Verificación de programas en modelos de computación no convencionales

Memoria presentada por
Fernando Sancho Caparrini
para optar al grado de
Doctor en Matemáticas
por la Universidad de Sevilla

Fernando Sancho Caparrini

V. B. Director

D. Mario de J. Pérez Jiménez

Sevilla, Abril de 2002

A mis padres,

Agradecimientos

En primer lugar, querría que estos agradecimientos se pudiesen leer en paralelo, sin el orden secuencial en el que están escritos. De esa forma reflejarían más fielmente su significado.

Quiero expresar mi más sincero agradecimiento a Mario, que confió en mí cuando no había razones objetivas para hacerlo. A su lado he aprendido a trabajar e investigar y, sin duda, es una enseñanza que marcará mi futuro. Espero algún día poder devolverle el esfuerzo que ha realizado.

Al Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla (en pleno, sin detallarlos), por su calurosa acogida (incluso con mis rarezas). Muy especialmente a Andrés, que me aguanta cada día con su mejor cara.

Al Grupo de Investigación en Computación Natural de la Universidad de Sevilla. Formado recientemente, pero bien asentado y con un prometedor futuro. Esta memoria puede considerarse como la primera de las que saldrán del trabajo en equipo que en él se desarrolla.

Por supuesto, no puedo pasar por alto los años que disfruté en el Dpto. de Análisis de la Universidad de Sevilla. También de ellos es parte de esta memoria, porque tras dejar de formar parte de un mismo equipo me hacen sentir que todavía hay lazos que nos unen (y ni siquiera nombro al GIMP, que todavía me deben una visita desinteresada).

A mi familia, por su paciencia (llevaban tiempo esperándolo, y aquí está).

A Juan Carlos, porque me ayudó a ver la belleza de las matemáticas. A Elena, para demostrarle que todo es posible, y que aún nos queda una. A Ana, porque con ella aprendí a razonar las matemáticas. A Lucía, porque con ella aprendí a valorar el esfuerzo que hay que invertir. Con Chary, porque con ella sigo aprendiendo.

Y finalmente a los buenos amigos (el grupo del ruso blanco, los zaragozanos,etc.).

Abril, 2002

Índice general

Introducción

La Ciencia, a lo largo de su historia, ha experimentado una serie de sobresaltos motivados por la consecución de unos resultados que, unas veces, han culminado un proceso previo, generalmente largo, de gestación y, otras, han provocado una auténtica convulsión entre los investigadores. En todo caso, esos hitos han producido un avance cualitativo en el conocimiento científico.

En la historia de la *Computación* se pueden observar unos puntos de inflexión que han marcado de forma decisiva el devenir de la misma, y han propiciado el desarrollo de nuevas herramientas, tanto a nivel teórico como a nivel práctico, que han sido consideradas fundamentales. Curiosamente, la mayoría de esos momentos críticos están asociados con las *limitaciones* inherentes a la potencia de determinados métodos y/o modelos.

El primer hito relevante que queremos destacar se produce con la respuesta negativa a dos cuestiones planteadas por D. Hilbert, en 1928, relativas a la *consistencia* y *completitud* de las Matemáticas, en donde se establecen las *limitaciones* inherentes al *método axiomático*. La *Teoría de la Computación* como disciplina de estudio comienza, propiamente, con los trabajos realizados para dar respuesta a estas cuestiones.

La posibilidad de que existan problemas abstractos que no se puedan resolver de *manera efectiva* y la necesidad de dar respuesta a la tercera cuestión formulada por D. Hilbert, en 1928, relativa a la *decidibilidad* de las Matemáticas, plantea la conveniencia de formalizar el concepto de *algoritmo* y de *función computable*. Así, en la década de los treinta, aparecen los primeros modelos teóricos de computación y se inicia el estudio formal del concepto de procedimiento *efectivo*. La existencia de problemas que no pueden ser resueltos mediante *procedimientos mecánicos* en los modelos diseñados (por ejemplo, la *indecidibilidad de la lógica de primer orden* o el *problema de la parada*) proporciona una *limitación* importante al método de formalización del concepto de algoritmo.

En la década de los cincuenta aparecen los primeros ordenadores electrónicos y, con ellos, surge lo que podríamos denominar informalmente como *modelos de*

computación práctica. En ellos se pueden implementar los algoritmos diