{i+1}]^0_s$, para $i = 1, \ldots, 2n$.
$[z_{2n+1} \to d_0 d_1]^0_s$; $\quad d_0[\ ]^0_r \to [d_0]^-_r$; $\quad [d_1]^0_s \to [\ ]^+_s d_1$.

Como ya se ha dicho, en este capítulo se usará una nueva estrategia para controlar la salida. El proceso se activa mediante los objetos d_0 y d_1, tras $2n + 1$ pasos de computación. En ese instante ya se ha completado la fase de generación en todas las membranas relevantes y, por tanto, sabemos que no tendrán carga positiva hasta que no terminen su fase de chequeo.

(*k*) $[g_0 \to g]^+_s$; $\quad g[\ ]^+_e \to [g]^0_e$.

El núcleo de las operaciones de control de la fase de salida reside en los objetos g_0 que están presentes en la membrana piel, así como en la membrana auxiliar

r (ver el siguiente conjunto de reglas). Teniendo presente que cada membrana relevante envía un objeto g_0, y que tenemos una membrana relevante para cada $B \subseteq A$ tal que $a_1 \in B$, tendremos en total 2^{n-1} copias del objeto g_0.

(*l*) $[h_0 \to h_1]_r^-$; $[h_1 \to h_0]_r^+$; $[h_1 \to h_0]_r^0$.
$[b]_r^- \to [\ ]_r^+ b$; $g[\ ]_r^+ \to [g]_r^-$; $b[\ ]_r^- \to [b]_r^0$; $[g]_r^0 \to [\ ]_r^- g$.
$[h_0]_r^+ \to [\ ]_r^+ d_2$; $[d_2]_s^+ \to [\ ]_s^- d_2$.

La membrana etiquetada por r está presente en la configuración inicial, pero permanece inactiva hasta que un objecto d_0 la "despierta" (ver las últimas reglas del apartado (j)). El propósito de esta membrana es ejecutar un bucle en el que intervienen los objetos g, de manera que se pueda detectar si ya no quedan objetos g presentes en la membrana piel. Este hecho nos indicará que todas las membranas relevantes han terminado su fase de chequeo y que, por tanto, el sistema ya es capaz de enviar la respuesta al entorno (Yes o No).

(*m*) $[No_0 \to No]_s^-$; $[Yes]_s^- \to [\ ]_s^0 Yes$; $[No_0]_s^- \to [\ ]_s^0 No$.

Por último, para enviar la respuesta al entorno se usa la misma estrategia que en los capítulos anteriores: el objeto Yes puede salir directamente en cuanto la membrana piel tenga carga negativa, pero el objeto No_0 necesita un paso intermedio.

6.3. Seguimiento informal de la computación

Recordemos que para resolver un problema mediante una familia de sistemas celulares vamos a asociar a cada instancia del problema un multiconjunto (de entrada) y un número ("tamaño" de la instancia) de manera que a la hora de resolver una instancia concreta, se considera el sistema que corresponda al tamaño de la instancia, se le introduce el multiconjunto apropiado como entrada y, entonces, comienza la computación.

En nuestro caso, dada una instancia $u = (n, (w_1, \ldots, w_n))$ del problema de la Partición, consideramos la codificación polinomial formada por $s(u) = n$ como *función tamaño* y $cod(u) = x_1^{w_1} \ldots x_n^{w_n}$ como función que proporciona los *multiconjuntos de entrada*, de tal manera que el procesamiento de la instancia u (es decir, decidir si se acepta o se rechaza dicha instancia) se realiza a través del análisis de las computaciones del sistema $\Pi(s(u))$ con entrada $cod(u)$. A continuación describiremos informalmente el desarrollo de las mismas.

Las reglas del apartado (a), en las que intervienen los objetos e_i, son las que se aplican durante la *fase de generación*. Como se comentó brevemente en la sección

anterior, durante esta primera fase se producen divisiones de membranas encaminadas a obtener un número exponencial de membranas de trabajo. De este modo se puede lograr que todas las posibles particiones $A = B \cup B^c$ se evalúen en paralelo para comprobar si el peso de A se distribuye equitativamente o no.

En el problema de la Partición, a diferencia de lo que ocurría en los dos capítulos anteriores, no hay que comparar el peso de cada subconjunto con una constante dada, sino que hay que encontrar un subconjunto cuyo peso sea igual al de su complementario. Por ello, en este caso en lugar de considerar para cada membrana un *subconjunto asociado* (Definición 4.1), generaremos una membrana para cada pareja (B, B^c), siendo B un subconjunto de A que contiene al elemento a_1 (para evitar que una misma pareja se considere dos veces). Es decir, el número de posibilidades que hay que evaluar se reduce a la mitad, se generarán 2^{n-1} membranas de trabajo, en lugar de 2^n como ocurría en los capítulos anteriores. Por este motivo los índices de los objetos e_i empiezan ahora en 1, en lugar de empezar en 0.

El método que vamos a seguir consiste en ir actualizando simultáneamente dos subconjuntos en cada membrana a lo largo de la fase de generación, facilitando de esta manera que se ejecute al mismo tiempo la fase de cálculo. Los elementos de A serán añadidos en una membrana al subconjunto *primario* (que se corresponde con el subconjunto asociado de la Definición 4.1), o al *secundario* (que terminará siendo el complementario del anterior), siguiendo las siguientes directrices:

- Si en una membrana con carga neutra aparece el objeto e_i $(1 \leq i \leq n)$, entonces el elemento $a_i \in A$ pasa a formar parte del subconjunto primario de dicha membrana.
- Si el objeto e_i $(1 \leq i < n)$ aparece en una membrana cargada positivamente, entonces el elemento $a_i \in A$, si no ha sido asignado ya al subconjunto primario de la membrana, pasa a formar parte del subconjunto secundario.
- Por último, cuando en una membrana aparece el objeto q (y la membrana tiene carga negativa) se asignan todos los elementos de A no considerados aún al subconjunto secundario, para conseguir que los dos subconjuntos sean complementarios.

Las membranas relevantes (Definición 4.2) asociadas a los distintos pares (B, B^c) irán apareciendo en este caso también por orden lexicográfico, en cierto sentido, como sucedía en los capítulos anteriores. Esto es, si un elemento a_j ya ha sido considerado y se ha añadido al subconjunto asociado o al secundario, entonces no se añadirá a ninguno de ellos, posteriormente, ningún elemento $a_{j'}$ con $j' < j$. En total se obtendrán exactamente 2^{n-1} membranas relevantes, todas ellas distintas entre sí.

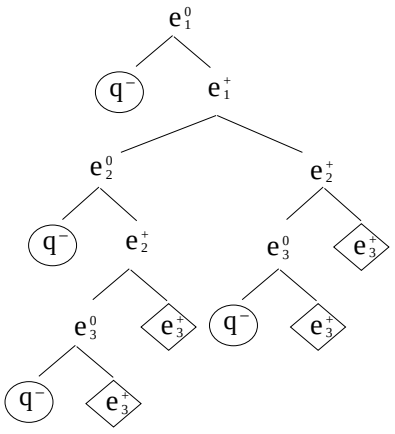

Figura 6.2: Fase de generación para $n = 3$

Al aplicar una regla del tipo $[e_i]_e^0 \rightarrow [q]_e^-[e_i]_e^+$, las dos membranas creadas heredan el subconjunto primario y el secundario de la membrana original. La membrana donde aparece el objeto q abandona la fase de generación, y ya no se volverá a dividir en lo que resta de computación (dichas membranas aparecen en la Figura 6.2 señaladas con un círculo). Por otra parte, la membrana creada en la que aparece un objeto e_i, y que tiene carga positiva, seguirá dividiéndose por la aplicación de la regla $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0[e_{i+1}]_e^+$, con objeto de producir membranas asociadas a nuevas particiones, obtenidas añadiendo al subconjunto primario actual objetos de índice $i+1$ o superior. Obsérvese que en el caso $i = n$ la membrana se queda bloqueada, puesto que no quedan más elementos que añadir (esta circunstancia la hemos señalado con un rombo en la Figura 6.2).

La *fase de cálculo* se ejecuta en paralelo con la fase de generación. Veamos a continuación cómo funciona esta fase.

Para hallar el peso de un subconjunto, hay que sumar los pesos de todos sus elementos. El sistema está diseñado de tal manera que la suma de los pesos de los elementos se efectúa al mismo tiempo que dichos elementos van siendo añadidos, bien al subconjunto asociado a la membrana, o bien al subconjunto secundario. El proceso continúa hasta llegar a una membrana relevante y, entonces, el número de copias del objeto a coincidirá con el peso del subconjunto asociado y el número de copias del objeto p coincidirá con el peso del complementario.

En la siguiente sección se demostrará que en toda membrana con carga positiva (y etiquetada por e) donde aparezca el objeto e_i, con $1 \leq i < n$, la multiplicidad del objeto x_2 será igual al peso del elemento $a_{i+1} \in A$. Por tanto, cuando en el

siguiente paso de computación se apliquen las reglas $[x_2 \rightarrow x_1]_e^+$, $[x_1 \rightarrow p_0]_e^+$ y $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0[e_{i+1}]_e^+$, obtendremos en cada una de las membranas resultantes exactamente $w(a_{i+1})$ apariciones de x_1.

En el caso de la membrana con carga neutra, el elemento a_{i+1} es añadido al subconjunto asociado y en el siguiente paso se producirán $w(a_{i+1})$ nuevas copias de a_0 en la membrana. La situación es distinta para la otra membrana que resulta de la división (con carga positiva), donde el elemento a_{i+1} no será añadido al subconjunto asociado a la membrana y, por tanto, pasa al subconjunto secundario, con lo que se producen $w(a_{i+1})$ nuevas copias de p_0 en la membrana. En el siguiente paso el número de copias del objeto x_1 es igual a $w(a_{i+2})$, o bien ya no queda ningún objeto x_1, en el caso de que $i + 1 = n$.

A continuación se desarrolla la *fase de chequeo*. Esta fase comienza a ejecutarse en una membrana cuando ésta llega a ser relevante; es decir, esto sucede exactamente un paso después de que aparezca el objeto q y de que la membrana esté cargada negativamente por primera vez en la computación. En ese paso de transición tienen lugar varios renombramientos de objetos (por la aplicación de las reglas del grupo (c) y las del grupo (b) para membranas con carga negativa): a_0 y p_0 pasan a ser a y p, respectivamente, los objetos x_i que queden aún en la membrana también se transforman en objetos p (recordemos que todos los elementos de A que no hayan sido añadidos ni al subconjunto primario ni al secundario pasan al complementario), y el objeto q se transforma en q_0, con lo que la membrana pasa a ser relevante. Además, se envía desde cada membrana a la piel un objeto g_0 que entrará en juego en la fase de respuesta.

Seguidamente, se comprueba si hay el mismo número de objetos a y p: son contados uno a uno alternativamente, cambiando la carga de la membrana en cada paso, de negativa a neutra y viceversa, como indican las reglas del apartado (e).

El número exacto de pasos que dará el bucle de chequeo en cada membrana dependerá del peso del subconjunto asociado, pero estará siempre acotado por el peso total de A. En este caso es imposible disponer de un contador similar al de los capítulos anteriores para controlar el resultado de la comparación, ya que el conjunto de índices de ese contador y, por tanto, las reglas que controlan su comportamiento, dependerían del peso de A. Esa es una información concreta de cada instancia, con lo que el diseño sería mucho menos uniforme: cada sistema de la familia resolvería el problema para conjuntos A que tuvieran, no sólo el mismo número, n, de elementos (como es nuestro caso), sino también el mismo peso total, $w(A)$.

Las reglas de los apartados (f), (g) y (h) son las encargadas de controlar el resultado de la comparación, y permiten discernir en cada membrana si el peso del

subconjunto asociado era mayor, menor o igual al de su complementario. Estas ocho reglas se comportan como un sistema bi-estable; es decir, mientras que la carga de la membrana siga cambiando en cada paso por efecto de las reglas de (e), el marcador q_i oscila entre $i = 0$ e $i = 1$. Ahora bien, cuando la membrana mantenga la misma polaridad durante dos pasos consecutivos, eso será indicativo de que el bucle de comparación (reglas de (e)) se ha detenido. En ese caso las reglas de (g) y (h) se ocupan de que se envíe un objeto Yes a la piel si el peso del subconjunto asociado a la membrana era igual al de su complementario y de que, en caso contrario, se envíe un objeto No_0. Este proceso será descrito con más detalle en la siguiente sección.

Por último, pasemos a comentar en qué consiste la *fase de respuesta*. Ya se adelantó en la sección anterior que hay una membrana auxiliar en el sistema, etiquetada por r, que tendrá un papel central en la fase de respuesta. Expliquemos a continuación el proceso paso a paso (ver la Figura 6.3 para una descripción gráfica del bucle detector).

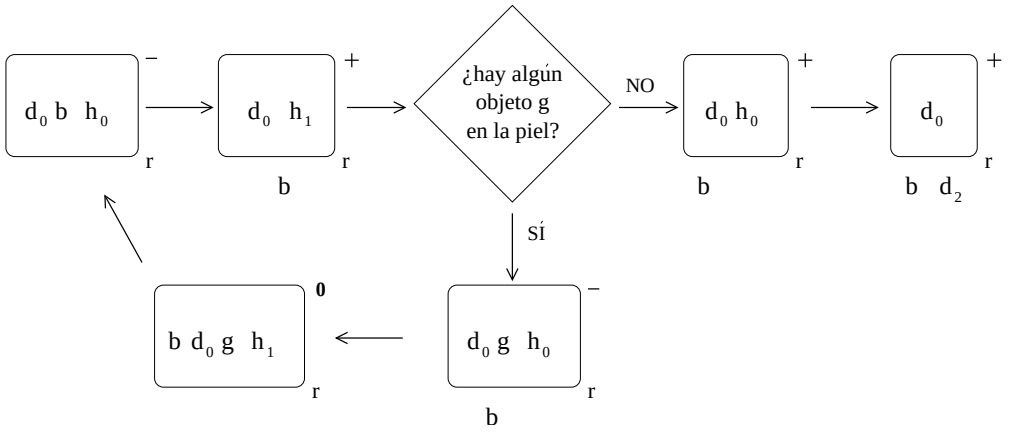

Figura 6.3: Bucle detector de la membrana r

Antes de poder dar una respuesta fiable, el sistema debe comprobar que las 2^{n-1} membranas relevantes han terminado sus fases de chequeo y, para ello, recurriremos a 2^{n-1} objetos g_0 y haremos uso del paralelismo masivo del modelo. Esta cantidad exponencial de objetos se obtiene como sigue: partimos de un objeto g_1, presente en la configuración inicial dentro de la membrana e, que es replicado cada vez que se aplica una regla de división y luego, cuando la membrana llega a ser relevante, se aplica la regla $[g_1]_e^- \rightarrow [\ ]_e^- g_0$.

La idea consiste en hacer que estos objetos vuelvan a entrar en las membranas etiquetadas por e a medida que éstas van terminando sus fases de chequeo (obsérvese que, con independencia del resultado del chequeo, las membranas siempre obtienen

carga eléctrica positiva al final). Para evitar interferencias entre este proceso y la fase de generación, durante la cual las membranas también tienen carga positiva en algunos instantes, se esperan $2n+2$ pasos (usando el contador z_i) antes de generar los objetos d_0 y d_1 en la piel. En ese instante se activa la membrana r, mediante la regla $d_0[\ ]_r^0 \rightarrow [d_0]_r^-$ y, en el mismo paso, se envía fuera del sistema el objeto d_1 cambiando la polaridad de la membrana piel (mediante la regla $[d_1]_s^0 \rightarrow [\ ]_s^+ d_1$).

Como resultado del cambio de carga de la piel se ejecuta la regla de renombramiento $[g_0 \rightarrow g]_s^+$. Al mismo tiempo, al pasar a tener carga negativa, la membrana r comienza a ejecutar su "bucle de detección de objetos g", descrito por las reglas del apartado (l) (ver Figura 6.3). Intuitivamente, la membrana r trata de *pescar* los objetos g presentes en la piel, usando el objeto b como *cebo*: la regla $[b]_r^- \rightarrow [\ ]_r^+ b$ cambia la carga de la membrana r a positiva y, por tanto, en el siguiente paso se podría aplicar la regla $g[\ ]_r^+ \rightarrow [g]_r^-$ (es claro que si no quedasen objetos g en la piel, entonces no se podría aplicar la regla). Hay un marcador h_i dentro de la membrana r que controla si algún objeto g entra en la membrana. En caso afirmativo, el bucle sigue su curso hasta volver a la situación inicial, pero en caso negativo se genera un objeto d_2 que es enviado a la piel para terminar la fase de respuesta.

Una vez que el objeto d_2 es enviado fuera del sistema y la piel pasa a tener carga negativa, de acuerdo con lo que se acaba de decir acerca del bucle detector de la membrana r, se puede garantizar que todas las membranas han terminado sus correspondientes fases de chequeo. Por tanto, no queda más que comprobar si alguna de ellas produjo un objeto Yes y enviar la respuesta adecuada al exterior. De esto se ocupan las reglas del apartado (m), de manera similar al caso de los problemas Subset Sum y Knapsack.

6.4. Verificación Formal

En esta sección se trata de probar que la familia $\mathbf{\Pi}$ de sistemas celulares con membranas activas diseñados en la Sección 6.2 proporciona una solución celular polinomial para el problema de la Partición, en el sentido de la Definición 2.13; esto es, $\mathit{Partición} \in \mathbf{PMC}_{\mathcal{AM}}$.

En primer lugar, hay que justificar que la familia definida es $\mathcal{AM}$-consistente; es decir, que todos los sistemas de la familia son sistemas P reconocedores de lenguajes con membranas activas. Es claro que se han diseñado sistemas con entrada y con salida externa que usan membranas activas, y que el alfabeto de trabajo contiene los objetos Yes y No. Por tanto, se trata de sistemas aceptadores de lenguajes. Al igual que se hizo en los capítulos anteriores, para probar que se trata de sistemas reconocedores,

se demostrará que todas las computaciones paran y lo hacen de manera confluente. Es decir, o bien todas las computaciones asociadas a un multiconjunto de entrada son de aceptación (envían un objeto Yes al entorno y en el último paso de la computación), o bien todas son de rechazo (en lugar del objeto Yes se envía No).

6.4.1. Uniformidad polinomial de la familia

A continuación, veamos que la familia es polinomialmente uniforme por máquinas de Turing. Se puede observar que la definición de la familia está hecha de manera recursiva a partir de las instancias, en particular a partir del valor de la constante n. Además, los recursos *iniciales* necesarios para construir un elemento de la familia son de orden lineal con respecto a dicha constante:

- tamaño del alfabeto: $3n + 23 \in \Theta(n)$,
- número de membranas: $3 \in \Theta(1)$,
- $|\mathcal{M}_e| + |\mathcal{M}_r| + |\mathcal{M}_s| = 5 \in \Theta(1)$,
- número total de reglas: $6n + 36 \in \Theta(n)$.

Antes de pasar a ocuparnos de la tercera condición de la Definición 2.13, recordemos que las funciones *cod* y s se han definido en la sección anterior para una instancia $u = (n, (w_1, \ldots, w_n))$ como sigue: $cod(u) = x_1^{w_1} \ldots x_n^{w_n}$ y $s(u) = n$, respectivamente.

Ambas funciones son totales y polinomialmente computables. Más aún, el par (cod, s) forma una codificación polinomial del conjunto de instancias del problema de la Partición en la familia $\mathbf{\Pi}$, ya que se verifica que, para cualquier instancia u, el multiconjunto $cod(u)$ es una entrada válida para el sistema $\Pi(s(u))$.

Obsérvese que cada instancia $u = (n, (w_1, \ldots, w_n))$ es introducida en la configuración inicial de su sistema celular asociado mediante un multiconjunto de entrada (es decir, en una representación 1-aria) y, por tanto, se tiene que $|u| \in O(w_1 + \cdots + w_n)$.

Ahora bien, de manera similar a como se hizo en los dos capítulos anteriores, estudiaremos la computación del sistema $\Pi(s(u))$ con entrada $cod(u)$ para una instancia genérica u del problema de la Partición, y probaremos, mediante una serie de proposiciones, que se verifica la tercera condición de la Definición 2.13

6.4.2. Acotación polinomial de la familia

En primer lugar, procedemos a justificar que el número de pasos (celulares) es de orden lineal respecto al tamaño del dato de entrada.

Proposición 6.1 *La familia $\mathbf{\Pi} = (\Pi(t))_{t \in \mathbb{N}}$, definida en la Sección 6.2, está polinomialmente acotada respecto a $(Partición, cod, s)$.*

Demostración. Para cerciorarnos de que el sistema $\Pi(s(u))$ con entrada $cod(u)$ está polinomialmente (de hecho, linealmente) acotado, basta encontrar el instante en que la computación para o, al menos, una cota superior del mismo. Como veremos a continuación, el número de pasos de las computaciones de los sistemas de la familia pueden acotarse siempre por una función *lineal* sobre $|u|$. Sin embargo, conviene mencionar que la cantidad de recursos pre-computados para cada instancia u es polinomial en el tamaño de la misma, puesto que los valores $cod(u)$ y $s(u)$ necesitan ser calculados y el sistema $\Pi(s(u))$ debe ser construido.

En primer lugar, consideremos la *fase de generación* junto con la *fase de cálculo*. Las reglas que gobiernan estas fases son las de los apartados (a) y (b), que son claramente análogas a las de los mismos apartados en los diseños de soluciones para los problemas Subset Sum y Knapsack. Por tanto, siguiendo un razonamiento idéntico al de los dos capítulos anteriores, pero teniendo en cuenta que esta vez se considera un índice menos (no hay objetos e_0 ni x_0 en el alfabeto), concluimos que tras $2n-1$ pasos ya no se producirán más divisiones. Las membranas finalizan su fase de generación en el instante en que obtengan por primera vez carga negativa, y la fase de cálculo finaliza un paso después, tras la aplicación (simultánea) de las reglas $[x_i \rightarrow p]_e^-$, para todos los objetos x_i (con $2 \leq i \leq n$) que aún queden en la membrana. Si no queda ninguno, entonces las dos primeras fases finalizan conjuntamente. Es decir, tras $2n$ pasos podemos asegurar que ya no se ejecutará ninguna regla de los apartados (a) y (b).

Una vez que la membrana obtiene carga negativa, son aplicables las reglas de renombramiento del apartado (c), que generan en la membrana una serie de objetos a y p, junto con un objeto q_0 (con lo que la membrana pasa a ser *relevante*). Además, como sabemos que en ese paso ha concluido la fase de cálculo, las multiplicidades de los objetos a y p coinciden con los pesos del subconjunto asociado y de su complementario, respectivamente.

Pasemos ahora a estudiar la *fase de chequeo*. A lo largo de esta fase se comparan las multiplicidades de los objetos a y p, mediante las reglas del apartado (e). Dichas reglas se comportan como un bucle, en el siguiente sentido: en cada paso del bucle cambia la carga de la membrana, volviendo a restablecerse en el siguiente paso, de manera que tras cada vuelta del bucle (es decir, cada dos pasos de computación) hay un par de objetos menos. Este proceso continúa hasta que se agoten las copias de uno de los dos objetos (o de ambos). Dado que los objetos a y p representan los pesos de un subconjunto de A y de su complementario, la cantidad total de estos objetos será igual a $w(A)$. Tenemos, por tanto, que el peso de A es una cota superior para

el número de pasos del bucle de chequeo y, consecuentemente, se trata de un proceso finito.

Las reglas de los apartados (g) y (h), encargadas de finalizar la fase de chequeo, se aplican en un número finito de pasos. Estas reglas sólo pasan a ser aplicables en una membrana si durante la fase de chequeo de dicha membrana la carga eléctrica se mantiene igual durante dos pasos seguidos. El mecanismo de cambio de carga actúa en forma de bucle, como sigue:

1. Inicialmente (cuando la membrana llega a ser relevante) la carga es negativa y el índice del marcador q_i es $i = 0$. Entonces, en el siguiente paso se elimina un objeto a, el índice de q_i pasa a ser $i = 1$ y la carga pasa a ser neutra.

2. A continuación se aplica la regla $[p]_e^0 \rightarrow [\ ]_e^-\#$, eliminando un objeto p y restableciendo la carga negativa. Además, simultáneamente el marcador q_i vuelve a tener índice $i = 0$.

Este proceso continúa mientras queden objetos a y p en la membrana.

Si se agotan primero los objetos a, entonces llegará un momento en el que la carga de la membrana sea negativa durante dos pasos consecutivos, porque no se pueda pasar de 1) a 2) en el bucle. El sistema detecta que la carga no ha variado cuando el objeto q_1 está en la membrana y ésta tiene carga negativa. La fase de chequeo ejecutará cuatro pasos más y finalizará en esa membrana: un paso para la regla $[q_1 \rightarrow q_2 c]_e^-$; otro para la regla $[c]_e^- \rightarrow [\ ]_e^0\#$; otro para la regla $[q_2 \rightarrow q_3]_e^0$ (y al mismo tiempo también $[p]_e^0 \rightarrow [\ ]_e^-\#$, si queda algún objeto p) y, finalmente, se aplica o bien la regla $[q_3]_e^0 \rightarrow [\ ]_e^+ Yes$, o bien $[q_3]_e^- \rightarrow [\ ]_e^+ No_0$.

Si se agotan primero los objetos p, entonces llegará un momento en el que la carga de la membrana sea neutra durante dos pasos consecutivos, porque no se pueda pasar de 2) a 1) en el bucle. En este caso el sistema detecta que la carga no ha variado cuando el objeto q_0 está en la membrana y ésta tiene carga neutra. La fase de chequeo ejecutará un solo paso más y finalizará en esa membrana: se aplicará la regla $[q_0]_e^0 \rightarrow [\ ]_e^+ No_0$.

En ambos casos la fase termina dejando la membrana cargada positivamente, y entonces se aplican (en un solo paso) las reglas del apartado (i), para eliminar los objetos sobrantes de la comparación.

Por último, veamos que la *fase de respuesta*, en la que intervienen las reglas de los apartados (j), (k), (l) y (m), es también un proceso finito. En primer lugar, mencionemos el contador z_i: su tarea consiste en activar el "bucle detector de objetos g" después de que todas las membranas hayan terminado su fase de generación. Tras

$2n+1$ pasos (empezando a contar desde el inicio de la computación) ya no quedarán objetos z_i en el sistema, y aparecerán en la piel los objetos especiales d_0 y d_1.

El objeto d_0 cambia la carga de la membrana r, haciendo que se vuelvan aplicables las reglas $[h_0 \to h_1]_r^-$ y $[b]_r^- \to [\,]_r^+ b$. Al mismo tiempo, el objeto d_1 es enviado fuera del sistema, dejando la membrana piel cargada positivamente, con lo que se aplicará en el siguiente paso el renombramiento $[g_0 \to g]_s^+$ sobre las 2^{n-1} copias de g_0 que hay en la piel.

Las reglas de (l), relativas a la membrana r, dejarán de ser aplicables un número finito de pasos después de que la última membrana relevante haya acabado su fase de chequeo. En efecto, el "bucle detector de objetos g" tiene un ciclo de cuatro pasos de computación (ver Figura 6.3):

1. *Sale el objeto b (cambio de carga negativa a positiva)*
2. *Entra un objeto g (cambio de carga positiva a negativa)*
3. *Entra el objeto b (cambio de carga negativa a neutra)*
4. *Sale un objeto g (cambio de carga neutra a negativa)*

Nótese que durante el paso de *1*) a *2*) no hay ningún objeto g dentro de la membrana r, y no es aplicable la regla $g[\,]_r^+ \to [g]_r^-$. Por tanto, en cada vuelta del bucle hay un momento en el que, si quedan objetos g en la piel y la regla $g[\,]_e^+ \to [g]_e^0$ es aplicable, entonces debe ser aplicada obligatoriamente en todas las membranas etiquetadas por e con carga positiva (que, recordemos, son membranas relevantes que han terminado su chequeo) porque no hay ninguna otra regla aplicable en ese instante para g.

Así, el bucle detector de la membrana r parará, a lo sumo, cuatro pasos después de que la última membrana relevante haya finalizado su fase de chequeo. Al terminar el bucle, se envía un objeto d_2 a la piel, y en el siguiente paso dicho objeto es expulsado al entorno, cambiando la carga de la membrana piel a negativa. Entonces, si existe algún objeto Yes en la piel el sistema parará tras ejecutar un último paso (en el que se aplican las reglas $[No_0 \to No]_s^-$ y $[Yes]_s^- \to [\,]_s^0 Yes$) y si, por el contrario, no existe ningún objeto Yes en la piel, entonces el sistema parará tras ejecutar dos pasos de computación (en el primero se aplica la regla $[No_0 \to No]_s^-$ y en el segundo se aplica $[No_0]_s^- \to [\,]_s^0 No$).

En resumen, la computación de un sistema $\Pi(s(u))$ con entrada $cod(u)$ parará en, a lo sumo, $2n + w(A) + 11$ pasos:

- Tras $2n-1$ pasos no se producirán más divisiones (fin de la fase de generación).
- La fase de cálculo termina, a lo sumo, un paso después de la de generación.

- La fase de chequeo ejecutará, a lo sumo, $w(A)/2$ ciclos del bucle de comparación (cada ciclo son dos pasos) y luego son necesarios un máximo de cinco pasos para decidir si se envía a la piel un objeto Yes o un objeto No_0.
- El bucle detector de la membrana r parará, a lo sumo, cuatro pasos después de que la última membrana relevante haya finalizado su fase de chequeo.
- Si la respuesta es afirmativa se ejecuta un paso más, pero si es negativa se necesitan dos pasos más.

□

6.4.3. Adecuación y completitud de la familia

A continuación, probaremos que los sistemas de la familia son adecuados y completos con respecto a $(Partición, cod, s)$, de acuerdo con la Definición 2.13. Es decir, vamos a probar que dada una instancia, u, del problema, el sistema P de la familia asociado a dicha instancia, $\Pi(s(u))$ con entrada $cod(u)$, envía fuera del sistema en el último paso de computación un objeto Yes si y sólo si la instancia considerada tiene respuesta afirmativa. También veremos que, en caso contrario, se envía un objeto No.

Proposición 6.2 *La familia* $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$*, definida en la Sección 6.2, es adecuada y completa respecto a* $(Partición, cod, s)$*.*

Demostración. La prueba se hará estableciendo el correcto funcionamiento de cada fase por separado.

Lema 6.1 (fase de generación) *Existe una única membrana relevante asociada a cada partición* (B, B^c)*, siendo* $B \subseteq A$ *tal que* $a_1 \in B$*.*

Demostración. Consideremos un subconjunto arbitrario $B = \{a_{i_1}, \ldots, a_{i_r}\}$ (con $1 = i_1 < i_2 < \cdots < i_r \leq n$). Dado que las reglas de la fase de generación son análogas a las utilizadas en los anteriores capítulos, podemos seguir el mismo razonamiento que entonces, considerando que el subconjunto asociado refleja de algún modo la *historia* de la membrana. Así, el subconjunto asociado a una membrana queda unívocamente determinado por los índices de los objetos e_i en los instantes de la evolución de la membrana en los que ha tenido carga neutra.

Por ejemplo, en la Figura 6.4 aparece la sucesión que conduce a la partición (B, B^c), para $n = 4$ y $B = \{a_1, a_3, a_4\}$.

Razonando como en los capítulos anteriores, concluimos que a lo largo de la computación aparecen 2^{n-1} membranas relevantes, y cada una está asociada a una partición de A distinta.

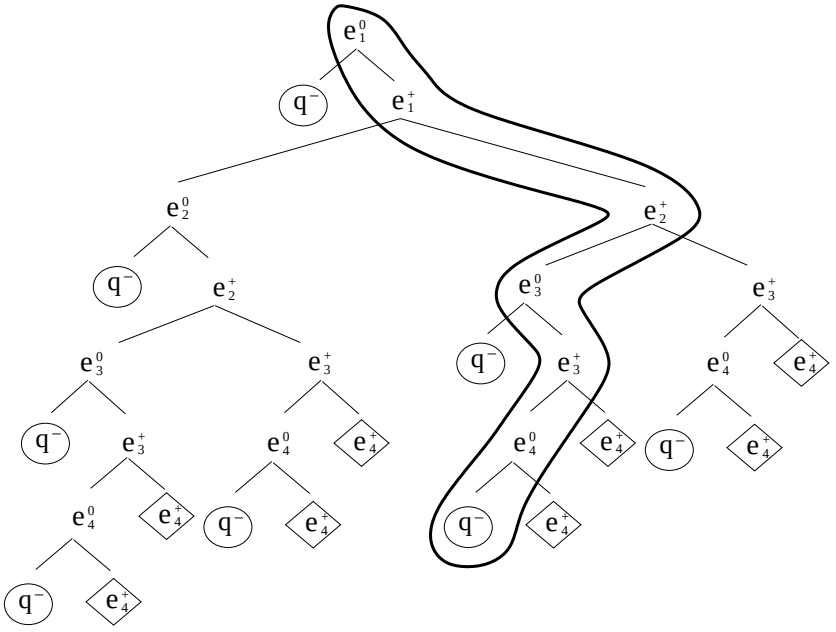

Figura 6.4: Membrana relevante asociada a la partición $(\{a_1, a_3, a_4\}, \{a_2\})$

□

A continuación, para asegurarnos de que la segunda fase también funciona correctamente, veremos que las multiplicidades de los objetos a y p en la membrana antes mencionada son $w(B)$ y $w(B^c)$, respectivamente.

Lema 6.2 *En una membrana interna con carga positiva en la que aparezca un objeto e_i, con $1 \leq i \leq n-1$, la multiplicidad del objeto x_j coincide con el peso del elemento $a_{i+j-1} \in A$, para cada j tal que $2 \leq j \leq n+1-i$.*

Demostración. Por inducción sobre i.

Para el caso base, $i = 1$, se sabe que el multiconjunto de la membrana etiquetada por e, en la configuración inicial, es $e_1 g_1 x_1^{w_1} \cdots x_n^{w_n}$. Entonces se aplica la regla $[e_1]_e^0 \rightarrow [q]_e^- [e_1]_e^+$ (de hecho, esa es la única regla que se puede aplicar sobre ese multiconjunto de objetos) y se obtienen dos membranas: una de ellas con carga negativa, y otra con carga positiva que contiene al multiconjunto de objetos $e_1 g_1 a_0^{w_1} x_2^{w_2} \cdots x_n^{w_n}$. En ese instante, para cada j tal que $1 \leq j \leq n$, la multiplicidad del objeto x_j es exactamente el peso del elemento $a_j \in A$.

Para establecer el paso inductivo, supongamos que el resultado es cierto para $i < n-1$. Consideremos una membrana con carga positiva y etiquetada por e que contenga un objeto e_{i+1}. Hemos de probar que la multiplicidad del objeto x_j coincide

con el peso del elemento $a_{(i+1)+j-1} \in A$, para cada j tal que $2 \leq j \leq n+1-(i+1)$. Distinguiremos dos casos:

1. Supongamos, en primer lugar, que la membrana considerada ha sido obtenida a partir de la aplicación de la regla $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0[e_{i+1}]_e^+$. Por hipótesis de inducción se sabe que las multiplicidades de los objetos x_j en la membrana original (en la que aparece e_i) coinciden exactamente con los pesos de los elementos $a_{i+j-1} \in A$, para todo j tal que $2 \leq j \leq n+1-i$. Obsérvese que junto con la regla de división también se han aplicado en ese paso las reglas $[x_i \rightarrow x_{i-1}]_e^+$, para $i = 2, \ldots, n$, y $[x_1 \rightarrow p_0]_e^+$. Por tanto, las multiplicidades de los objetos x_j en nuestra membrana serán iguales a las multiplicidades de los objetos x_{j+1} en la membrana original. Así pues, de este hecho y de la hipótesis de inducción se deduce que las multiplicidades de los objetos x_j coinciden exactamente con los pesos de los elementos $a_{i+j} \in A$, para $2 \leq j \leq n-i$.

2. Supongamos ahora que la membrana considerada se ha creado mediante la aplicación de la regla $[e_{i+1}]_e^0 \rightarrow [q]_e^-[e_{i+1}]_e^+$. Se puede seguir un razonamiento similar al anterior para la membrana original (en este caso la membrana original es $[e_{i+1}]_e^0$) ya que ésta ha sido obtenida a su vez como resultado de la aplicación de la regla $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0[e_{i+1}]_e^+$. Se concluye entonces que las multiplicidades de los objetos x_j en la membrana $[e_{i+1}]_e^0$ coinciden exactamente con los pesos de los elementos $a_{i+j} \in A$, para $2 \leq j \leq n-i$. Por último, téngase presente que la multiplicidad de dichos objetos x_j no varía en el último paso, ya que la única regla que se podría aplicar conjuntamente con la regla de división sería $[x_1 \rightarrow a_0]_e^0$, pero ésta no afecta a los objetos x_j con $j \geq 2$.

□

Lema 6.3 (fase de cálculo) *En una membrana relevante el número de copias de los objetos a y p coinciden con los pesos de los subconjuntos primario y secundario, respectivamente.*

Demostración. Dada una membrana relevante arbitraria, sea $B = \{a_{i_1}, \ldots, a_{i_r}\}$ su subconjunto asociado, con $i_1 < i_2 < \cdots < i_r$, y $r \leq n$. Entonces no existe, a lo largo de la computación, ninguna otra membrana relevante asociada al mismo par (B, B^c).

En la sucesión de membranas que conduce desde la membrana inicial hasta la membrana relevante considerada, existen $r + i_r + 1$ membranas intermedias. Todas ellas tienen carga positiva excepto algunas que tienen carga neutra y contienen un objeto e_{i_l}. Concretamente, hay una por cada $a_{i_l} \in B$(con $1 \leq l \leq r$). Además,

para todo j con $1 \leq j \leq i_r - 1$, hay una membrana cargada positivamente que contiene al objeto e_j. Por último, también aparecen en la sucesión dos membranas cargadas negativamente (las dos últimas) que contienen al objeto q y al objeto q_0, respectivamente.

Apliquemos el Lema 6.2 para un i arbitrario (con $1 \leq i \leq i_r - 1$) y para $j = 2$, a la membrana cargada positivamente que contiene al objeto e_i. Resulta que en dicha membrana, dado que contiene al objeto e_i, la multiplicidad del objeto x_2 es exactamente w_{i+1}; es decir, el peso del elemento $a_{i+1} \in A$. La siguiente membrana de la sucesión contiene al objeto e_{i+1}, y o bien tiene carga neutra (si a_{i+1} pertenece a B) o bien positiva (si a_{i+1} pertenece a B^c); es decir, la siguiente membrana de la sucesión es una de las que resulta de aplicar la regla $[e_i]_e^+ \rightarrow [e_{i+1}]_e^0[e_{i+1}]_e^+$.

Nótese que, en paralelo a esa regla de división, también se aplicarán las reglas de (b), con lo que se deduce que en la siguiente membrana la multiplicidad del objeto x_1 es w_{i+1}. Consecuentemente, en el siguiente paso se generarán w_{i+1} copias de a_0 (si la membrana tiene carga neutra, y a_{i+1} es añadido en ese momento al subconjunto B) o bien w_{i+1} copias de p_0 (si la membrana tiene carga positiva, y a_{i+1} es añadido en ese momento al subconjunto B^c). Esto se verifica para todo i_l con $1 \leq l \leq r$ (es decir, para todos los elementos del subconjunto primario, B), y también para todo j con $1 \leq j \leq i_r - 1$ tal que $j \notin \{i_1, \ldots, i_r\}$ (es decir, para todos los elementos del subconjunto secundario, B^c, que hayan sido considerados antes de que la membrana obtenga carga neutra.

El sistema calcula los pesos del resto de los elementos de A (los que tienen índice mayor que i_r, si existen) cuando la membrana obtiene carga negativa. En ese momento finaliza la fase de generación, y por tanto todos los elementos que no hayan sido considerados aún (aquellos con índice j tal que $i_r < j \leq n$) son añadidos al subconjunto secundario. Entonces, dado que la membrana ya tiene carga neutra, se aplican las reglas de renombramiento del apartado (c), transformando los objetos a_0 y p_0 en objetos a y p, respectivamente. Además, en el mismo paso, los pesos de los elementos de A que no hayan sido considerados aún se suman al peso del subconjunto secundario (en efecto, todos los objetos x_j que estén presentes en la membrana se transforman en copias del objeto p mediante la aplicación de las reglas del apartado (b) referidas a condición de carga negativa).

Concluimos por tanto que en la membrana con carga negativa donde aparece el objeto q_0 (la membrana relevante, la última de la sucesión), la multiplicidad del objeto a es la suma de los pesos de los elementos del subconjunto B (esto es, el peso de B) y la multiplicidad del objeto p es la suma de los pesos del resto de los elementos de A (esto es, el peso de B^c).

□

A continuación se estudia la fase de chequeo.

Lema 6.4 (fase de chequeo) *Una membrana relevante, asociada a una partición* (B, B^c) *(siendo* $B \subseteq A$ *tal que* $a_1 \in A$*), superará con éxito la fase de chequeo (mandando a continuación un objeto* Yes *a la piel) si y sólo si se verifica* $w(B) = w(B^c)$.

Demostración. Para probar este lema, consideremos la membrana relevante arbitraria y sean $w(B) = w_1$ y $w(B^c) = w_2$ los pesos del subconjunto asociado, B, y de su complementario, B^c, respectivamente. Entonces el multiconjunto de objetos contenido en dicha membrana es $q_0 a^{w_1} p^{w_2}$.

Se trata de comparar el número de objetos p con el número de objetos a presentes en la membrana. El mecanismo de comparación está basado en las reglas del apartado (e), que funcionan de manera análoga a las que se encargaban de las fases de chequeo en los capítulos anteriores.

Ahora bien, a la hora de finalizar la fase de chequeo nos encontramos con una dificultad adicional. Como ya se adelantó en la sección anterior, en el caso del problema de la Partición no disponemos de contadores que nos ayuden a determinar el resultado de la comparación antes descrita, como se hacía en los capítulos anteriores. La función del contador la desempeñan los objetos q_i, con $i = 0, 1, 2, 3$, cuya evolución está regida por las reglas de los apartados (f), (g) y (h). Veremos a continuación que estos objetos actúan como marcadores y que se establece un mecanismo fiable (y finito) para determinar si el chequeo tiene éxito o no. Estudiando los distintos casos posibles.

1. Supongamos que $w(B) > w(B^c)$. En este caso, la evolución de la membrana asociada al par (B, B^c) está descrita en la Tabla 1.

Multiconjunto	Carga	
$q_0\, a^{w_B}\, p^{w_{B^c}}$	$-$	$\left.\begin{array}{l}\\\\\\\\\\\\\end{array}\right\}$ $2w_{B^c}$ pasos
$q_1\, a^{w_B-1}\, p^{w_{B^c}}$	0	
$q_0\, a^{w_B-1}\, p^{w_{B^c}-1}$	$-$	
$\vdots$	$\vdots$	
$q_0\, a^{w_B-w_{B^c}}$	$-$	
$q_1\, a^{w_B-w_{B^c}-1}$	0	
$q_0\, a^{w_B-w_{B^c}-1}$	0	
$a^{w_B-w_{B^c}-1}(No_0$ a la piel)	$+$	

Tabla 1

2. Supongamos que $w(B) \leq w(B^c)$. En este caso, la evolución de la membrana asociada al par (B, B^c) está descrita en la Tabla 2.

Multiconjunto	Carga
$q_0\, a^{w_B}\, p^{w_{B^c}}$	$-$
$\vdots$	$\vdots$
$q_0\, p^{w_{B^c}-w_B}$	$-$
$q_1\, p^{w_{B^c}-w_B}$	$-$
$q_2\, c\, p^{w_{B^c}-w_B}$	$-$
$q_2\, p^{w_{B^c}-w_B}$	0
$q_3\, p^{w_{B^c}-w_B-1}$	$-$
$p^{w_{B^c}-w_B-1}$ (No_0 a la piel)	$+$

Caso $w(B) < w(B^c)$

Multiconjunto	Carga
$q_0\, a^{w_B}\, p^{w_{B^c}}$	$-$
$\vdots$	$\vdots$
q_0	$-$
q_1	$-$
$q_2\, c$	$-$
q_2	0
q_3	0
$\emptyset$ (Yes a la piel)	$+$

$\}\ 2w_B$ pasos

Caso $w(B) = w(B^c)$

Tabla 2

Es decir, el marcador q_i está diseñado de manera que sólo enviará un objeto Yes a la piel en el caso en el que las multiplicidades de los objetos a y p coincidan; esto es, en el caso en que $w(B) = w(B^c)$. En otro caso, se envía a la piel un objeto No_0.

Por tanto, hemos probado que la fase de chequeo funciona debidamente.

□

En cuanto a la fase de respuesta, el funcionamiento es similar a los diseños de los problemas Subset Sum y Knapsack. Tenemos un mecanismo que provoca un cambio de carga en la piel tras un determinado número de pasos, de manera que se garantiza que todos los procesos internos han finalizado en el momento en que dicho cambio se produce. En los capítulos anteriores, dicho mecanismo se reducía a un contador z_i en la piel, pero en este caso, el contador z_i está complementado con las reglas del apartado (l), referidas a una nueva membrana auxiliar etiquetada por r, cuyo funcionamiento hemos descrito en la sección anterior. Básicamente, se trata de la combinación de dos procesos:

- Por un lado, se tienen 2^{n-1} objetos g en la piel (tantos como membranas relevantes, pues un objeto g_0 es enviado a la piel cada vez que una membrana llega a ser relevante). Dichos objetos van siendo enviados dentro de las membranas internas a medida que éstas finalizan su fase de chequeo y obtienen carga positiva.

- Por otro lado, la membrana r implementa un "bucle detector" que parará cuan-

do no quede ningún objeto g en la piel. Esto se corresponderá con el hecho de que todas las membranas relevantes hayan terminado su fase de chequeo (ya que cada objeto g entra en una membrana relevante).

Obsérvese que para que las membranas con carga positiva sean las 2^{n-1} membranas relevantes es esencial que se disuelvan las membranas "sobrantes" $[e_n]_e^+$ durante la fase de generación. Esto no sería necesario si modificásemos la fase de generación de manera que no se generen membranas "inútiles".

Después de que se produzca el cambio de carga (en este caso, mediante la regla $[d_2]_s^+ \rightarrow [\,]_s^- d_2$), cualquier objeto Yes presente en la membrana piel puede ser enviado al entorno justo en el paso siguiente (mediante la regla $[Yes]_s^- \rightarrow [\,]_s^0 Yes$). Para obtener una respuesta negativa son necesarios dos pasos (uno de renombramiento $[No_0 \rightarrow No]_s^-$ y otro de comunicación $[No]_s^- \rightarrow [\,]_s^0 No$), luego no existe conflicto a la hora de enviar la respuesta. Se envía al entorno un objeto Yes si y sólo si alguna membrana relevante ha encontrado una solución, y en caso contrario se envía un objeto No.

□

De todo lo expuesto en esta sección, y de acuerdo con la Definición 2.13, se deduce el siguiente resultado:

Teorema 3 *a*) *Partición* $\in \mathbf{PMC}_{\mathcal{AM}}$.

b) $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

Demostración. Para demostrar *b*), basta hacer las siguientes observaciones: el problema de la Partición es **NP**–completo, *Partición* $\in \mathbf{PMC}_{\mathcal{AM}}$ (por el primer apartado) y la clase $\mathbf{PMC}_{\mathcal{AM}}$ es estable bajo reducción polinomial y cerrada bajo complementario.

□

6.5. Mejoras en el diseño

Al estudiar el comportamiento del sistema durante la fase de generación, vimos que hay una serie de membranas que no aportan nada a la computación, pues son disueltas inmediatamente después de ser generadas (las que estaban marcadas con un rombo en la Figura 6.2, con carga positiva y que contenían al objeto e_n). El número total de estas "membranas inútiles" es 2^{n-1}; es decir, tantas como membranas relevantes. En lo que respecta al modelo formal esta circunstancia no nos preocupa, ya que se trata de espacio de trabajo creado durante la computación y, por tanto, no se requieren

recursos *a priori*. Pero si estamos interesados en una simulación del diseño celular en un ordenador, entonces la complejidad computacional en espacio se torna mucho más importante. Incluso si usamos el artificio de disolver las membranas inmediatamente después de que sean generadas, los recursos que requiere el sistema para su simulación son excesivos.

El hecho de que se generen este tipo de membranas es consecuencia del diseño. Se antepuso la simetría de los dos esquemas de reglas de división que componen el apartado (a) a las consideraciones de complejidad en espacio. Ahora, una vez que el mecanismo de la fase de generación ya se ha estudiado en profundidad, vamos a tratar de evitar la generación de "membranas inútiles" modificando las reglas. Una primera idea sería recurrir a una estrategia de división basada en una estructura de árbol binario completo, obteniendo todas las membranas relevantes a la vez tras un número lineal de pasos celulares. En lugar de seguir esta estrategia, trataremos de lograr que algunas membranas relevantes se obtengan antes de que termine la fase de generación. La motivación de esta decisión es que pretendemos conseguir una mayor eficiencia (en tiempo) para los casos mejor y promedio.

A continuación, proponemos una alternativa a las reglas del apartado (a) presentadas en la Sección 6.2. Esta vez sin producir membranas sobrantes, con lo que las reglas de disolución ya no son necesarias. Más aún, el número de cargas eléctricas utilizadas se reduce a dos: la variación con el tiempo de la carga eléctrica de las membranas sigue siendo significativa, pero ahora el final de la fase no lo marca el obtener carga negativa, sino el mantener carga neutra durante dos pasos de evolución consecutivos (véase [2] para un ejemplo de uso de sistemas celulares con membranas activas que usan sólo dos cargas). Esta nueva condición de parada es controlada mediante unos "objetos-testigo" e'_i y e''_i, que indican si la carga en el paso previo fue neutra (e''_i) o positiva (e'_i).

Fase de generación

$[e_i]^0_e \to [q]^0_e[e_i]^+_e$, para $i = 1, \ldots, n-1$,

$[e_i]^+_e \to [e_{i+1}]^0_e[e_{i+1}]^+_e$, para $i = 1, \ldots, n-2$,

$[e_n \to q]^0_e$, $[e_{n-1}]^+_e \to [\ ]^0_e\#$,

$[e'_i \to e'_{i+1}]^+_e$, $[e''_i \to e'_{i+1}]^+_e$, para $i = 1, \ldots, n-2$,

$[e'_i \to e''_i]^0_e$, $[e''_i \to \lambda]^0_e$, para $i = 1, \ldots, n-1$,

$[e'_{n-1} \to e_n]^+_e$, $[e''_{n-1} \to e_n]^+_e$.

Si quisiéramos insertar estas reglas en el diseño como nuevo apartado (a), entonces

habríamos de adaptar también la fase de cálculo (ya que dependía de los cambios de carga causados por las reglas de división). Una posible adaptación sería la siguiente:

Fase de cálculo de pesos

$[x_1 \to a_0]_e^0$; $\quad [x_1 \to p_0]_e^+$

$[x'_1 \to a_0]_e^0$; $\quad [x'_1 \to p_0]_e^+$

$[x_i \to x_{i-1}]_e^+$, $\qquad$ para $i = 2, \ldots, n$

$[x'_i \to x_{i-1}]_e^+$, $\qquad$ para $i = 2, \ldots, n$

$[x_i \to x'_i]_e^0$, $\qquad$ para $i = 2, \ldots, n$

$[x'_i \to p_0]_e^0$, $\qquad$ para $i = 2, \ldots, n$

En este caso no se trata de un cambio de estrategia, sino que el cambio consiste en introducir "objetos-testigo" que recuerden la carga del paso anterior. Esto permite detectar cuándo termina la fase de generación y así podemos finalizar el cálculo de pesos en el mismo instante.

Capítulo 7

Hacia un lenguaje de programación celular

Hemos presentado en esta memoria varias soluciones a problemas **NP**–completos mediante familias de sistemas celulares reconocedores, y se han hecho patentes, al desarrollar y estudiar los diseños de las familias correspondientes a los distintos problemas, fuertes similitudes entre dichos diseños. En base a esto, trataremos en este capítulo de mostrar que la idea de un *lenguaje de programación celular* es factible (al menos para cierta familia relevante de problemas **NP**-completos), explicitando algunas "subrutinas" que pueden ser usadas para ciertos propósitos en situaciones diversas y, por tanto, podrían ser de utilidad a la hora de diseñar soluciones para nuevos problemas en el futuro.

Resaltaremos las similitudes entre los diseños de familias de sistemas celulares presentados en los Capítulos 4, 5 y 6, y extraeremos las oportunas conclusiones encaminadas a agrupar conjuntos de reglas que permitan ejecutar, globalmente, determinadas tareas computacionales. Asimismo se mencionarán las soluciones para los problemas SAT y VALIDITY (presentadas en [42] y [46]) con objeto de acomodar también a esos casos el proyecto de lenguaje de programación celular, describiendo dichas soluciones a modo de *programas*.

El capítulo está organizado como sigue. En primer lugar describimos algunas ideas genéricas que justifican la búsqueda de procesos, en cierto sentido invariantes, que aparecen en diseños de sistemas celulares. En la Sección 7.2 se estudian dichos procesos de manera sistemática a partir de una solución del problema Partición. Por último, en la Sección 7.3 se aplican las subrutinas obtenidas para describir de forma alternativa las soluciones presentadas de otros problemas **NP**–completos.

7.1. Introducción

En los Capítulos 4, 5 y 6 se han presentado los diseños de familias de sistemas P con membranas activas que resuelven problemas **NP**–completos en tiempo (de computación celular) lineal. Es claro que el factor determinante para lograr esta aceleración está en la posibilidad de duplicar en un paso el espacio de trabajo, mediante la división de membranas, permitiendo la construcción de un espacio de trabajo de tamaño exponencial en tiempo lineal. Pero hay otra singularidad del modelo con membranas activas que ha desempeñado un papel esencial en los diseños: la asignación de cargas eléctricas a las membranas (las reglas no se asocian a una etiqueta de membrana, sino a un par etiqueta–carga) y la posibilidad de cambiar ésta mediante objetos que atraviesen la membrana (para cada membrana, sólo un objeto puede atravesarla en cada paso de transición).

Sin embargo, no nos interesa destacar el uso de cargas eléctricas *per se*, sino más bien las funciones que pueden desempeñar durante la computación. Es posible implementar, de manera implícita, una relación de prioridad entre reglas, "inhibiendo" algunas de ellas en función de la carga que requieran para ser aplicables. También podemos sincronizar de alguna manera varios procesos, incluyendo reglas de renombramiento que sólo se aplican con una carga determinada (así, los procesos que queramos sincronizar no pueden empezar hasta que no aparezcan, tras el renombramiento, los objetos a los que afectan las reglas). Como último ejemplo, recordemos la estrategia de diseñar bucles en los que en cada paso cambian tanto la carga de la membrana como el índice de un marcador o contador situado en el interior de dicha membrana, de manera que sea posible detectar el final del bucle a partir de la información que aportan dicho índice y la carga de la membrana.

Estas funciones que acabamos de citar no son exclusivas del modelo de membranas activas con cargas eléctricas. También es posible utilizar estrategias similares en otras variantes de modelos de computación celular (modelos con prioridad entre reglas, modelos donde se permiten reglas con cooperación, etc.). En este capítulo se tratará de agrupar las reglas de los sistemas presentados en los Capítulos 4, 5 y 6 en función de la tarea específica que desempeñan durante la computación. El objetivo es iniciar el camino hacia un lenguaje de programación celular, indicando una serie de instrucciones que puedan ser usadas para el diseño de *algoritmos* en el marco de los sistemas celulares. En este capítulo interpretaremos las instrucciones de este lenguaje según la función que desempeñen, y presentaremos grupos de reglas (para nuestra variante de sistemas celulares con membranas activas) que se correspondan con dicha función. Es decir, las instrucciones que vamos a considerar en este capítulo vienen a ser una especie de *macros* en los sistemas celulares.

7.2. Un programa celular para el problema Partición

Recordemos que un problema de decisión, X, es un par (I_X, θ_X), donde I_X es un lenguaje sobre un alfabeto finito (cuyos elementos denominaremos *instancias*) y θ_X es una función total booleana sobre I_X. Es decir, la respuesta a cada instancia del problema será o bien VERDADERO o bien FALSO. Por eso, estamos interesados en que los dispositivos de computación que usemos sean capaces de recibir una entrada, procesarla, y devolver una respuesta de tipo booleana.

Los dispositivos base con los que se trabaja en esta memoria son la clase de sistemas celulares con entrada y con salida externa (además, se consideran objetos especiales Yes y No en el alfabeto de trabajo de los sistemas para poder implementar la respuesta booleana). Por otra parte, con objeto de lograr una aceleración significativa en las computaciones, se ha elegido la variante de los sistemas celulares con membranas activas. De este modo, es posible obtener un espacio de trabajo de orden exponencial en tiempo polinomial, mediante la división de membranas elementales. También se han considerado otras condiciones o restricciones adicionales, con objeto de lograr la adecuación de los dispositivos, y de facilitar la verificación formal de los diseños: los sistemas han de ser *confluentes* (todas las computaciones con la *misma* entrada deben conducir a la *misma* respuesta) y todas las computaciones deben ser finitas; más aún, el sistema producirá una respuesta exactamente en el último paso de la computación, enviando al entorno un objeto especial Yes o No. De esta manera, un usuario situado fuera del sistema puede saber, exactamente, el momento en que la computación finaliza y, además, conocer la respuesta. Estas condiciones, junto con otras consideraciones, fueron introducidas en el Capítulo 2, estableciendo un marco de *clases de complejidad* en sistemas celulares. En este marco, se ha demostrado, en los Capítulos 4, 5 y 6, que existen soluciones celulares eficientes (que se ejecutan en tiempo lineal) para problemas **NP**-completos.

En este capítulo dejaremos un poco de lado la búsqueda de la eficiencia, para centrarnos en tratar que los diseños sean tan versátiles como sea posible. Como hilo argumental para esta sección, retomaremos el problema de la Partición, presentando una solución ligeramente distinta a la que se vio en el capítulo anterior y comentando su funcionamiento y sus posibles *adaptaciones* a otros problemas.

Consideremos la siguiente familia de sistemas celulares:

$$\mathbf{\Pi} = \{(\Pi(n), \Sigma(n), i(n)) : n \in \mathbb{N}\}$$

Donde, para cada elemento de la familia, el alfabeto de entrada es $\Sigma(n) = \{x_1, \ldots, x_n\}$, la etiqueta de la membrana de entrada es $i(n) = e$, y el sistema $\Pi(n)$ viene dado por la tupla $(\Gamma(n), \{e, r, s\}, \mu, \mathcal{M}_e, \mathcal{M}_r, \mathcal{M}_s, R)$, que describimos a conti-

nuación:

- Alfabeto: $\Gamma(n) = \Sigma(n,k) \cup \{a_0, a, b_0, b, c, d_0, d_1, d_2, e_1, \ldots, e_n, g, g_0, g_1, h_0, h_1, p, p_0, q, q_0, q_1, q_2, q_3, Yes, No, No_0, z_1, \ldots, z_{2n+1}, \#\}$
- Estructura de membranas: $\mu = [\ [\ ]_e^0\ [\ ]_r^0\ ]_s^0$.
- Multiconjuntos iniciales: $\mathcal{M}_e = e_0\, g_1$; $\mathcal{M}_r = b_0\, h_0$ y $\mathcal{M}_s = z_1$
- El conjunto de reglas de evolución, R, consta de las siguientes reglas:

  (a) $[e_i]_e^0 \to [q]_e^- [e_i]_e^+$, para $i = 0, \ldots, n$.
  $[e_i]_e^+ \to [e_{i+1}]_e^0 [e_{i+1}]_e^+$, para $i = 0, \ldots, n-1$.

  (b) $[x_0 \to a_0]_e^0$; $[x_0 \to p_0]_e^+$.
  $[x_i \to x_{i-1}]_e^+$, para $i = 1, \ldots, n$.
  $[x_i \to p_0]_e^-$, para $i = 1, \ldots, n$.

  (c) $[q \to q_0]_e^-$; $[p_0 \to p]_e^-$; $[a_0 \to a]_e^-$.

  (d) $[a]_e^- \to [\ ]_e^0 \#$; $[p]_e^0 \to [\ ]_e^- \#$.

  (e) $[q_0 \to q_1]_e^-$; $[q_1 \to q_0]_e^0$.

  (f) $[q_0]_e^0 \to [\ ]_e^+ No_0$.

  (g) $[q_1 \to q_2 c]_e^-$; $[c]_e^- \to [\ ]_e^0 \#$; $[q_2 \to q_3]_e^0$.
  $[q_3]_e^0 \to [\ ]_e^+ Yes$; $[q_3]_e^- \to [\ ]_e^+ No_0$.

  (h) $[p \to \#]_e^+$; $[a \to \#]_e^+$.

  (i) $[e_n]_e^+ \to \#$; $[a_0 \to \#]_s^0$; $[p_0 \to \#]_s^0$; $[g_1 \to \#]_s^0$.

  (j) $[z_i \to z_{i+1}]_s^0$, para $i = 1, \ldots, 2n$.
  $[z_{2n+1} \to d_0 d_1]_s^0$; $[d_1]_s^0 \to [\ ]_s^+ d_1$.

  (k) $[g_1]_e^- \to [\ ]_e^- g_0$; $[g_0 \to g]_s^+$; $g[\ ]_e^+ \to [g]_e^0$.

  (l) $d_0[\ ]_r^0 \to [d_0]_r^-$.
  $[h_0 \to h_1]_r^-$; $[h_1 \to h_0]_r^+$; $[h_1 \to h_0]_r^0$.
  $[b]_r^- \to [\ ]_r^+ b$; $g[\ ]_r^+ \to [g]_r^-$; $b[\ ]_r^- \to [b]_r^0$; $[g]_r^0 \to [\ ]_r^- g$.
  $[h_0]_r^+ \to [\ ]_r^+ d_2$; $[d_2]_s^+ \to [\ ]_s^0 d_2$.

  (m) $[No_0 \to No]_s^-$; $[Yes]_s^- \to [\ ]_s^0 Yes$; $[No]_s^- \to [\ ]_s^0 No$.

A continuación vamos a analizar las primeras *instrucciones* que se podrían añadir a la librería de subrutinas que se pretende crear, a modo de comandos de un lenguaje de programación celular. Para ello, se comentarán las distintas fases de la computación del sistema antes diseñado, separando los distintos procesos que se llevan a cabo en el sistema (aclararemos qué objetos y qué reglas intervienen en cada proceso).

Para estructurar esta sección, seguiremos el esquema general para resolución de problemas **NP**–completos mediante sistemas celulares reconocedores con membranas activas, mostrado en la Figura 3.1. Recordémoslo brevemente: en una primera fase, se *genera* una cantidad exponencial de membranas, mediante la división sucesiva de algunas de ellas, de manera que cada membrana obtenida codifique una posible solución; en segundo lugar, cada membrana (en paralelo) ejecuta una *fase de chequeo* para comprobar si la solución codificada en su interior es afirmativa; por último, se requiere algún procedimiento para *interpretar* los *resultados* de los chequeos de todas las membranas y enviar un mensaje apropiado (la *respuesta*) al entorno externo.

Generación de la biblioteca

En el primer paso de computación se aplica la regla $[e_i]_e^0 \rightarrow [q]_e^-[e_i]_e^+$, para $i = 0$. A partir de este instante, las reglas de división de (a) se irán aplicando sucesivamente sobre las membranas del sistema, en función de la carga que tengan y del índice del objeto e_i que pertenezca a ellas. Cuando una membrana obtiene carga negativa, dicha membrana ya no se dividirá más en lo que resta de computación (la fase de generación termina en esa membrana).

Como hemos visto en los capítulos anteriores, las reglas del apartado (a) que acabamos de mostrar aparecen repetidas en los diseños de las soluciones a los problemas Subset Sum y Knapsack. Por tanto, podríamos definir una nueva *instrucción*, que fuese utilizable en el diseño de sistemas celulares, y que podríamos llamar, por ejemplo *gen_subsets*(n). Su misión sería generar 2^n membranas y podría definirse como sigue:

$$gen\_subsets(n) \equiv \left(\begin{array}{ll} [e_i]_e^0 \rightarrow [q]_e^-[e_i]_e^+, & \text{para } i = 0, \ldots, n \\ [e_i]_e^+ \rightarrow [e_{i+1}]_e^0[e_{i+1}]_e^+, & \text{para } i = 0, \ldots, n-1 \end{array} \right)$$

Esto no es más que una nueva notación; es decir, cada vez que aparezca esta instrucción en un diseño, se entiende que está representando al conjunto de reglas de (a). Obsérvese que se ha optado por una expresión simplificada de la instrucción, sin hacer referencia explícita a los *nombres* de los objetos que aparecen en las reglas que la definen.

En cuanto al concepto de *subconjunto asociado*, que es aludido muchas veces en los capítulos anteriores, nótese que es tan sólo una abstracción, pues no hay marcadores u objetos-testigo en la membrana que atestigüen qué elementos pertenecen al subconjunto asociado de la membrana. Por tanto, se puede hacer uso, si se desea, de la noción semántica de subconjunto asociado en fases futuras de la computación.

Por ejemplo, esto ocurre así en la fase de cálculo. Las reglas del apartado (b) están diseñadas para trabajar en coordinación con las de (a): los pesos de los elementos son transformados en objetos *subs* o en objetos *compl* (en nuestro caso a_0 y p_0, respectivamente) dependiendo de si el elemento correspondiente pertenece o no al subconjunto asociado. Con estas ideas, podemos definir una instrucción $calc\_weight(n)$ cuyo objetivo sea hallar el peso de los subconjuntos representados por las membranas. Dicha instrucción podría definirse como sigue:

$$calc\_weight(n) \equiv \left(\begin{array}{ll} [x_0 \rightarrow subs]_e^0, & \\ [x_0 \rightarrow compl]_e^+, & \\ [x_i \rightarrow x_{i-1}]_e^+, & \text{para } \; i = 1, \ldots, n \\ [x_i \rightarrow compl]_e^-, & \text{para } \; i = 1, \ldots, n \end{array} \right)$$

La multiplicidad del objeto *subs* codifica el peso del subconjunto asociado, entendiendo por "peso" cualquier función aditiva que estemos considerando en el problema. Por ejemplo, en el caso del problema Knapsack, hay dos funciones que deben ser computadas: el peso y el valor del subconjunto. Así pues, hay que incluir dos copias del esquema de reglas $calc\_weight(n)$, una para cada función. Para evitar interferencias, se renombran los objetos que intervienen en las reglas (los objetos x_i pasan a ser y_i, el objeto *subs* es en el primer caso a_0 y en el segundo b_0). Por otra parte, el objeto *compl* puede ser sustituido por λ si no nos interesa la información acerca del complementario.

Las fases de generación y cálculo (reglas de $gen\_subsets(n)$ y $calc\_weight(n)$) terminan cuando la membrana recibe por primera vez carga eléctrica negativa. En ese mismo instante, aparece un objeto q en la membrana, que puede ser usado si fuese necesario para tareas de coordinación o bien, como es nuestro caso, para generar objetos especiales que entren en juego en fases posteriores.

Para evitar posibles interacciones no deseadas entre reglas de distintas fases, es útil evitar situaciones en las que existan elecciones no-deterministas incontroladas (sobre todo es importante mantener un cierto control sobre las reglas que afectan a las cargas). Para ello, una posible solución es efectuar renombramientos de todos los objetos presentes en la membrana entre una fase y la siguiente. En nuestro caso, la comparación de las multiplicidades de los objetos *subs* y *compl* se realiza enviándolos

fuera de la membrana uno a uno alternativamente, cambiando la carga en cada paso. Evitamos que las reglas del chequeo interfieran en las fases de generación y cálculo mediante el renombramiento que indican las reglas de (c).

Como se puede observar en los capítulos anteriores, el renombramiento puede variar en función del número de fases que queden aún por ejecutar (en el caso del problema Knapsack hay que renombrar más objetos, porque hay una fase más de chequeo). Además, en el caso del problema de la Partición, se envía desde cada membrana a la piel un objeto g_0, que entrará en juego en la fase de respuesta.

Ahora bien, recuérdese que en el sistema también se generan membranas etiquetadas por e durante la fase de generación pero que no aportan nada a la computación (las membranas donde aparece un objeto e_n y tienen carga positiva). Dichas membranas podrían interferir en la fase de respuesta y, por tanto, deben ser disueltas. Para ello se incluye en el diseño una instrucción *clean_dissolve* que se ocupa de disolver dichas membranas sobrantes y de "limpiar" los objetos que son liberados en la piel como resultado de las disoluciones.

$$clean\_dissolve(n) \equiv \left(\begin{array}{l} [e_n]_e^+ \rightarrow \# \\ [a_0 \rightarrow \#]_s^0; \quad [p_0 \rightarrow \#]_s^0; \quad [g_1 \rightarrow \#]_s^0 \end{array} \right)$$

Comprobación / chequeo

Pasemos al siguiente apartado, las dos reglas de (d). Debido a la aplicación alternativa de estas reglas, la carga de la membrana va cambiando en cada paso, de manera que tras dos pasos de evolución la situación es casi idéntica a la inicial: se tiene la misma carga, pero hay una pareja menos de objetos. El bucle comparador continúa funcionando hasta que se agota uno de los objetos que intervienen en él (en nuestro caso, los objetos son a_0 y a). Esto da pie a considerar una nueva instrucción, *check_weight*, como sigue:

$$check\_weight \equiv \left(\begin{array}{l} [obj1]_e^- \rightarrow [\ ]_e^0 \# \\ [obj2]_e^0 \rightarrow [\ ]_e^- \# \end{array} \right)$$

En este caso la instrucción *check_weight* representa un número constante de reglas, independiente de n. Lo importante en este caso es tener dos objetos distintos, $obj1$ y $obj2$, y trabajar con dos cargas distintas, no tienen porqué ser, precisamente, neutra y negativa. De hecho, para el problema Knapsack se usa un bucle con cargas negativa y neutra, para el peso, y otro bucle con cargas positiva y neutra, para el valor. Lo que caracteriza la instrucción *check_weight* es la estrategia de bucle con rotación de dos cargas eléctricas.

A continuación, vamos a comentar las reglas que se ocupan del *resultado del chequeo*. Como ya se ha comentado, en este caso no utilizamos un contador que incremente su índice en cada paso. En su lugar disponemos de dos objetos que actúan como marcadores, y esto es suficiente para detectar cuándo para el bucle de chequeo y para decidir el resultado del mismo. Las reglas que gobiernan la evolución de este contador son las del apartado (e):

$$[i_{same} \to i_{diff}]_e^-; \quad [i_{diff} \to i_{same}]_e^0$$

Veamos cómo se comporta el sistema según los distintos resultados del chequeo. Consideremos que en un sistema celular aparecen la instrucción *check_weight* junto con las dos reglas anteriores para los objetos i_{same} e i_{diff}. Vamos a generalizar las reglas de los apartados (f) y (g), explicando su funcionamiento.

1. En primer lugar, supongamos que la multiplicidad del objeto $obj1$ es mayor que la del objeto $obj2$. Entonces, cuando se hayan agotado estos últimos, habrá un paso en el que la regla $[obj1]_e^- \to [\,]_e^0\#$ será aplicada, pero el bucle se detendrá en el siguiente paso, porque la regla $[obj2]_e^0 \to [\,]_e^-\#$ no será aplicable, con lo que la membrana tendrá la misma polaridad (neutra) durante dos pasos consecutivos. Esto hace que el marcador i_{same} finalice esta fase, mediante la regla

   $$[i_{same}]_e^0 \to [\,]_e^+ z_{more}$$

   En el caso del problema de la Partición, el hecho de que haya más copias del objeto $obj1$ que del objeto $obj2$ significa que el chequeo tiene un resultado negativo, por tanto tendríamos que reemplazar el objeto z_{more} por un objeto No_0, pero para otros problemas se podría reemplazar por Yes o por cualquier otro objeto especial que sirva para activar posteriores etapas.

2. Supongamos que, por el contrario, la multiplicidad del objeto $obj1$ es menor o igual que la del objeto $obj2$. En este caso las dos reglas de *check_weight* se aplicarán el mismo número de veces, hasta el momento en que se hayan agotado los objetos $obj1$. En ese instante la regla $[obj1]_e^- \to [\,]_e^0\#$ no será aplicable, con lo que la membrana tendrá la misma polaridad (negativa) durante dos pasos consecutivos. Ahora bien, tenemos que diferenciar los casos en los que aún haya objetos $obj2$ en la membrana de los casos en los que no quede ninguno. Para ello, se cambia la polaridad de la membrana, pasando a carga neutra, mediante las reglas

   $$[i_{diff} \to aux_1 c]_e^-; \quad [c]_e^- \to [\,]_e^0\#$$

   En el siguiente paso, el marcador auxiliar aux_1 evoluciona a aux_2 y, además, la carga de la membrana será modificada si y sólo si quedaban objetos $obj2$ en ella

(a través de la regla $[obj2]_e^0 \rightarrow [\ ]_e^- \#$). Consecuentemente, podemos decidir cuál es el resultado del chequeo según la carga de la membrana en el paso siguiente, mediante las reglas:

$$[aux_2]_e^0 \rightarrow [\ ]_e^+ z_{equal}; \quad [aux_2]_e^- \rightarrow [\ ]_e^+ z_{less}$$

En el caso del problema de la Partición, el chequeo tiene resultado positivo si había tantos objetos $obj1$ como objetos $obj2$. Por eso, en el diseño presentado anteriormente se ha sustituido z_{equal} por Yes, y z_{less} por No_0.

Además, para evitar que las reglas de la instrucción *check_weight* se puedan aplicar después del final de la fase de chequeo (en el caso en que las multiplicidades de $obj1$ y $obj2$ no coincidan), en el diseño se incluyen las reglas de borrado $[obj1 \rightarrow \#]_e^+$ y $[obj2 \rightarrow \#]_e^+$.

En consecuencia, podemos considerar una instrucción *marker_eq* encargada de controlar el resultado del chequeo. Su definición podría ser la siguiente:

$$marker\_eq \equiv \left(\begin{array}{l} [i_{same} \rightarrow i_{diff}]_e^-; \quad [i_{diff} \rightarrow i_{same}]_e^0 \\ [i_{same}]_e^0 \rightarrow [\ ]_e^+ No_0 \\ [i_{diff} \rightarrow aux_1 c]_e^-; \quad [c]_e^- \rightarrow [\ ]_e^0 \# \\ [aux_1 \rightarrow aux_2]_e^0 \\ [aux_2]_e^0 \rightarrow [\ ]_e^+ Yes; \quad [aux_2]_e^- \rightarrow [\ ]_e^+ No_0 \\ [obj1 \rightarrow \#]_e^+; \quad [obj2 \rightarrow \#]_e^+ \end{array} \right)$$

De manera natural se pueden definir variantes de la instrucción anterior, *marker_leq* y *marker_geq*, en las que se considere que el chequeo tiene un resultado exitoso si se detecta que el número de copias de $obj1$ es menor (o mayor, respectivamente) que el número de copias de $obj2$. También es posible que en algunos problemas nos interese que en la membrana donde se realiza el chequeo con resultado no exitoso, en lugar de enviar fuera un objeto No_0, dicha membrana se bloquee. Esto se puede conseguir simplemente eliminando las correspondientes reglas $[i_j]_e^{\alpha_1} \rightarrow [\ ]_e^{\alpha_2} No_0$.

Contador e Intérprete

Por último, pasemos a estudiar cómo funciona el proceso de *sincronización de la respuesta*. Se trata de conseguir que el sistema no envíe ningún objeto especial (Yes o No) al entorno antes de que todos los procesos internos hayan terminado. Como ya se apuntó en el capítulo anterior, la membrana etiquetada por r desempeña un papel protagonista en esta fase. Expliquemos el proceso paso a paso (recuérdese la Figura 6.3 para una descripción gráfica del bucle de detección).

Los procesos internos del sistema finalizarán cuando todas las membranas relevantes hayan completado su fase de chequeo. Dado que hay una cantidad exponencial de membranas relevantes en el sistema, una buena manera de controlarlas es utilizar una cantidad exponencial de objetos auxiliares, aprovechando el paralelismo del modelo a nivel de membranas (todas evolucionan en paralelo y pueden interaccionar de forma independiente con dichos objetos auxiliares).

Ahora bien, para generar tantos objetos como membranas relevantes se puede recurrir a un método muy simple: se incluye en la configuración inicial un objeto g dentro de la membrana e, que será duplicado cada vez que se produzca una división de membranas, y se incluye una regla $[g]_e^- \rightarrow [\ ]_e^- g_0$. Entonces, cada vez que una membrana llegue a ser relevante, aparecerá un nuevo objeto g_0 en la piel. De esta manera conseguimos una cantidad exponencial de objetos g_0 (concretamente, la cantidad coincide con el número de membranas relevantes).

La idea es que estos objetos vuelvan a entrar en las membranas etiquetadas por e a medida que éstas vayan terminando su fase de chequeo, y así el hecho de que finalicen los procesos internos se corresponderá con el hecho de que no queden objetos auxiliares en la piel. Nótese que, según las reglas de los apartados (f) y (g), las membranas relevantes siempre reciben carga positiva al finalizar su fase de chequeo, tanto si el resultado es afirmativo como negativo. Por tanto, podríamos incluir una regla $g_0\,[\ ]_e^+ \rightarrow [g_0]_e^0$ para que los objetos g_0 regresen a sus membranas de origen cuando éstas terminen su chequeo.

Sin embargo, pueden surgir conflictos con membranas etiquetadas por e que se hallen aún en la fase de generación y tengan carga positiva en un momento en el que otra membrana ya haya enviado un objeto g_0 a la piel. Para evitar estas interferencias, se añade un contador en la piel, z_i, junto con una regla de renombramiento, $[g_0 \rightarrow g]_s^+$, de manera que los objetos involucrados en el "bucle detector" no son copias de g_0 sino de g, y el proceso de detectar si todas las membranas han acabado sus chequeos no comienza hasta que un objeto d_1 (que evoluciona a partir del contador z_i) es enviado al entorno externo, cambiando la polaridad de la piel a positiva. Esto nos lleva a introducir dos nuevas instrucciones, *counter*(n) y *rename* que se definen como sigue:

$$counter(n) \equiv \left(\begin{array}{l} [z_i \rightarrow z_{i+1}]_s^0, \text{ para } i = 1, \ldots, 2n \\ [z_{2n+1} \rightarrow d_0 d_1]_s^0 \end{array} \right)$$

$$rename \equiv \left(\ [d_1]_s^0 \rightarrow [\ ]_s^+ d_1; \quad [g_0 \rightarrow g]_s^+ \ \right)$$

A efectos de sincronización, el contador z_i genera, conjuntamente al indicador d_1, otro indicador d_0 cuya función es activar el bucle descrito por las reglas de (l)

cambiando la carga de la membrana r (mediante la regla $d_0[\ ]_r^0 \rightarrow [d_0]_r^-$). Informalmente, el mecanismo determinado por las reglas de (l) es un bucle que intenta *pescar* cualquier objeto g que se encuentre en la región de la piel. El objeto b sirve como *cebo*, porque cambia la carga de la membrana r a positiva, haciendo que la regla $g[\ ]_r^+ \rightarrow [g]_r^-$ sea aplicable (es claro que, si no quedan objetos g en la piel, entonces esta regla no se puede aplicar). Hay un marcador, h_i, en el interior de la membrana r que controla si algún objeto g ha atravesado efectivamente la membrana (cambiando la carga a negativa) y en caso negativo se expulsa un objeto d_2 a la piel para pasar a la fase de respuesta propiamente dicha.

Esto justifica que consideremos una nueva instrucción, que denominaremos *detector*, y que puede definirse como sigue:

$$detector \equiv \left(\begin{array}{ll} d_0[\ ]_r^0 \rightarrow [d_0]_r^- & \\ [h_{neg} \rightarrow h_{pos}]_r^-; & [b]_r^- \rightarrow [\ ]_r^+ b \\ [h_{pos} \rightarrow h_{neg}]_r^+; & g[\ ]_r^+ \rightarrow [g]_r^- \\ b[\ ]_r^- \rightarrow [b]_r^0 & \\ [h_{pos} \rightarrow h_{neg}]_r^0; & [g]_r^0 \rightarrow [\ ]_r^- g \\ [h_{neg}]_r^+ \rightarrow [\ ]_r^+ d_2 & \\ [d_2]_s^+ \rightarrow [\ ]_s^- d_2 & \end{array} \right)$$

Respuesta

El último paso a realizar consiste en expulsar la respuesta apropiada al exterior. Si la respuesta a la instancia es afirmativa, un objeto Yes deberá ser enviado al entorno y, en caso contrario, se deberá expulsar un objeto No. El método elegido ha sido idéntico en los Capítulos 4, 5 y 6: se trata de introducir un paso extra para el objeto No (teniendo presente que antes de ser enviado al entorno debe aplicarse la regla de renombramiento $[No_0 \rightarrow No]_s^-$), de manera que si se recibe en el entorno un objeto No estemos seguros de que no hay ningún objeto Yes en la piel (ya que si lo hubiese, habría salido del sistema en el paso anterior). En cualquier caso, al recibir en el exterior un objeto Yes o un objeto No, sabremos que el sistema ha parado (la computación ha terminado) y que el objeto recibido responde correctamente a la instancia del problema que estábamos considerando.

Por ello, consideramos una instrucción *answer* que se define como sigue:

$$answer \equiv \left(\begin{array}{l} [No_0 \rightarrow No]_s^- \\ [Yes]_s^- \rightarrow [\ ]_s^0 Yes \\ [No]_s^- \rightarrow [\ ]_s^0 No \end{array} \right)$$

De todo lo anterior se deduce que la familia de sistemas reconocedores presentada al inicio de la sección (que proporciona una solución del problema de la Partición) puede ser descrita mediante el siguiente *programa celular*:

```
procedimiento PARTICION
   gen_subsets (n)
   calc_weight (n)
   rename
   check_weight
   marker_eq
   counter (n)
   clean_dissolve
   detector
   answer
```

7.3. Aplicaciones

Se ha señalado en varias ocasiones que en el caso del problema Partición no es posible incluir un contador para controlar el fin de todas las fases de chequeo sin perder uniformidad en el diseño. Sin embargo, sí es posible el proceso inverso: se puede utilizar la instrucción *marker_leq* para detectar el final de las fases de chequeo en los diseños correspondientes a los problemas Subset Sum y Knapsack, en lugar de la estrategia basada en un contador q_i, y en comprobar si el índice del contador es par o impar al final del chequeo. Asimismo, la fase de respuesta utilizada en los Capítulos 4 y 5, basada en un contador z_i cuyo rango de índices dependía de n y k (para el Subset Sum) o de n, k y c (para el Knapsack), puede ser sustituida por un esquema basado en las reglas del bucle *detector* y en un contador cuyo rango de índices sólo depende de n.

A continuación, como ejemplo de la utilidad de las instrucciones/subrutinas esbozadas en la sección anterior, vamos a ver cómo quedaría el diseño de las soluciones para los problemas Subset Sum y Knapsack reescritos como *programas celulares* (no se corresponden exactamente con las presentadas en los Capítulos 4 y 5, ya que todo el diseño depende únicamente de n, y los valores de k y c se introducen como parte del multiconjunto de entrada). Además, presentamos un conjunto de instrucciones para el problema Bin Packing utilizando estas nuevas estrategias, a partir de la solución presentada en [41].

```
procedimiento SUBSET SUM
  gen_subsets (n)
  calc_weight (n)
  rename
  check_weight
  marker_eq
  counter (n)
  clean_dissolve
  detector
  answer
```

```
procedimiento KNAPSACK
  gen_subsets (n)
  calc_weight1 (n)
  calc_weight2 (n)
  rename
  check_weight1
  marker_leq
  rename
  check_weight2
  marker_geq
  counter (n)
  clean_dissolve
  detector
  answer
```

```
procedimiento BINPACKING
  for i = 1, ..., b − 1 do
    gen_subsets (n_i)
    calc_weight (n_i)
    rename
    check_weight
    marker_leq
    counter(n)
    clean_dissolve
  end for
  calc_weight (n_b)
  rename
  check_weight
  marker_leq
  counter (n)
  clean_dissolve
  detector
  answer
```

También se puede utilizar este meta-lenguaje para reescribir soluciones celulares a los problemas SAT y VALIDITY (véase [44] para una descripción exhaustiva de las reglas que componen dichas soluciones, junto con las similitudes entre ambos diseños). Recordemos que se trata, dada una fórmula en forma normal conjuntiva, de decidir si la fórmula es satisfactible (SAT) o de decidir si la fórmula es una tautología (VALIDITY).

```
procedimiento SAT/VALIDITY
   gen_assignments (n)
   calc_satisfied_clauses (n,m)
   synchronization
   check_truth_value (n,m)
   counter (n,m)
   answer
```

No entraremos aquí en detalles acerca de las reglas que corresponden a cada instrucción del *algoritmo* que acabamos de describir, pero comentaremos brevemente la función que desempeña cada una.

Al inicio de la computación se lleva a cabo un proceso de división de membranas, con objeto de generar una membrana para cada una de las posibles asignaciones de verdad sobre las n variables de la fórmula considerada. Aquí nos encontramos que, al igual que ocurre en las fases de generación para los problemas numéricos estudiados, la

carga eléctrica de las membranas durante la división es muy relevante en el proceso, pues determina si a la variable considerada en ese momento se le asigna el valor VERDADERO o bien el valor FALSO.

Hemos elegido denominar *gen_assignments* a la instrucción que se ocupa de la fase de generación, en lugar de *gen_subsets*, porque existen algunas diferencias en el diseño. Por una parte, en [44] se utiliza un esquema de divisiones en forma de árbol binario completo, con lo que todas las membranas relevantes se obtienen en el mismo paso, al finalizar la fase de generación.

Por otra parte, en paralelo al proceso de división, dentro de cada membrana van apareciendo objetos que codifican cuáles de las m cláusulas de la fórmula se satisfacen cada vez que asignamos un valor de verdad a una nueva variable. Esto se puede interpretar en cierta forma como una fase de cálculo: si la cláusula i resulta ser VERDADERA por el valor de verdad asignado a la variable j, entonces se añade en la membrana un objeto "testigo" para esa cláusula; en caso contrario, se pasa a la siguiente variable. La instrucción que ejecuta esta fase es $calc\_satisfied\_clauses$, y se utiliza la técnica de rotación de índices, al igual que sucedía en la instrucción *calc_weights*.

Sin embargo, existen algunas diferencias, ya que se consideran contadores con *dos* subíndices, y esto hace que se requiera una fase de sincronización para homogeneizar los subíndices en todas las membranas antes de pasar a la fase de chequeo. De esto se encarga la instrucción *synchronization*.

En cuanto a la fase de chequeo, la situación es distinta de la que teníamos con los problemas numéricos, porque ahora se trata de comprobar que todas las cláusulas se satisfacen, en lugar de tener que comparar las multiplicidades de dos objetos. De esta fase se encarga la instrucción $check\_truth\_value(n, m)$.

El método utilizado para controlar cuándo termina la fase de chequeo, implementado a través de la instrucción *counter*, utiliza un contador similar al usado para los problemas Subset Sum y Knapsack en los Capítulos 4 y 5.

También se repite el sistema elegido para enviar la respuesta adecuada al entorno: se le otorga cierta prioridad al objeto Yes sobre el objeto No usando la carga eléctrica de la membrana piel. De este proceso se encarga la instrucción *answer*.

Con lo presentado hasta ahora en este capítulo se puede observar que es posible extraer de algunos diseños de sistemas celulares ciertos grupos de reglas cuya ejecución, de acuerdo con la semántica de dichos sistemas, produce una determinada acción, que puede ser considerada como una especie de instrucción de un lenguaje de programación. El ensamblaje de las distintas instrucciones deberá realizarse con mucha precaución, a fin de evitar interferencias no deseadas entre las reglas (recuérdese que

los sistemas P no son secuenciales y, por tanto, en principio cualquier regla puede ser aplicable desde el primer momento). Lo que sí parece claro es que estas instrucciones, a modo de subrutinas, pueden ser de mucha utilidad a la hora de diseñar sistemas celulares que, como bien es sabido, es una tarea extremadamente compleja en los modelos de computación orientados a máquinas, como son los modelos con los que se trabaja en esta memoria.

Capítulo 8

Simulador Prolog de sistemas P con membranas activas

En este capítulo presentamos un simulador de sistemas celulares, escrito en Prolog, y con él se realizarán simulaciones de las soluciones celulares presentadas en los capítulos anteriores. Hay que destacar que el simulador no ha sido programado *ex profeso* para las familias de sistemas P reconocedores presentadas en esta memoria, sino que ha sido concebido para poder simular cualquier sistema P con membranas activas.

La realización de simulaciones de sistemas celulares en ordenadores ayuda a comprender mejor el funcionamiento de dichos sistemas, y puede servir de ayuda para la detección de errores en los diseños. El simulador que se presenta en esta memoria no busca eficiencia en la simulación (téngase presente que se está simulando un modelo masivamente paralelo en un marco secuencial), sino que trata de seguir, de manera detallada, lo que ocurre en cada paso de evolución del sistema, proporcionando una especie de *historia* de la computación realizada.

El capítulo está organizado como sigue. En primer lugar se hacen unas breves consideraciones que justifican la conveniencia de estudiar simulaciones de sistemas celulares en medios electrónicos. En la Sección 8.2 se presenta un simulador programado en Prolog, que es ilustrado con una sesión para una instancia del problema Knapsack. El capítulo incluye un apéndice donde aparece de manera detallada el conjunto de reglas generadas por el simulador para la instancia presentada como ejemplo.

8.1. Introducción

Desde que Gh. Păun creara la Computación con Membranas como una nueva rama de la Computación Natural, la mayoría de las variantes consideradas han sido sistemas celulares que proporcionan dispositivos computacionales de carácter generativo. Además, en el estudio de dichos sistemas se ha enfatizado el análisis de la completitud computacional de los mismos.

En esta memoria nos hemos centrado en otras variantes (sistemas P reconocedores) que proporcionan un marco especialmente adecuado para atacar la resolución de problemas de decisión, en general, y de problemas numéricos **NP**–completos, en particular (para el caso de sistemas P con membranas activas).

El modelo con el que se trabaja en esta memoria, los sistemas P reconocedores con membranas activas (introducidos en el Capítulo 3), incluye reglas de división de membranas. Estas reglas posibilitan la creación de un espacio de trabajo exponencial en tiempo lineal y, por tanto, permiten diseñar soluciones celulares *eficientes* a problemas **NP**–completos.

Ahora bien, dado que se trata de un modelo de computación orientado a máquinas, a la hora de diseñar una solución celular para un problema, muchas veces es difícil captar de manera intuitiva cómo se comportará el sistema diseñado. Esto hace que la verificación formal de las soluciones celulares diseñadas en este marco sea especialmente complicada.

Por ello parece natural buscar una herramienta que pueda servir de asistente o ayudante para simular computaciones de sistemas celulares. Con este espíritu nace el simulador Prolog que comentaremos a continuación (presentado por primera vez en [14] y [15]). Se trata de un programa que, recibiendo como datos una configuración y un conjunto de reglas, es capaz de ejecutar un paso de transición, detallando toda la información relevante (nueva configuración, reglas usadas y objetos enviados al entorno).

Este simulador nos ha permitido efectuar algunos retoques sobre los procesos de verificación de soluciones celulares que habían sido diseñadas previamente, y de las que se había probado su corrección y completitud de manera formal. Dichas soluciones, a su vez, han sido utilizadas como ejemplos para simular, permitiendo que se detecten y se corrijan algunos errores en el simulador. Es decir, se produce una vez más un proceso mutuo de retroalimentación entre los procesos de diseño y de verificación, como suele ser habitual en la algorítmica tradicional.

Ahora bien, los sistemas celulares son, en general, sistemas no deterministas, con lo que a partir de una configuración es posible llegar a más de una configuración siguiente. Por tanto, la simulación sólo será fiable cuando tratemos con sistemas celulares

confluentes (como es el caso de las familias de sistemas reconocedores consideradas en la clase de complejidad introducida en el Capítulo 2).

8.2. El simulador

En este capítulo se describe una versión revisada del simulador presentado en [15]. Este simulador está escrito en Prolog[1]. Se trata de un lenguaje suficientemente expresivo como para manejar conocimiento simbólico de manera natural, y es capaz de tratar estructuras complejas, como por ejemplo las configuraciones de sistemas celulares (que incluyen información de la estructura de membranas, con su jerarquía, sus etiquetas y sus cargas, así como de los multiconjuntos contenidos en cada membrana).

Por un lado, la estructura de datos basada en árboles y el uso de operadores infijos definidos *ad hoc* por el programador nos permiten *imitar* el lenguaje natural, de manera que un usuario del simulador no necesite conocimientos de Prolog para leer la información acerca de la evolución del sistema simulado. Por otra parte, la estructura de membranas y las reglas de evolución del sistema son tratados por Prolog como *hechos*, y el diseño del motor de inferencia encargado de llevar a cabo los pasos de evolución tiene un tratamiento natural desde el punto de vista de un programador.

En la versión actual[2], el simulador consta de dos partes totalmente distintas. En primer lugar, tenemos el motor de inferencia. Este es un programa que contiene todas las directrices necesarias para llevar a cabo los pasos de transición del sistema: se le proporcionan una configuración y un conjunto de reglas y es capaz de aplicar las reglas de manera paralela y maximal, produciendo una nueva configuración. Conviene resaltar en este punto que este motor de inferencia es completamente general; es decir, no depende del sistema P que se quiera simular. También cabe señalar que por el momento el motor de inferencia no va acompañado de un analizador sintáctico (*parser*). Es decir, no es capaz de detectar si el sistema celular introducido es o no sintácticamente correcto. Este es un punto interesante que deberá ser abordado a corto plazo, a fin de incorporarlo al simulador.

La segunda parte del simulador es una herramienta de generación, que nos permite construir automáticamente las configuraciones iniciales y los multiconjuntos de entrada asociadas a instancias de varios problemas **NP**–completos (en la versión actual: Subset Sum, Knapsack y Partición), siguiendo los esquemas de las correspondientes soluciones celulares. El diseño de esta herramienta de generación, basado en módu-

[1] Para una introducción a Prolog ver [11] o [56].

[2] Disponible vía e-mail: `ariscosn@us.es`

los auxiliares para cada problema, nos permite añadir fácilmente en el futuro más módulos para nuevos problemas. El diseño de los módulos no es demasiado complejo, y cualquier usuario con conocimientos básicos del lenguaje Prolog y de computación con membranas puede diseñar sus propios módulos para realizar experimentos de simulación.

Con objeto de aclarar la notación elegida para representar las configuraciones y las reglas del sistema, recordemos que la estructura de membranas se corresponde con un árbol etiquetado. Identificaremos los nodos del árbol mediante el alfabeto de los corchetes, como sigue: [ ] es la *posición* que denota la raíz del árbol, asociada a la membrana piel; $[i]$ denotará la *posición* de la i-ésima membrana que se encuentra dentro de la piel.

En general, para denotar que en el instante t-ésimo de su evolución, en la configuración de un sistema celular P hay una membrana en la posición $[pos]$ etiquetada por h, con carga eléctrica α y con m como multiconjunto asociado, escribiremos

$$P \texttt{ :: } h \texttt{ ec } \alpha \texttt{ at } [pos] \texttt{ with } m \texttt{ at\_time } t\texttt{.}$$

Las reglas se representarán también como literales, de la siguiente manera:

- $[x \rightarrow y]_h^\alpha$

  P `rule` x `evolves_to` $[y]$ `in` h `ec` α`.`

- $[x]_h^{\alpha_1} \rightarrow [\ ]_h^{\alpha_2} y$

  P `rule` x `inside_of` h `ec` α_1 `sends_out` y `of` h `ec` α_2`.`

- $x[\ ]_h^{\alpha_1} \rightarrow [y]_h^{\alpha_2}$

  P `rule` x `out_of` h `ec` α_1 `sends_in` y `of` h `ec` α_2`.`

- $[x]_h^\alpha \rightarrow y$

  P `rule` x `inside_of` h `ec` α `dissolves_and_sends_out` y`.`

- $[x]_h^{\alpha_1} \rightarrow [y]_h^{\alpha_2}[z]_h^{\alpha_3}$

  P `rule` x `inside_of` h `ec` α_1 `divides_into` y `inside_of` h `ec` α_2
  `and` z `inside_of` h `ec` α_3`.`

En todos los casos cada regla de evolución del modelo celular se transforma en un *hecho* Prolog. Se han definido *ad hoc* varios operadores infijos (como `inside_of`, `divides_into`, etc.) para lograr una apariencia que imite al lenguaje natural.

8.3. Una sesión del simulador Prolog

En esta sección se incluye una sesión del simulador sobre una instancia del problema Knapsack, $u = (4, (3, 2, 3, 1), (1, 3, 3, 2), 3, 4)$. Esto es, consideramos el conjunto $A = \{a_1, a_2, a_3, a_4\}$ ($n = 4$), con los pesos $w(a_1) = 3$, $w(a_2) = 2$, $w(a_3) = 3$, $w(a_4) = 1$ y los valores $v(a_1) = 1$, $v(a_2) = 3$, $v(a_3) = 3$ y $v(a_4) = 2$. Se trata de decidir si existe un subconjunto $B \subseteq A$ tal que el peso de B no supere 3 ($k = 3$) y su valor sea mayor que 4 ($c = 4$).

Siguiendo el diseño presentado en el Capítulo 5, el sistema P que se encarga de resolver esa instancia es $\Pi(819) = \Pi(\langle 4, 3, 4\rangle)$, con entrada $x_1^3\, x_2^2\, x_3^3\, x_4\, y_1\, y_2^3\, y_3^3\, y_4^2$. La configuración inicial asociada a dicho multiconjunto de entrada se muestra en la Figura 8.1, y la representación[3] de dicha configuración en Prolog consiste en los siguientes *hechos* (`p1` es un nombre asignado al sistema $\Pi(\langle 4, 3, 4\rangle)$):

```
p1 :: s ec 0 at [] with [z0] at_time 0.
p1 :: e ec 0 at[1] with [e0, a_, a_, a_, b_, b_, b_, b_,
                         x1, x1, x1, x2, x2, x3, x3, x3,
                         x4, y1, y2, y2, y2, y3, y3, y3,
                         y4, y4] at_time 0.
```

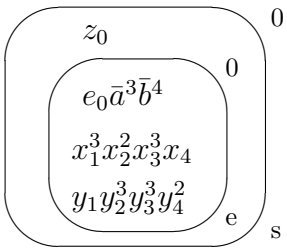

Figura 8.1: Configuración inicial

El simulador consta de un módulo (`generator.pl`) que es capaz de generar estos hechos automáticamente a partir de la instancia introducida por el usuario, y almacenarlos en un fichero de texto. Para ello se ha creado un fichero auxiliar (`knp_file.pl`) que contiene las "instrucciones" para determinar la configuración inicial asociada a la entrada específica para la instancia, según las indicaciones del diseño presentado en el Capítulo 5.

Nótese que las multiplicidades de los objetos x_j e y_j se corresponden con los pesos y los valores, respectivamente, de los elementos a_j. Asimismo, están presentes tres copias del objeto $\bar{a}$ ($k = 3$) y cuatro del objeto $\bar{b}$ ($c = 4$).

[3]Obsérvese que se han usado caracteres ASCII para representar los objetos del alfabeto del sistema P; por ejemplo, `a_` representa al objeto $\bar{a}$, `b0g` representa al objeto $\hat{b}_0$, y así sucesivamente.

Además de la configuración inicial, el simulador genera automáticamente el conjunto de reglas asociado a la instancia del problema. Esta generación se lleva a cabo instanciando los esquemas de reglas presentado en el Capítulo 5 para los valores concretos de los parámetros n, k y c. Se produce otro fichero de texto que contiene los hechos Prolog que representan a las reglas. Nótese que, dado que el conjunto de reglas sólo depende de los valores de n, k y c, si quisiéramos resolver varias instancias con los mismos parámetros, sólo se necesita generar una vez el conjunto de reglas. El fichero que las contiene puede ser fácilmente editado, modificado o reutilizado por el usuario.

En el ejemplo que hemos elegido se obtienen 89 reglas, que pueden consultarse en el Apéndice de este capítulo.

Pasemos ya a detallar cómo se desarrolla una sesión del simulador. Supongamos que tenemos un terminal Prolog, y que ya tenemos cargados los ficheros con las reglas de $\Pi(\langle 4,3,4\rangle)$ (`p1`) y con la configuración inicial asociada a la entrada (ver Figura 8.1). Para comenzar la simulación de la evolución del sistema, tecleamos el siguiente comando:

```
?- evolve(p1,0).
```

El simulador nos devuelve la configuración tras el primer paso de computación (instante `at_time 1`) y detalla el conjunto de reglas usadas en este paso, especificando cuántas veces se ha usado cada una. Además, si algún objeto es enviado fuera de la piel, este hecho también será comunicado al usuario. Para mostrar en pantalla los multiconjuntos de objetos se utiliza una notación en forma de pares `obj`–`n`, donde `obj` es un objeto del alfabeto y `n` es la multiplicidad de `obj` en el multiconjunto.

```
?- evolve(p1,0).

p1 :: s ec 0 at [] with [z1-1] at_time 1
p1 :: e ec -1 at [1] with [a_-3, b_-4, q-1, x1-3, x2-2, x3-3,
                           x4-1, y1-1, y2-3, y3-3, y4-2] at_time 1
p1 :: e ec 1 at [2] with [a_-3, b_-4, e0-1, x1-3, x2-2, x3-3,
                           x4-1, y1-1, y2-3, y3-3, y4-2] at_time 1

Used rules in the step 1:
  * The rule 1 has been used only once
  * The rule 57 has been used only once
```

En este primer paso únicamente se aplican las reglas 1 y 57. La regla 1 es una regla de división: la membrana etiquetada por e en la posición `[1]` de la estructura de membranas se divide, produciendo dos membranas que son colocadas en las posiciones

`[1]` y `[2]`; la regla 57 es una regla de evolución, el objeto z_0 evoluciona a z_1 dentro de la membrana piel.

Para obtener la siguiente configuración de la evolución de `p1`, ahora tecleamos:

```
?- evolve(p1,1).

p1 :: s ec 0 at [] with [z2-1] at_time 2
p1 :: e ec -1 at [1] with [a-3, bg-4, q0-1, q_-1, x1-3,
           x2-2, x3-3, x4-1, y1-1, y2-3, y3-3, y4-2] at_time 2
p1 :: e ec 0 at [2] with [a_-3, b_-4, e1-1, x0-3, x1-2,
           x2-3, x3-1, y0-1, y1-3, y2-3, y3-2] at_time 2
p1 :: e ec 1 at [3] with [a_-3, b_-4, e1-1, x0-3, x1-2,
           x2-3, x3-1, y0-1, y1-3, y2-3, y3-2] at_time 2

Used rules in the step 2:
  * The rule 6 has been used only once
  * The rule 14 has been used 3 times
  * The rule 15 has been used 2 times
             ...
           reglas de evolución del apartado (b) (ver Apéndice)
             ...
  * The rule 22 has been used only once
  * The rule 25 has been used 4 times
  * The rule 26 has been used 3 times
  * The rule 58 has been used only once
```

En este paso aparece la primera membrana relevante (es decir, una membrana que tiene carga negativa y contiene al objeto `q0`) aparece en la posición `[1]`. El subconjunto asociado a esta membrana (Definición 4.1) es el conjunto vacío. Por su parte, la membrana que ocupa la posición `[2]` continuará dividiéndose y generando nuevas membranas. El subconjunto asociado a esta última membrana, así como los asociados a sus descendientes, contendrán todos al objeto a_1. Y, por último, la membrana que ocupa la posición `[3]` dará origen a todas las membranas asociadas a subconjuntos de A (no vacíos) que no contengan al objeto a_1.

Nótese que se han aplicado reglas del apartado (b). Esto quiere decir que la *fase de cálculo* (de pesos y valores) ya ha comenzado. Esta fase se ejecuta en cada membrana en paralelo con la *fase de generación*. Inmediatamente después de que un elemento a_j sea añadido al subconjunto asociado (es decir, cuando está presente el objeto `ej` y

la membrana tiene carga neutra), $w(a_j)$ nuevas copias del objeto `a0_` y $v(a_j)$ nuevas copias de `b0_` son generadas en la membrana. Este proceso está regido por las reglas del apartado (b), en función de los cambios de polaridad que se produzcan por la acción de las reglas de división.

Recordemos que el objetivo de la primera fase es generar una única membrana relevante para cada subconjunto de A, esto es, en total $2^n = 2^4 = 16$ membranas relevantes. También se generan membranas "sobrantes", que se quedan bloqueadas cuando aparece el objeto e_n y la carga eléctrica es positiva. Esto se debe a cuestiones técnicas del diseño, como se comentó en los Capítulos 4, 5 y 6 (ver Figura 6.2). Esto se lleva a cabo en los $2n + 2 = 10$ primeros pasos de computación.

El simulador también ofrece la posibilidad de mostrar la configuración en un cierto instante t sin necesidad de simular y mostrar todos los pasos previos. En este caso el comando que hay que teclear es:

```
?- configuration(p1,4).
       ...
p1 :: e ec -1 at [2] with [a-3, a0-3, b0g-1, bg-4, q0-1, q_-1,
                x1-2, x2-3, x3-1, y1-3, y2-3, y3-2] at_time 4
       ...
```

En este instante observamos que aparece la segunda membrana relevante. Nótese que la membrana que ocupa la posición `[2]` está cargada negativamente (`ec -1`) y contiene al objeto q_0 (`q0-1` está en la lista que representa al multiconjunto de la membrana). El subconjunto asociado a esta membrana es $\{a_1\}$. Podemos observar también que en el multiconjunto de la membrana aparecen tres copias de `a0` ($w(a_1) = 3$) y una copia de `b0g` ($v(a_1) = 1$).

```
?- evolve(p1,4).
       ...
p1 :: e ec  0 at [2] with [a-3, a0-2, b0g-1, bg-4, q1-1,
            q_-1, x1-2, x2-3, x3-1, y1-3, y2-3, y3-2] at_time 5
p1 :: e ec -1 at [3] with [a-3, a0-2, b0g-3, bg-4, q0-1,
            q_-1, x1-3, x2-1, y1-3, y2-2] at_time 5
       ...

Used rules in the step 5:
       ...
  * The rule 27 has been used only once
       ...
```

La membrana que ocupa la posición `[2]` comienza ahora su fase de chequeo para w (se ha aplicado la regla 27, que pertenece al apartado (d), ver Apéndice). Además, una nueva membrana relevante aparece en la posición `[3]`. El subconjunto asociado a esta membrana es $\{a_2\}$. En los pasos siguientes seguirán apareciendo otras membranas relevantes, y darán comienzo sus correspondientes fases de chequeo.

Pasemos directamente al paso $2n + 2 = 10$. En este paso aparece la membrana relevante asociada al subconjunto total. Se trata de la última membrana relevante que aparece en la computación; es decir, ya no se producirán más divisiones en los siguientes pasos y no aparecerán nuevas membranas relevantes.

```
?- configuration(p1,10).

        ...
p1 :: e ec -1 at [12] with [a-2, a0-2, b0g-5, bg-4, q2-1, q_-1]
                                                    at_time 10
        ...
p1 :: e ec -1 at [24] with [a-3, a0-9, b0g-9, bg-4, q0-1, q_-1]
                                                    at_time 10
        ...
```

La membrana que ocupa la posición `[24]` es la membrana relevante cuyo subconjunto asociado es A. En su multiconjunto aparecen nueve copias de `a0` ($w(A) = 9$) y otras nueve copias de `b0g` ($v(A) = 9$).

Vamos a centrarnos en otra membrana, la que ocupa la posición `[12]`. Esta membrana está asociada al subconjunto $B = \{a_2, a_4\}$, el único que es solución para la instancia considerada. Ya ha ejecutado dos pasos de su fase de chequeo para w (obsérvese el contador `q2`). Esta fase terminará con éxito cuando se aplique la regla $[q_{2j+1}]_e^- \rightarrow [\ ]_e^+\#$ para $j = w(B) = 3$ (ver en la Sección 5.2 las reglas del apartado (f)). Eso sucederá después de seis pasos: en primer lugar, el contador debe evolucionar de q_2 a q_7, lo que requiere cinco pasos de evolución, y luego se necesita otro paso más para que salga de la membrana, cambiando la carga.

```
?- configuration(p1,16).
            ...
p1 :: e ec 1 at [12] with [b0g-5, bg-4, q_-1] at_time 16
            ...
```

A continuación, se ejecuta un paso de transición que conduce a la fase de chequeo para la función v. Dicho paso consiste en un renombramiento de los objetos (ver reglas del apartado (g)), tras el cual tenemos el valor del subconjunto asociado representado

mediante la multiplicidad del objeto `b0`, y la constante c mediante la multiplicidad de `b`. Durante esta fase, análogamente a la anterior, se van eliminando alternativamente dichos objetos hasta agotar alguno de ellos. Veamos cuál es el estado de la membrana `[12]` transcurridos el paso de renombramiento y $2c = 8$ pasos de comparación:

```
?- configuration(p1,25).
      ...
p1 :: e ec 1 at [12] with [b0-1, q_8-1] at_time 25
      ...
```

Podemos observar que la fase de chequeo para v va a terminar con éxito, ya que el número de copias del objeto `b0` al inicio de la fase era mayor que el número de objetos `b`. Veamos con detalle los últimos pasos de esta fase y cómo se enlaza con la *fase de respuesta*.

```
?- evolve(p1,25).

p1 :: s ec 0 at [] with [# -127, z26-1] at_time 26
           ...
p1 :: e ec 0 at [12] with [q_9-1] at_time 26
           ...
```

La membrana en la posición `[12]` ha superado con éxito las dos fases de chequeo (para w y para v), así que en el siguiente paso se enviará un objeto Yes a la piel (ver regla 56 en el Apéndice).

```
?- evolve(p1,26).

p1 :: s ec 0 at [] with [# -127, yes-1, z27-1] at_time 27
           ...
p1 :: e ec 0 at [12] with [] at_time 27
           ...

Used rules in the step 27:
  * The rule 56 has been used only once
  * The rule 83 has been used only once
```

Aunque el objeto Yes esté presente en la piel, el sistema no lo expulsará aún. Como se explicó en el Capítulo 5, el contador z_j de la piel esperará unos pasos más antes de producir los objetos especiales d_+ y d_- (esto se hace para asegurarse que los chequeos han finalizado en todas las membranas).

```
?- configuration(p1,29).

p1 :: s ec 0 at [] with [# -127, d1-1, d2-1, yes-1] at_time 29
       ...
```

En este instante todos los procesos internos han concluido. El único objeto que ha evolucionado en el paso anterior es `z28` (ver reglas del apartado (k) en la Sección 5.2, teniendo presente que $28 = 2n + 2k + 2c + 6$). Simulemos otro paso más de computación.

```
?- evolve(p1,29).

p1 :: s ec 1 at [] with [# -127, d2-1, yes-1] at_time 30
      ...

Used rules in the step 30:
  * The rule 86 has been used only once
The P-system has sent out d1 at step 30
```

En este paso el objeto `d1` ha abandonado el sistema y la carga de la piel ha cambiado a positiva.

```
?- evolve(p1,30).

p1 :: s ec 0 at [] with [# -127, no-1] at_time 31
        ...

Used rules in the step 31:
  * The rule 87 has been used only once
  * The rule 88 has been used only once
The P-system has sent out d1 at step 30
The P-system has sent out yes at step 31
```

El sistema ha enviado al entorno un objeto `yes` como respuesta. Podemos observar cómo dentro de la piel aparece un objeto `no`, pero ya no puede ser enviado fuera porque ahora la piel vuelve a tener carga neutra. Para poner a prueba el sistema, y ver si la respuesta se ha enviado exactamente en el último paso de computación, intentamos simular un nuevo paso de transición.

```
?- evolve(p1,31).
No more evolution!
```

```
The P-system p1 has already reached a halting configuration at
 step 31
```

El simulador nos indica que el sistema ha llegado ya a una configuración de parada y, en consecuencia, no puede evolucionar más.

Apéndice

A continuación se muestran las reglas de evolución generadas por el simulador (de hecho, por el módulo `generator.pl`) para la instancia del problema Knapsack considerada en la Sección 8.3. Este conjunto de reglas sigue los esquemas del diseño presentado en el Capítulo 5. El número que aparece detrás de la marca `**` es el ordinal de la regla correspondiente.

```
% Set (a)
p1 rule e0 inside_of e ec 0 divides_into q inside_of e ec -1
   and e0 inside_of e ec 1 ** 1.
            ...
p1 rule e4 inside_of e ec 0 divides_into q inside_of e ec -1
   and e4 inside_of e ec 1 ** 5.

p1 rule e0 inside_of e ec 1 divides_into e1 inside_of e ec 0
   and e1 inside_of e ec 1 ** 6.
            ...
p1 rule e3 inside_of e ec 1 divides_into e4 inside_of e ec 0
   and e4 inside_of e ec 1 ** 9.

% Set (b)
p1 rule x0 evolves_to [a0_] in e ec 0 ** 10.

p1 rule y0 evolves_to [b0_] in e ec 0 ** 11.

p1 rule x0 evolves_to [] in e ec 1 ** 12.

p1 rule y0 evolves_to [] in e ec 1 ** 13.

p1 rule x1 evolves_to [x0] in e ec 1 ** 14.
             ...
```

```
p1 rule x4 evolves_to [x3] in e ec 1 ** 17.

p1 rule y1 evolves_to [y0] in e ec 1 ** 18.
            ...
p1 rule y4 evolves_to [y3] in e ec 1 ** 21.

% Set (c)
p1 rule q evolves_to [q_, q0] in e ec -1 ** 22.

p1 rule b0_ evolves_to [b0g] in e ec -1 ** 23.

p1 rule a0_ evolves_to [a0] in e ec -1 ** 24.

p1 rule b_ evolves_to [bg] in e ec -1 ** 25.

p1 rule a_ evolves_to [a] in e ec -1 ** 26.

% Set (d)
p1 rule a0 inside_of e ec -1 sends_out # of e ec 0 ** 27.

p1 rule a inside_of e ec 0 sends_out # of e ec -1 ** 28.

% Set (e)
p1 rule q0 evolves_to [q1] in e ec -1 ** 29.

p1 rule q2 evolves_to [q3] in e ec -1 ** 30.

p1 rule q4 evolves_to [q5] in e ec -1 ** 31.

p1 rule q6 evolves_to [q7] in e ec -1 ** 32.

p1 rule q1 evolves_to [q2] in e ec 0 ** 33.

p1 rule q3 evolves_to [q4] in e ec 0 ** 34.

p1 rule q5 evolves_to [q6] in e ec 0 ** 35.
```

```
% Set (f)
p1 rule q1 inside_of e ec -1 sends_out # of e ec 1 ** 36.

p1 rule q3 inside_of e ec -1 sends_out # of e ec 1 ** 37.

p1 rule q5 inside_of e ec -1 sends_out # of e ec 1 ** 38.

p1 rule q7 inside_of e ec -1 sends_out # of e ec 1 ** 39.

% Set (g)
p1 rule q_ evolves_to [q_0] in e ec 1 ** 40.

p1 rule b0g evolves_to [b0] in e ec 1 ** 41.

p1 rule bg evolves_to [b] in e ec 1 ** 42.

p1 rule a evolves_to [] in e ec 1 ** 43.

% Set (h)
p1 rule b0 inside_of e ec 1 sends_out # of e ec 0 ** 44.

p1 rule b inside_of e ec 0 sends_out # of e ec 1 ** 45.

% Set (i)
p1 rule q_0 evolves_to [q_1] in e ec 1 ** 46.

p1 rule q_2 evolves_to [q_3] in e ec 1 ** 47.

p1 rule q_4 evolves_to [q_5] in e ec 1 ** 48.

p1 rule q_6 evolves_to [q_7] in e ec 1 ** 49.

p1 rule q_8 evolves_to [q_9] in e ec 1 ** 50.

p1 rule q_1 evolves_to [q_2] in e ec 0 ** 51.

p1 rule q_3 evolves_to [q_4] in e ec 0 ** 52.
```

```
p1 rule q_5 evolves_to [q_6] in e ec 0 ** 53.

p1 rule q_7 evolves_to [q_8] in e ec 0 ** 54.

% Set (j)
p1 rule q_9 inside_of e ec 1 sends_out yes of e ec 0 ** 55.

p1 rule q_9 inside_of e ec 0 sends_out yes of e ec 0 ** 56.

% Set (k)
p1 rule z0 evolves_to [z1] in s ec 0 ** 57.

p1 rule z1 evolves_to [z2] in s ec 0 ** 58.
            ...
p1 rule z26 evolves_to [z27] in s ec 0 ** 83.

p1 rule z27 evolves_to [z28] in s ec 0 ** 84.

p1 rule z28 evolves_to [d1, d2] in s ec 0 ** 85.

% Set (l)
p1 rule d1 inside_of s ec 0 sends_out d1 of s ec 1 ** 86.

p1 rule d2 evolves_to [no] in s ec 1 ** 87.

p1 rule yes inside_of s ec 1 sends_out yes of s ec 0 ** 88.

p1 rule no inside_of s ec 1 sends_out no of s ec 0 ** 89.
```

Capítulo 9

Sobre Complejidad Descriptiva de sistemas P

En este capítulo se aborda el problema de describir la complejidad de los procesos de evolución que se producen en las distintas computaciones de los sistemas P. Este problema es especialmente complicado en el caso de los sistemas con membranas activas, en los que el número de pasos de la computación es un dato insuficiente para reflejar la complejidad de la misma. Las *Sevilla Carpets* (literalmente: *alfombras de Sevilla*) fueron introducidas en [12], y describen la complejidad espacio-temporal de los sistemas celulares. Basándonos en ellas, introduciremos algunos nuevos parámetros que pueden ser de utilidad para comparar las evoluciones de distintos sistemas celulares.

A lo largo del capítulo se pone de manifiesto la necesidad de profundizar en el estudio de nuevos parámetros que permitan analizar desde nuevas perspectivas la complejidad de los sistemas P como dispositivos computacionales que resuelven problemas de decisión. Esos parámetros pueden orientar al usuario en la tarea de diseñar *mejores* sistemas que resuelven un mismo problema.

El capítulo está estructurado como sigue. En primer lugar se justifica brevemente la necesidad de considerar una complejidad descriptiva que complemente la complejidad computacional, en el marco de los sistemas celulares con los que se trabaja en esta memoria. En la Sección 9.2 se presentan las Sevilla Carpets, para seguidamente introducir una serie de nuevos parámetros relacionados con ellas en la Sección 9.3. Además, como ejemplo de ilustración, compararemos en la Sección 9.4 a través de dichos parámetros las computaciones correspondientes a dos soluciones celulares (distintas) para el problema Subset Sum ejecutadas sobre la misma instancia. Finalmente, se presentan las descripciones de las computaciones correspondientes a dos instancias

de los problemas Knapsack y Partición, en las Secciones 9.5 y 9.6.

9.1. Introducción

La evolución de un sistema P es un proceso complejo en el que intervienen (eventualmente) un gran número de objetos, membranas y reglas. Nos centraremos en los sistemas celulares con membranas activas, en donde el problema de describir la complejidad del proceso computacional tiene una dificultad especial.

Las membranas elementales pueden dividirse dando lugar a dos nuevas membranas y, mediante la reiteración del proceso, teniendo en cuenta el paralelismo inherente a los sistemas celulares, se puede obtener una cantidad exponencial de membranas en tiempo polinomial. Esta posibilidad hace de los sistemas P con membranas activas una herramienta muy potente para atacar la resolución de problemas **NP**–completos y, de hecho, se han publicado distintas soluciones eficientes a este tipo de problemas (véase por ejemplo [42] o [46], y las soluciones presentadas en los Capítulos 4, 5 y 6 de esta memoria).

Todas estas soluciones están diseñadas dentro del marco de los *sistemas reconocedores (con entrada y con salida externa)*, y presentan similitudes significativas entre ellas. La idea básica subyacente en los diseños es la creación de un número exponencial de membranas (*espacio de trabajo*) en tiempo polinomial, y el uso de cada membrana como mecanismo computacional independiente. Todas las membranas evolucionan *en paralelo* y la computación tiene un coste en tiempo polinomial. El proceso termina con una fase final (de nuevo con coste polinomial) que interpreta las respuestas de todos esos mecanismos y envía la salida al entorno. De todo ello, se concluye que la complejidad en *tiempo* (de computación celular) de dichas soluciones es polinomial, pero parece evidente que el tiempo no debe ser la única variable a considerar a la hora de evaluar la complejidad del proceso.

Es aquí donde entra en juego el estudio de la complejidad descriptiva. Dependiendo de las restricciones existentes en cada caso, principalmente en lo que concierne a la implementación real de soluciones celulares, puede resultar muy importante conocer el nivel de paralelismo que se da en la computación asociada a una instancia (número de reglas que se aplican en un cierto paso, número de membranas que evolucionan en paralelo, etc.), o bien conocer los objetos y/o reglas que pueden intervenir de manera más relevante en algún paso de la computación considerada. En este capítulo iniciamos un camino en esta dirección.

A continuación, se presenta una representación bidimensional útil para describir la computación de sistemas celulares, debida a G. Ciobanu, Gh. Păun y Gh. Stefănescu.

9.2. Sevilla Carpets

Dado que las Sevilla Carpets están inspiradas en una noción definida en el marco de los Lenguajes Formales (los lenguajes de Szilard), introduzcamos brevemente dicha noción.

A los modelos que se manejan en teoría de Lenguajes Formales se les denomina *gramáticas*, y constan de un conjunto de reglas de reescritura (o *reglas de evolución*) definidas sobre un cierto alfabeto (donde se tiene un subconjunto distinguido de *objetos terminales*). Partiendo de un objeto inicial (o *axioma*), y mediante la aplicación sucesiva de las reglas de evolución se puede obtener, eventualmente, una cadena de símbolos terminales. En ese caso se dice que la computación es exitosa y que dicha cadena es una *palabra* del *lenguaje generado* por la gramática.

En las gramáticas clásicas (jerarquía de Chomsky[1]) se suele considerar que en cada paso sólo se puede aplicar una regla, por lo que una manera de representar una computación es escribir la secuencia de las reglas aplicadas como una cadena de etiquetas (suponiendo que se tiene un etiquetado unívoco de las reglas). El conjunto de todas las cadenas (o *palabras*) de etiquetas que representan computaciones exitosas se conoce como *lenguaje de Szilard* asociado a la gramática (ver [28, 32] o [50]).

Ampliando la noción de lenguaje de Szilard a los casos donde se pueden aplicar simultáneamente más de una regla, G. Ciobanu, Gh. Păun y Gh. Stefănescu presentaron en [12] una nueva forma de describir la complejidad de una computación de un sistema P: las llamadas *Sevilla Carpets*.

La *alfombra* (o *Sevilla Carpet*) asociada a una computación de un sistema P es una tabla bidimensional tal que en el eje horizontal se coloca el tiempo (número de pasos) y a lo largo del eje vertical se disponen las etiquetas de las reglas, de manera que en cada celda de la tabla se da información acerca de una regla en un cierto paso. Dependiendo del tipo y la cantidad de información que se aporta para describir la evolución, los autores citados anteriormente proponen cinco variantes de Sevilla Carpets:

1. Se especifica para cada unidad de tiempo y cada membrana si, al menos, una regla es aplicada en esa región.

2. Se especifica para cada unidad de tiempo y cada regla si ésta es aplicada o no.

3. Se explicita para cada unidad de tiempo y cada regla el número de aplicaciones de dicha regla en ese paso (este número es cero cuando la regla no se aplica, pero

[1] No entraremos aquí en detalles: para una introducción de los preliminares básicos de teoría de lenguajes formales ver [31]; un desarrollo mucho más completo y detallado puede encontrarse en [48].

puede ser arbitrariamente grande cuando los multiconjuntos de objetos que se estén manejando sean grandes). Esta será la variante elegida a lo largo de este capítulo.

4. Se especifica cuál de los siguientes casos se da: la regla no es aplicable, la regla es aplicable pero no se ha aplicado en el paso considerado debido a una elección no determinista o, por último, la regla es aplicada.

5. Se le asigna un coste a cada regla, y se multiplica dicho coste por el número de veces que la regla es aplicada.

Además, en [12] se define el *peso* de la tabla como la suma de todos sus elementos. Dicho valor coincide con el número total de aplicaciones de reglas durante la computación, y puede ser usado también como medida de complejidad.

Es posible lograr una imágen gráfica de las alfombras como superficies si se proyectan los datos en el espacio tridimensional. Se procede como sigue: se coloca la tabla sobre el plano XY, y los valores de las celdas de la tabla se representan como alturas sobre el eje Z. Debido a la naturaleza discreta de los datos (el número de pasos, las etiquetas de las reglas y el número de aplicaciones son todos números naturales) lo que realmente se obtiene es una nube de puntos, pero se puede tratar de acomodar todos esos puntos en una superficie, más o menos irregular. Más adelante veremos varios ejemplos.

9.3. Parámetros de Complejidad Descriptiva

Como se ha dicho en la sección anterior, en muchas ocasiones no sólo estamos interesados en el número de pasos celulares de una computación, sino también en otro tipo de recursos que se requieren para llevar a cabo la computación. Especialmente si se quiere realizar una implementación *in silico* de un sistema celular, hay que tener en cuenta el número de veces que se aplican las reglas, y quizás también el número de membranas y/o de objetos presentes en cada configuración.

Para enriquecer la descripción de la complejidad de la computación, proponemos los siguientes parámetros:

- **Peso:** introducido en la sección anterior, se define como el número total de aplicaciones de reglas durante la computación. Si se le asigna un cierto coste a la ejecución de una regla, entonces el peso refleja el coste total de la computación.

- **Superficie:** se obtiene multiplicando el número de pasos de la computación por el número de reglas del sistema. Gráficamente, representa la superficie sobre la que se asienta la tabla; podría decirse que es el *tamaño potencial* de la

computación. Desde un punto de vista computacional, no estamos interesados únicamente en que los sistemas celulares que diseñemos ejecuten computaciones que duren *pocos* pasos, sino en que la cantidad de recursos utilizados sea lo menor posible. La superficie de la Sevilla Carpet es una medida de los recursos utilizados en el diseño del sistema.

- **Altura:** es el valor máximo de la tabla. Es decir, el máximo número de aplicaciones de una regla en un paso de computación. Gráficamente, se corresponde con el punto más alto de la Sevilla Carpet.

- **Peso medio:** se obtiene dividiendo el *peso* entre la *superficie* de la Sevilla Carpet. Por tanto, expresa la relación entre ambos parámetros, indicando el grado en el que el sistema explota su paralelismo masivo.

9.4. Comparación de diseños celulares

Seguidamente vamos a presentar las Sevilla Carpets correspondientes a dos familias de sistemas reconocedores que resuelven el problema Subset Sum, una será la presentada en el Capítulo 4 y la otra será una variante inspirada en los comentarios acerca de subrutinas reutilizables que se vieron en el Capítulo 7. Se trata de utilizar los parámetros introducidos en la sección anterior para encontrar diferencias entre las computaciones correspondientes a los dos diseños actuando sobre una misma instancia. Consideremos, por ejemplo, la instancia $n = 5, k = 9$ y $w_1 = 3, w_2 = 5, w_3 = 3, w_4 = 2, w_5 = 5$ del problema Subset Sum.

9.4.1. Primera solución

En primer lugar, vamos a comentar con algo de detalle la complejidad de las computaciones que realizan los sistemas celulares de la familia diseñada en la Sección 4.2 para resolver el problema Subset Sum.

En el apartado de acotación polinomial de la familia se estudió el número de pasos celulares de las computaciones, probando que era de orden lineal. Ahora bien, como ya hemos dicho, en algunos casos no sólo nos interesa conocer el número de pasos que efectúa el sistema, sino que también queremos saber el número de reglas aplicadas en cada paso. Este número guarda relación con el número total de objetos y de membranas que hay en cada configuración, además de, como es lógico, con el número de reglas de evolución del sistema. En el caso que nos ocupa, el diseño depende de dos constantes que se dan como entrada en el problema: n y k. Se tienen $5n+5k+18$ reglas de evolución, y si se introduce un multiconjunto de entrada apropiado dentro de la

membrana interna e antes de iniciar la computación, el sistema parará y devolverá una respuesta a los $2n+2k+6$ pasos de computación (si la respuesta es No) o en $2n+2k+5$ pasos (si la respuesta es Yes).

Se trata de resolver la instancia $u = (5, (3, 5, 3, 2, 5), 9)$ del problema Subset Sum. El sistema celular que se encarga de resolverlo, $\Pi(119) = \Pi(\langle 5, 9\rangle)$, consta de 88 reglas de evolución, y si le proporcionamos la entrada adecuada, $cod(u) = x_1^3\, x_2^5\, x_3^3\, x_4^2\, x_5^5$, entonces el sistema resuelve la instancia en $2n+2k+5 = 33$ pasos celulares, expulsando un objeto No.

Presentamos una expresión gráfica de la Sevilla Carpet asociada a dicha computación en la Figura 9.1.

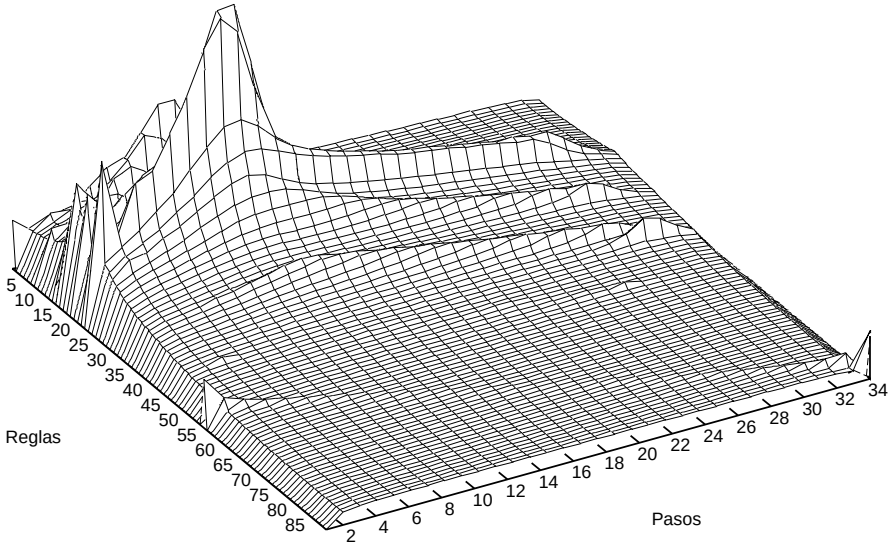

Figura 9.1: Computación de $\Pi(\langle 5, 9\rangle)$ con entrada $cod(u) = x_1^3\, x_2^2\, x_3^7\, x_4^3\, x_5^4$

A continuación, incluimos algunos comentarios que ayudan a describir la complejidad de la computación. Ya hemos indicado el número de pasos celulares que dura la computación, vamos a comentar la "intensidad" de la computación, en el sentido de cuántas aplicaciones de reglas se producen en cada paso. Todas las reglas de evolución son aplicadas en algún momento de la computación, con la excepción de las siguientes reglas: $[q_{19}]_e^- \to [\ ]_e^0 Yes$, $[q_3]_e^- \to [\ ]_e^-\#$, $[q_9]_e^- \to [\ ]_e^-\#$ y $[Yes]_s^- \to [\ ]_s^0 Yes$.

El *peso* de la Sevilla Carpet (el número total de aplicaciones de reglas durante la computación) es 2179. La *altura* de la Sevilla Carpet (el máximo número de veces que se aplica una regla en un paso de evolución) se alcanza en el paso 9, cuando la regla $[\bar{a}_0 \to a_0]_e^-$ es aplicada 82 veces. La *superficie* de la Sevilla Carpet es 2904, y la *altura media* es 0'749656.

En cuanto a la interpretación visual de la Sevilla Carpet, podemos destacar cómo los "picos" de la gráfica se corresponden a los primeros instantes de la computación y a las reglas de los primeros grupos ((a), (b) y (c)). Es decir, durante los primeros pasos de computación, debido al aumento exponencial de membranas que produce la ejecución de las reglas de división, el número de objetos involucrados en la fase de cálculo de pesos (es decir, afectados por reglas de reescritura del grupo (b) o por las reglas de "limpieza" del grupo (c)) también crece de forma exponencial y, por tanto, se produce un número muy elevado de aplicaciones de reglas.

Además, otro detalle a resaltar es la marca que deja en la superficie la evolución del contador z_i. Las reglas que rigen la evolución de ese contador son las del grupo (g), que ocupan casi las últimas posiciones en la enumeración de las reglas del sistema (para la instancia considerada, desde la posición 54 a la 84). Como se puede apreciar, si en un instante se aplica la regla $[z_i \to z_{i+1}]_s^0$, en el instante siguiente se aplicará la regla $[z_{i+1} \to z_{i+2}]_s^0$ (que ocupa el siguiente lugar en la enumeración de las reglas). Esto provoca que aparezca en la gráfica una estela en diagonal en la zona correspondiente a dichas reglas y durante prácticamente todos los pasos de la computación (después de que el contador z_i desaparezca solo se ejecutan tres pasos más).

9.4.2. Segunda solución

A continuación presentamos una nueva familia de sistemas celulares reconocedores que resuelve el problema Subset Sum, inspirada en la anterior. Se han hecho algunas modificaciones, siguiendo las ideas presentadas en el Capítulo 7 (recordemos que se daban esquemas de reglas generales para realizar bucles de comparación y detección de fin de la computación, siendo, además, diseños uniformes).

Para cada $n \in \mathbb{N}$, consideramos el sistema $(\Pi_2(n), \Sigma(n), i(n))$, donde el alfabeto de entrada es $\Sigma(n) = \{x_1, \ldots, x_n\}$, la membrana de entrada es $i(n) = e$ y $\Pi_2(n) = (\Gamma(n), \{e, r, s\}, \mu, \mathcal{M}_e, \mathcal{M}_r, \mathcal{M}_s, R)$ se define como sigue:

- Alfabeto: $\Gamma(n) = \Sigma(n) \cup \{\bar{a}_0, \bar{a}, a_0, a, c, d_0, d_1, d_2, e_0, \ldots, e_n, g, \bar{g}, \hat{g}, h_0, h_1,$
$q, q_0, q_1, q_2, q_3, Yes, No, No_0, z_0, \ldots, z_{2n+1}, \#\}$

- Estructura de membranas: $\mu = [\ [\ ]_e^0\ ]_s^0$.

- Multiconjuntos iniciales: $\mathcal{M}_s = z_0$; $\mathcal{M}_e = e_0 g \bar{a}^k$; $\mathcal{M}_r = h_0 b$.

- El conjunto de reglas de evolución, R, consta de las siguientes reglas:

(fase de generación)

(a) $[e_i]_e^0 \to [q]_e^- [e_i]_e^+$, para $i = 0, \ldots, n$.
$[e_i]_e^+ \to [e_{i+1}]_e^0 [e_{i+1}]_e^+$, para $i = 0, \ldots, n-1$.

(fase de cálculo de pesos)

(*b*) $[x_0 \to \bar{a}_0]_e^0$; $[x_0 \to \lambda]_e^+$; $[x_i \to x_{i-1}]_e^+$, para $i = 1, \ldots, n$.

(renombramiento / limpieza)

(*c*) $[q \to q_0]_e^-$; $[\bar{a}_0 \to a_0]_e^-$; $[\bar{a} \to a]_e^-$.
$[g]_e^- \to \bar{g}[\ ]_e^-$.

$[e_n]_e^+ \to \#$.
$[\bar{a}_0 \to \lambda]_s^0$; $[\bar{a} \to \lambda]_s^0$; $[g \to \lambda]_s^0$.
$[a \to \lambda]_e^+$; $[a_0 \to \lambda]_e^+$.

(fase de chequeo)

(*d*) $[a_0]_e^- \to [\ ]_e^0 \#$; $[a]_e^0 \to [\ ]_e^- \#$.

(*e*) $[q_0 \to q_1]_e^-$; $[q_1 \to q_0]_e^0$.
$[q_0]_e^0 \to No_0[\ ]_e^+$.
$[q_1 \to q_2 c]_e^-$; $[q_2 \to q_3]_e^0$; $[c]_e^- \to k[\ ]_e^0$.
$[q_3]_e^0 \to Yes[\ ]_e^+$; $[q_3]_e^- \to No_0[\ ]_e^+$.

(fase de respuesta)

(*g*) $[z_i \to z_{i+1}]_s^0$, para $i = 0, \ldots, 2n$; $[z_{2n+1} \to d_0 d_1]_s^0$.
$d_0[\ ]_r^0 \to [d_0]_r^-$; $[d_1]_s^0 \to [\ ]_s^+ d_1$.

(*det*) (bucle detector)

$[h_0 \to h_1]_r^-$, $[h_1 \to h_0]_r^+$,
$[b]_r^- \to [\ ]_r^+ b$, $\hat{g}[\ ]_r^+ \to [\hat{g}]_r^-$,
$b[\ ]_r^- \to [b]_r^+$, $[\hat{g}]_r^+ \to [\ ]_r^- \hat{g}$,
$[h_0]_r^+ \to [\ ]_r^+ d_2$, $[d_2]_s^+ \to [\ ]_s^- d_2$.

(*h*) $[No_0 \to No]_s^-$; $[Yes]_s^- \to [\ ]_s^0 Yes$; $[No]_s^- \to [\ ]_s^0 No$.

En esta solución, una instancia $u = (n, (w_1, \ldots, w_n), k)$ es procesada por el sistema $\Pi_2(n)$, proporcionándole el multiconjunto de entrada $x_1^{w_1} \ldots x_n^{w_n}$.

Este segundo diseño sólo depende de una de las constantes que son dadas como dato de entrada del problema: n. Es bastante similar al anterior, la diferencia está en las fases de chequeo y de respuesta. En lugar de utilizar un contador cuyo rango de índices dependía de k, se han utilizado estrategias de detección de final de una fase mediante marcadores (apartados (*e*) y (*det*)) que requieren únicamente una cantidad constante de reglas.

El número total de reglas de evolución en el sistema es $5n + 40$, y el número de pasos que dura cada computación depende de la instancia concreta que estemos resolviendo, aunque ese número está acotado linealmente.

Se trata de resolver la instancia $u = (5, (3, 5, 3, 2, 5), 9)$ del problema Subset Sum. El sistema celular que se encarga de resolverlo, $\Pi_2(5)$, consta de 65 reglas de evolución, y si le proporcionamos la entrada adecuada, $cod(u) = x_1^3\, x_2^5\, x_3^3\, x_4^2\, x_5^5$, entonces el sistema resuelve la instancia en 38 pasos celulares, expulsando un objeto No.

Presentamos una expresión gráfica de la Sevilla Carpet asociada a dicha computación en la Figura 9.2.

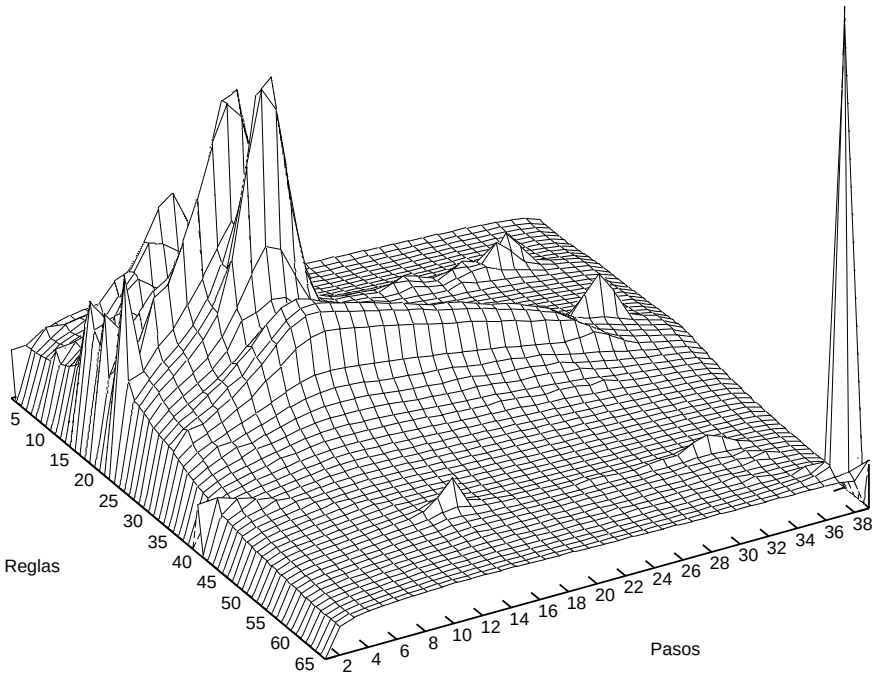

Figura 9.2: Computación de $\Pi_2(5)$ con entrada $cod(u) = x_1^3\, x_2^2\, x_3^7\, x_4^3\, x_5^4$

A continuación, como se hizo con el primer diseño, vamos a presentar los valores de ciertos parámetros de complejidad descriptiva asociados a esta computación, lo que nos permitirá comprobar las diferencias entre los dos diseños. Todas las reglas de evolución son aplicadas en algún momento de la computación, con la excepción de las siguientes reglas: $[q_3]_e^0 \to Yes[\ ]_e^+$ y $[Yes]_s^- \to [\ ]_s^0 Yes$.

El *peso* de la Sevilla Carpet (el número total de aplicaciones de reglas durante la computación) es 3368. La *altura* de la Sevilla Carpet (el máximo número de veces que se aplica una regla en un paso de evolución) se alcanza en el paso 10, cuando la regla $[\bar{a}_0 \to \lambda]_e^-$ es aplicada 108 veces. La *superficie* de la Sevilla Carpet es 2470, y la *altura media* es 1'36275.

A grandes rasgos, la apariencia visual de la gráfica en este caso es muy similar a la de la solución anterior. Es decir, los mayores "picos" se concentran en las reglas de los primeros grupos y en los primeros pasos de computación. Podemos observar cómo se distingue en este caso un pico aislado casi en el último paso de computación: se corresponde a la aplicación de la regla $[No_0 \rightarrow No]_s^-$, que se aplica sobre 2^5 objetos No_0 en la piel (ya que la instancia considerada tiene solución negativa y, por tanto, todas las membranas relevantes envían un objeto No_0 a la piel al final de su fase de chequeo).

Conviene aclarar que la apariencia visual de la Sevilla Carpet debe ser considerada como una información orientativa, ya que los datos no aparecen reflejados con toda fidelidad en la gráfica, puesto que son afectados por el programa gráfico, que trata de "suavizar" la superficie al representarla.

9.4.3. Comparación de las soluciones

En la siguiente tabla se contrastan los valores de los parámetros asociados a las dos soluciones:

	Solución 1	Solución 2
Reglas	88	65
Pasos	33	38
Superficie	2904	2470
Peso	2179	3368
Altura	82	108
Altura media	0'749656	1'36275

Tabla 9.1: Comparación de las dos soluciones celulares del problema Subset Sum

Si considerásemos el número de pasos como única medida de complejidad para comparar ambos diseños, entonces tendríamos que concluir que la primera solución es "mejor" que la segunda (aunque no de forma asintótica).

Obtenemos idéntica conclusión si nos guiamos por el peso de las Sevilla Carpets asociadas: de nuevo la primera solución resulta ser la "mejor", la que menos recursos utiliza durante la computación. Sin embargo, el hecho de que la altura media sea mayor para el segundo diseño puede interpretarse en el sentido de que dicho diseño aprovecha más el paralelismo del sistema (la computación es más "intensa").

La explicación de estas diferencias está en el uso de contadores en el primer diseño. Mediante la introducción de contadores se obtiene un gran control sobre la com-

putación, pero esto tiene sus inconvenientes. En primer lugar, para determinar el rango en el que varían los índices de cada contador es necesario hacer un estudio formal detallado de la computación. Además, el uso de contadores suele conllevar la introducción de muchas reglas de evolución en el diseño (una para cada índice).

Sin embargo, al utilizar estrategias con marcadores para detectar cuándo terminan los bucles (de chequeo o de detección) se requieren pocas reglas de evolución, pero éstas serán aplicadas más veces (una regla de evolución asociada a un contador sólo se aplica *una* vez durante toda la computación).

9.5. El problema Knapsack

A continuación, vamos a comentar con algo de detalle la complejidad de las computaciones que realizan los sistemas celulares de la familia diseñada en la Sección 5.2 para resolver el problema Knapsack.

Consideremos, por ejemplo, la instancia $u = (5, (2, 4, 3, 5, 3), (2, 1, 2, 3, 2), 9, 6)$ de dicho problema. El sistema celular que se encarga de resolverlo, $\Pi(9875) = \Pi(\langle 5, 9, 6\rangle)$, consta de 133 reglas, y si le proporcionamos la entrada adecuada, $cod(u) = x_1^2\, x_2^4\, x_3^3\, x_4^5\, x_5^3\, y_1^2\, y_2^1\, y_3^2\, y_4^3\, y_5^2$, entonces el sistema resuelve la instancia en $2n + 2k + 2c + 9 = 49$ pasos celulares, expulsando un objeto Yes.

Presentamos de manera gráfica la Sevilla Carpet asociada a dicha solución en la Figura 9.3, para ilustrar la complejidad descriptiva de la computación. Asimismo, en la Tabla 9.2 se muestran los correspondientes valores de los parámetros introducidos en la Sección 9.3.

Reglas	133
Pasos	49
Superficie	6517
Peso	3728
Altura	74
Altura media	0'571735

Tabla 9.2: Parámetros para el problema Knapsack

Todas las reglas de evolución son aplicadas en algún momento de la computación, con la excepción de las siguientes reglas: $[q_3]_e^- \to [\ ]_e^+\#$, $[q_{13}]_e^0 \to [\ ]_e^0 Yes$ y $[No]_s^+ \to [\ ]_s^0 No$.

El *peso* de la Sevilla Carpet (el número total de aplicaciones de reglas durante la computación) es 3728. La *altura* de la Sevilla Carpet (el máximo número de veces

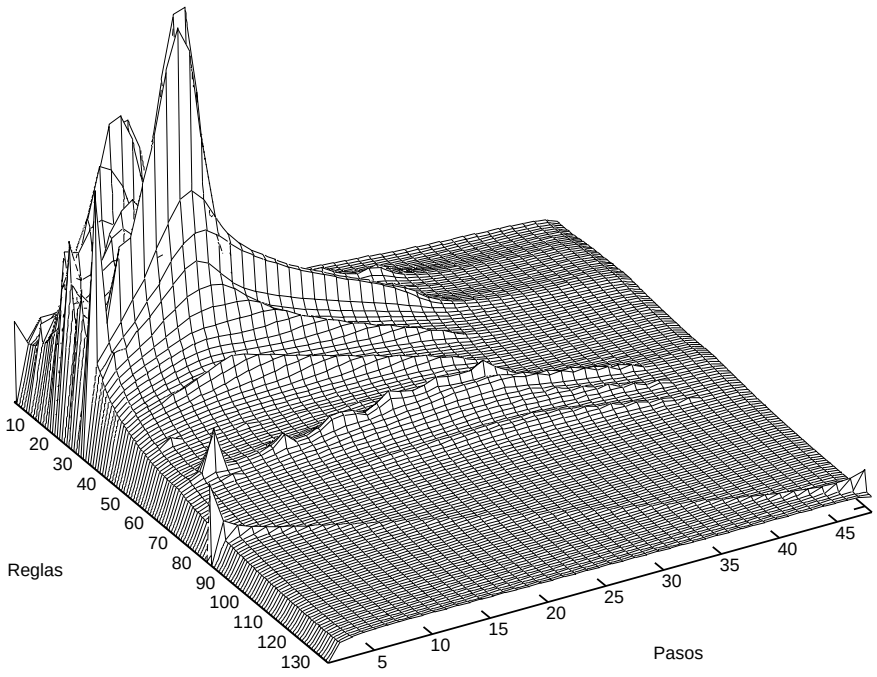

Figura 9.3: Descripción de la evolución del sistema celular que resuelve la instancia $u = (5, (2, 4, 3, 5, 3), (2, 1, 2, 3, 2), 9, 6)$ del problema Knapsack

que se aplica una regla en un paso de evolución) se alcanza en el paso 9, cuando la regla $[\bar{a}_0 \rightarrow a_0]_e^-$ es aplicada 74 veces. La *superficie* de la Sevilla Carpet es 6517, y la *altura media* es 0'571735.

9.6. El problema de la Partición

A continuación, vamos a comentar con algo de detalle la complejidad de las computaciones que realizan los sistemas celulares de la familia diseñada en la Sección 6.2 para resolver el problema de la Partición.

Consideremos, por ejemplo, la instancia $u = (5, (5, 4, 1, 8, 6))$ del problema de la Partición. El sistema celular que se encarga de resolverlo, $\Pi(5)$, consta de 67 reglas, y si le proporcionamos la entrada adecuada, $cod(u) = x_1^5\, x_2^4\, x_3^1\, x_4^8\, x_5^6$, entonces el sistema resuelve la instancia en 40 pasos celulares (recordemos que la cota superior es $2n + w(A) + 11 = 45$ pasos), expulsando un objeto Yes.

Presentamos de manera gráfica la Sevilla Carpet asociada a dicha solución en la Figura 9.4, para ilustrar la complejidad descriptiva de la computación.

A continuación presentamos en la Tabla 9.3 los valores de los parámetros asociados a la correspondiente computación.

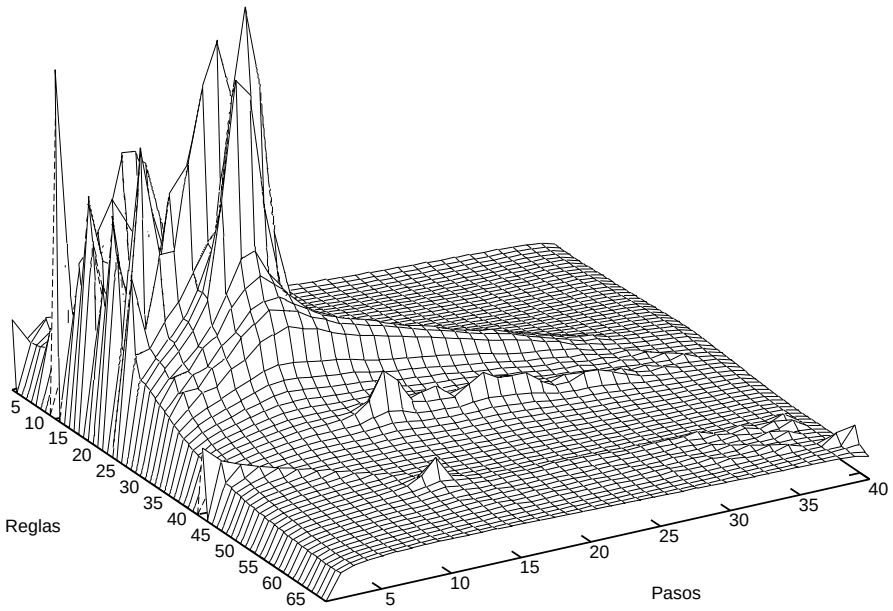

Figura 9.4: Computación de $\Pi(5)$ con entrada $cod(u) = x_1^5\, x_2^4\, x_3^1\, x_4^8\, x_5^6$

Reglas	67
Pasos	40
Superficie	2680
Peso	2065
Altura	87
Altura media	0'76791

Tabla 9.3: Parámetros para el problema Partición

Todas las reglas de evolución son aplicadas en algún momento de la computación, con la excepción de las siguientes reglas: $[x_1 \to p_0]_e^-$ y $[No]_s^- \to [\ ]_s^0 No$.

El *peso* de la Sevilla Carpet (el número total de aplicaciones de reglas durante la computación) es 2065. La *altura* de la Sevilla Carpet (el máximo número de veces que se aplica una regla en un paso de evolución) se alcanza en el paso 8, cuando la regla $[a_0 \to \#]_s^0$ es aplicada 87 veces. La *superficie* de la Sevilla Carpet es 2680, y la *altura media* es 0'76791.

Conclusiones y trabajo futuro

Finalizamos esta memoria presentando ciertas conclusiones que se pueden extraer de la misma, y algunos caminos a seguir como trabajo futuro.

Conclusiones

La Computación Celular con Membranas es una disciplina joven aunque emergente dentro del campo de los modelos de computación no convencionales, en este caso inspirados en procesos dinámicos que se dan en la Naturaleza viva.

Esta rama de la Computación Natural ha tenido un crecimiento espectacular en los últimos años y ha sido considerada en octubre de 2003 por el Institute for Scientific Information (ISI) como *Fast Emerging Research Front*. Fue inciada a finales de 1998 por Gheorghe Păun mediante un trabajo titulado *Computing with membranes* que circuló por la red y que sería posteriormente publicado en el año 2000 en la revista *Journal of Computer and System Sciences* (este trabajo ha sido considerado por el ISI como uno *de los trabajos recientes más citados* y considerado en febrero de 2003 como el *Fast Breaking Paper* en *Computer Science*).

Como consecuencia de ser una disciplina reciente, existen muchas cuestiones pendientes para abordar y tareas a realizar. En esta memoria hemos realizado una aportación en una serie de aspectos que consideramos importantes en el ámbito de los modelos de computación no convencionales.

En primer lugar, nuestro trabajo ha estado centrado en el estudio de la *eficiencia computacional* de algunas variantes de sistemas celulares que poseen la capacidad de *generar* de manera natural un espacio de trabajo de tamaño exponencial en tiempo polinomial, de tal manera que pueden ser dispositivos útiles a la hora de resolver instancias *razonablemente grandes* de problemas presuntamente intratables. Por ello, nos hemos centrado en la obtención de soluciones eficientes de problemas **NP**–completos (concretamente, problemas numéricos) en este nuevo marco.

Puntualicemos el significado de *soluciones eficientes* en el marco de las clases de complejidad introducidas en el Capítulo 2 de esta memoria. En nuestro caso, el *coste*

de la computación no se mide según los cánones de la computación paralela tradicional, en la que hay que tener en cuenta el número de procesadores que intervienen en el proceso. Este dato se obvia en el caso de los sistemas celulares, porque la capacidad de hacer aparecer nuevas membranas en el sistema, en principio sin *coste* alguno, es una habilidad que se le presupone al sistema, y que tiene su inspiración en la división celular (una célula puede dividirse siempre que haya suficientes nutrientes disponibles).

En este contexto se ha elegido el problema Knapsack, de gran relevancia en criptografía, y otros dos problemas relacionados directamente con éste: el problema Subset Sum y el de Partición. Para todos ellos se ha seguido el mismo proceso para justificar que son problemas *tratables* en el marco de los sistemas reconocedores con membranas activas. En primer lugar, se ha diseñado una familia de sistemas celulares y se han considerado dos funciones computables en tiempo polinomial tales que para cada instancia del problema, una de ellas (la función s) codifica el índice correspondiente del sistema de la familia que se encargará de procesarla, y la otra (la función *cod*) proporciona el multiconjunto de entrada necesario para el procesamiento de la instancia por dicho sistema.

Además, los diseños propuestos en esta memoria pretenden ser tan generales como sea posible y, al mismo tiempo, se intenta que sean *uniformes*, en el siguiente sentido: el diseño de una familia de sistemas P no se realiza asociando un sistema a cada instancia del problema. En lugar de eso, cada sistema celular de la familia procesará un conjunto de instancias del *mismo tamaño*. En nuestro caso, el esquema de reglas para los sistemas de la familia se hace en función de los valores de las constantes n, k y c, aunque en el Capítulo 7, tras introducir las posibles subrutinas para resolver problemas numéricos, se muestran diseños para los problemas Subset Sum, Knapsack y Partición donde las reglas sólo dependen de n.

También hay que tener en consideración que el número de pasos celulares de las computaciones es lineal en el tamaño del dato de entrada (que está codificado de forma 1-aria en el multiconjunto de entrada). Esta aceleración se ha conseguido mediante el uso de reglas de división de membranas elementales (y considerando membranas cargadas eléctricamente, que ayudan a sincronizar las distintas etapas de la computación). Sería interesante plantearse la búsqueda de modelos equivalentes reduciendo en lo posible la cantidad de especificaciones del modelo (por ejemplo ¿se mantiene la potencia computacional si consideramos sistemas P con membranas activas donde las membranas no tengan asociadas cargas eléctricas?).

A la hora de buscar nuevas variantes que sean equivalentes, una de las ideas que hay que tener presente es la aproximación a la realidad biológica. Aunque los modelos

de computación celular aún no han sido implementados en laboratorios (ni en ningún otro medio físico), existen investigaciones encaminadas en esa dirección, tanto *in vitro* (creando las membranas artificialmente como bi–capas de fosfolípidos) como *in vivo* (llevando la computación a una colonia de bacterias o de otros seres unicelulares).

La verificación de la familia construida respecto del problema de decisión que trata de resolver se ha realizado de acuerdo con el concepto de *resolubilidad en tiempo polinomial* definido de manera natural y siendo consecuente con la noción de clase de complejidad en modelos convencionales. Estos procesos de verificación son extraordinariamente laboriosos debido a la propia naturaleza del modelo con el que se trabaja, que está *orientado a máquinas*.

Tanto en el diseño de las soluciones como en los procesos de verificación formal, se han encontrado una serie de analogías o similitudes que no pueden ser pasadas por alto.

Por una parte, a la hora de implementar en sistemas celulares procedimientos de *fuerza bruta* para resolver un problema **NP**–completo, suele existir una fase de generación, seguida de una fase de chequeo y de una fase de respuesta (si bien algunas de estas fases pueden ser realizadas de manera simultánea). En cada una de estas fases se pueden considerar grupos de reglas de evolución que, en cierto sentido, se encargan de ejecutar tareas específicas y que, por tanto, pueden ser tratadas de manera independiente, a modo de las *macros* de un modelo de computación orientado a programas. Esto nos lleva a pensar en la posibilidad de considerar una especie de *lenguaje de programación celular* que sirva de ayuda para el diseño de familias de sistemas con membranas que resuelven problemas (tarea ésta que es muy compleja en modelos orientados a máquinas), al menos, para cierta clase de problemas.

Por otra parte, en los procesos de verificación se han detectado otras analogías que pueden representar un pequeño avance en el desarrollo de una metodología para atacar dichos procesos. Para ello, nos hemos basado en dos características fundamentales de los sistemas construidos:

- Toda computación del sistema está estructurada en varias fases, cada una de las cuales se encarga de realizar una tarea específica.
- El comienzo y el final de estas fases se pueden identificar por la presencia de ciertos objetos en determinadas membranas.

De esta forma, para justificar la corrección del proceso efectuado por estos sistemas, el procedimiento a seguir ha consistido en estudiar por separado el buen comportamiento de cada una de las fases en las distintas computaciones, para, finalmente, concluir con la corrección global de la familia. Desde luego, como hemos observado anteriormente,

dicha metodología sólo sería aplicable a cierta clase de problemas de decisión.

En relación con esto, es destacable la utilidad que ha demostrado tener el simulador escrito en Prolog que se presenta en el Capítulo 8. Esta herramienta informática es de gran ayuda en la tarea de depurar errores en el diseño y en la verificación de familias de sistemas P, ya que nos permite llevar a cabo estudios detallados de la evolución del sistema comprobando, paso a paso, cómo se aplican las reglas. También ha existido un proceso de retroalimentación, puesto que la simulación de sistemas ya verificados ha permitido detectar y subsanar algunas pequeñas irregularidades en la programación del simulador.

Además, se han diseñado una serie de módulos (ficheros Prolog) que permiten al simulador generar los conjuntos de reglas y las configuraciones iniciales correspondientes a los diseños presentados en esta memoria. Una posible línea de trabajo futuro consistiría en unificar dichos módulos siguiendo la iniciativa de un lenguaje de programación celular presentada en el Capítulo 7, de manera que el simulador permitiera al usuario introducir "programas celulares" y fuese capaz de traducir las instrucciones de dichos programas a una lista de reglas (a una especie de *lenguaje máquina*).

Por último, mencionemos que en esta memoria se ha estudiado la complejidad descriptiva de los sistemas celulares, introduciendo una representación gráfica de dicha complejidad (las *Sevilla Carpets*) junto con algunos parámetros significativos. Sin embargo, queda pendiente la definición de nuevos parámetros de complejidad que complementen la descripción de las computaciones y que permitan, por ejemplo, comparar distintas soluciones celulares para un mismo problema.

En el ejemplo presentado en el Capítulo 9, se confrontan dos soluciones para el problema Subset Sum y se compara su ejecución, aunque no se extraen conclusiones generales, dado que sólo se evalúa la complejidad de las computaciones asociadas a *una* instancia del problema (además, el tamaño de la instancia no es lo suficientemente grande como para acentuar las diferencias entre un diseño con $5n + 5k + 18$ reglas de evolución y otro con $5n + 40$ reglas). Por otra parte, es claro que para recabar toda la información necesaria para representar la Sevilla Carpet (número de veces que se aplica *cada* regla en *cada* paso de la computación) es necesario disponer de una herramienta informática que simule la computación. En nuestro caso se ha utilizado precisamente el simulador Prolog presentado en el Capítulo 8, pero esto impone una restricción sobre el tamaño de las instancias que se pueden estudiar (el simulador solo permite resolver instancias pequeñas sin bloquearse).

Trabajo futuro

Creemos que la investigación desarrollada en esta memoria debe tener una continuación en una serie de puntos que remarcamos como objetivos de posibles trabajos futuros, algunos de los cuales ya están siendo abordados en colaboración con otros miembros del *Grupo de Investigación en Computación Natural* de la Universidad de Sevilla.

1. Perfeccionamiento de la metodología seguida para la verificación formal de familias de sistemas reconocedores con membranas activas.

2. Definición del concepto de reducibilidad en tiempo polinomial mediante sistemas de computación celular con membranas, que permitan usar sólo conceptos relativos a sistemas de membranas.

3. Conversión de la fase de pre-computación (secuencial, computación clásica), a la que se hace referencia en la definición de las clases de complejidad $\mathbf{PMC}_{\mathcal{R}}$ (en la Sección 2.6), en términos de sistemas celulares.

4. Estudio de las clases de complejidad determinadas por otras variantes de sistemas celulares, tipo *tissue* con división de membranas.

5. Resolución, en tiempo polinomial, de problemas **NP**–completos mediante un solo sistema reconocedor con membranas activas.

6. Descripción de clases de complejidad clásicas a través de clases de complejidad polinomial asociadas a ciertos tipos de sistemas P. Por ejemplo: Encontrar una clase $\mathcal{F}$ de sistemas celulares reconocedores tal que se verifique $\mathbf{NP} \cup \mathbf{co}-\mathbf{NP} = \mathbf{PMC}_{\mathcal{F}}$, o bien $\mathbf{PSPACE} = \mathbf{PMC}_{\mathcal{F}}$.

7. Estudio detallado, desde el punto de vista de la eficiencia computacional, de la variante de sistemas reconocedores con membranas activas que no usan cooperación, ni prioridad, ni cambio de etiquetas ni polarización.

8. Análisis de estrategias de búsqueda para seleccionar una *buena* computación a la hora de *decidir* una instancia de un problema **NP**–completo, a través de sistemas celulares reconocedores.

9. Estudio de otras nociones de complejidad para sistemas P no deterministas (medidas de fiabilidad, entropía).

10. Extensión del lenguaje de programación celular definiendo nuevas instrucciones y extendiendo el ámbito de aplicación del mismo.

11. Elaboración de un interfaz que haga amigable la interacción del usuario con el simulador Prolog desarrollado.

12. Uso del Prolog paralelo para mejorar las prestaciones del simulador de sistemas celulares presentado en esta memoria.

13. Estudio de nuevos parámetros relativos a la complejidad descriptiva de sistemas celulares, que complementen los resultados que se obtienen a través de la complejidad computacional.

14. Aplicación de los sistemas celulares al estudio y análisis de la dinámica de poblaciones y los sistemas complejos.

Bibliografía

[1] Adleman, L.M. Molecular computations of solutions to combinatorial problems, *Science*, **226**, 1021–1024.

[2] Alhazov, A.; Freund, R. y Păun, Gh. P systems with active membranes and two polarizations, en Gh. Păun, A. Riscos Núñez, A. Romero Jiménez y F. Sancho Caparrini (eds.) *Proceedings of the 2nd Brainstorming Week on Membrane Computing*, TR 01/2004, RGNC, Universidad de Sevilla, 20–36.

[3] Ardelean, I.I.; Cavaliere, M. y Sburlan, D. Computing using signals: from cells to P systems, en Gh. Păun, A. Riscos Núñez, A. Romero Jiménez y F. Sancho Caparrini (eds.) *Proceedings of the 2nd Brainstorming Week on Membrane Computing*, TR 01/2004, RGNC, Universidad de Sevilla, 60–73.

[4] Atanasiu, A. y Martín-Vide, C. *P systems and arithmetical calculus*, TR 14/00, RGML, Universitat Rovira i Virgili, 2000.

[5] Bennet, C.H. Logical reversibility of computation, *IAM Journal of Research and Development*, **17**, 1973, 525–532.

[6] Bennet, C.H. Thermodynamics of computation: a review, *International Journal of Theoretical Physics*, **21**, 1982, 905–940.

[7] Bernardini, F. y Păun, A. Universality of minimal symport/antiport: Five membrane suffice, en C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg y A. Salomaa (eds.), *Membrane Computing*. Lecture Notes in Computer Science, **2933**, 2004, 43–54.

[8] Bessozi, D.; Ardelean, I.I. y Mauri, G. The potential of P systems for modelling the activity of mechanosensitive channels in E. Coli, en A. Alhazov, C. Martín-Vide y Gh. Păun (eds). *Proceedings of the Workshop on Membrane Computing*, TR 28/ 03, RGML, Universitat Rovira i Virgili, 2003, 84–102.

[9] Beyer, H.-G. *The Theory of Evolution Strategies*. Springer, Berlin, 2001.

[10] Box, G.E.P. Evolutionary operation: A method of increasing industrial productivity, *Applied Statistics*, **6**, 1957, 81–101.

[11] Bratko, I. *PROLOG Programming for Artificial Intelligence*, Third edition. Addison-Wesley, 2001.

[12] Ciobanu, G.; Păun, Gh. y Stefănescu, Gh. Sevilla carpets associated with P systems, en M. Cavaliere, C. Martín-Vide and Gh. Păun (eds.), *Proceedings of the Brainstorming Week on Membrane Computing*, TR 26/03, RGML, Universitat Rovira i Virgili, 2003, 135-140.

[13] Cobham, A. The intrinsec computational difficulty of functions, *Proceedings of the 1964 International Congress of Logic, Methodology, and Phylosophy of Science*. Elsevier/North-Holland, 1964, 24–30.

[14] Cordón Franco, A.; Gutiérrez Naranjo, M.A.; Pérez Jiménez, M.J. y Sancho Caparrini, F. A Prolog simulator for deterministic P systems with active membranes, en M. Cavaliere, C. Martín-Vide, y Gh. Păun (eds.) *Proceedings of the Brainstorming Week on Membrane Computing*, TR 26/03, RGML, Universitat Rovira i Virgili, 2003, 141–154.

[15] Cordón Franco, A.; Gutiérrez Naranjo, M.A.; Pérez Jiménez, M.J. y Sancho Caparrini, F. A Prolog simulator for deterministic P systems with active membranes, *New Generation Computing*, **22** (4), 2004, 349–363.

[16] Davis, M.D.; Sigal, R. y Weyuker, E.J. *Computability, Complexity and Languages: Fundamentals of Theoretical Computer Science*, Academic Press, 1994.

[17] Edmonds, J. Minimum partition of a matroid into independent subsets. *Journal Res. Nat. Bur. Standards* Sect. B, **69**, 1965, 67–72.

[18] Feyman, R.P. There's plenty of room at the bottom, en D.H. Gilbert (ed) *Miniaturization*, Reinhold, New York, 1961, 282–296.

[19] Fogel, L.; Owens, A. y Walsh, H. *Artificial Intelligence through simulated evolution*, Wiley, New York, 1966.

[20] Fraser, A. Simulation of genetic systems by automatic digital computers I: Introduction. *Australian Journal of Biological Science*, **10**, 1957, 484–491.

[21] Frisco, P. *Theory of Molecular Computing. Splicing and Membrane Computing.* Ph.D. Thesis, IPA (Institute voor Programaturkunde en Algorithmiek), Leiden, 2004.

[22] Gutiérrez Naranjo M.A.; Pérez Jiménez, M.J. y Riscos Núñez, A. Towards a programming language in cellular computing, en Gh. Păun, A. Riscos Núñez, A. Romero Jiménez y F. Sancho Caparrini (eds.) *Proceedings of the 2nd Brainstorming Week on Membrane Computing*, TR 01/2004, RGNC, Universidad de Sevilla, 247–257. Una versión revisada de este trabajo ha sido presentada en el WoLLIC'2004, Université de Paris XII, del 19 al 22 de Julio, París, Francia.

[23] Head, T.J. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, **49**, 1987, 737–759.

[24] Holland, J.H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[25] Ibarra, O.H. The number of membranes matters, en A. Alhazov, C. Martín-Vide y Gh. Păun (eds). *Preproceedings of the Workshop on Membrane Computing*, TR 28/ 03, RGML, Universitat Rovira i Virgili, 2003, 273–285.

[26] Krishna S.N. *Languages of P Systems: Computability and Complexity*, Ph.D. Thesis, Indian Institute of Technology Madras, 2001.

[27] Krishna, S.N.; Krithivasan, K. y Rama, R. P systems with picture objects, *Acta Cybernetica*, **15**, (1), 2001, 53–74.

[28] Mäkinen, E. A Bibliography on Szilard Languages, Dept. of Computer and Information Sciences, University of Tampere, `http://www.cs.uta.fi/reports/pdf/Szilard.pdf`

[29] Martín-Vide, C.; Păun, A. y Păun, Gh. On the power of P systems with symport rules, *Journal of Universal Computer Science*, **8** (2), 2002, 317–331.

[30] Martín-Vide, C.; Păun, A.; Păun, Gh. y Rozenberg, G: Membrane systems with coupled transport: universality and normal forms, *Fundamenta Informaticae*, **49** (1–3), 2002, 1–15.

[31] Martín-Vide, C. y Păun, Gh. *Elements of Formal Language Theory for Membrane Computing*, TR 21/01, RGML, Universitat Rovira i Virgili, 2001.

[32] Mateescu, A. y Salomaa, A. Aspects of Classical Language Theory, en G. Rozenberg y A. Salomaa (eds.) *Handbook of Formal Languages* (vol. 1), Springer-Verlag, Berlin Heidelberg, 1997.

[33] McCulloch, W.S. y Pitts, W. A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, **5**, 1943, 115–133.

[34] Mitchell, T.M. *Machine Learning.* McGraw-Hill, 1997.

[35] Pan, L. y Martín-Vide, C. Solving Multidimensional 0-1 Knapsack Problem by P Systems with Input and Active Membranes, en Gh. Păun, A. Riscos Núñez, A. Romero Jiménez, F. Sancho Caparrini (eds.) *Proceedings of the 2nd Brainstorming Week on Membrane Computing*, TR 01/2004, RGNC, Universidad de Sevilla, 342–353.

[36] Păun, A. y Păun, Gh. The power of the communication: P systems with symport/ antiport, *New Generation Computing*, **20** (3), 2002, 295–306.

[37] Păun, Gh. Computing with membranes, *Journal of Computer and System Sciences, **61**(1)*, 2000, 108–143.

[38] Păun, Gh. P Systems with active membranes: attacking NP complete problems, *Journal of Automata, Languages and Combinatorics* **6** (1), 2001, 75–90.

[39] Păun, Gh. *Membrane Computing. An introduction.* Springer-Verlag, Berlin, 2002.

[40] Păun, Gh. y Rozenberg, G. A guide to membrane computing, *Theoretical Computer Sciences*, **287**, 2002, 73–100.

[41] Pérez Jiménez, M.J. y Romero Campero, F.J. Solving the BINPACKING problem by recognizer P systems with active membranes, en Gh. Păun, A. Riscos Núñez, A. Romero Jiménez y F. Sancho Caparrini (eds.) *Proceedings of the 2nd Brainstorming Week on Membrane Computing*, TR 01/2004, RGNC, Universidad de Sevilla, 414–430.

[42] Pérez Jiménez, M.J.; Romero Jiménez, A. y Sancho Caparrini, F. A polynomial complexity class in P systems using membrane division, *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems*, Budapest, Hungría, 2003.

[43] Pérez Jiménez, M.J.; Romero Jiménez, A. y Sancho Caparrini, F. *Teoría de la Complejidad en modelos de computación celular con membranas.* Ed. Kronos, Sevilla, 2002.

[44] Pérez Jiménez, M.J.; Romero Jiménez, A. y Sancho Caparrini, F. Complexity classes in P systems, *Meeting of the European Molecular Computing Consortium. Volume of abstracts*, 2003.

[45] Pérez Jiménez, M.J.; Romero Jiménez, A. y Sancho Caparrini, F. Complexity classes in cellular computing with membranes, en M. Cavaliere, C. Martín-Vide, y Gh. Păun (eds.), *Proceedings of the Brainstorming Week on Membrane Computing*, TR 26/03, RGML, Universitat Rovira i Virgili, 2003, 270–278.

[46] Pérez Jiménez, M.J.; Romero Jiménez, A. y Sancho Caparrini, F. Solving Validity problem by active membranes with input, en M. Cavaliere, C. Martín-Vide, y Gh. Păun (eds.), *Proceedings of the Brainstorming Week on Membrane Computing*, TR 26/03, RGML, Universitat Rovira i Virgili, 2003, 279–290.

[47] Pérez Jiménez, M.J. y Sancho Caparrini, F. A formalization of transition P systems, *Fundamenta Informaticae*, **49**(1–3), 2002, 261–272.

[48] Rozenberg, G. y Salomaa, A. (eds.), *Handbook of Formal Languages*, Springer-Verlag, Berlin Heidelberg, 1997.

[49] Rumelhart, D.; Widrow, B. y Lehr, M. The basic ideas in neural networks, *Communications of the ACM*, **37** (3), 1994, 87–92.

[50] Salomaa, A. *Formal Languages*. Academic Press, New York, 1973.

[51] Sancho Caparrini, F. *Verificación de programas en modelos de computación no convencionales*. Ph.D. Thesis, Universidad de Sevilla, 2002.

[52] Von Neumann, J. A certain zero-sum two-person game equivalent to the optimal asignment problem, en H.W. Kahn y A.W. Tucker (eds.) *Contributions to the Theory of Games II*. Princenton Univ. Press, 1953.

[53] Zandron, C. *A Model for Molecular Computing: Membrane Systems*. Ph.D. Thesis, Università degli Studi di Milano, 2001.

[54] Zandron, C.; Mauri, G. y Ferretti, C. Solving NP–complete problems using P systems with active membranes, en I. Antoniou, C.S. Calude y M.J. Dinneen (eds), *Unconventional Models of Computation*, Springer-Verlag, 2000, 289–301.

[55] The P Systems Web Page: `http://psystems.disco.unimib.it/`

[56] Logic Programming: `http://www.afm.sbu.ac.uk/logic-prog/`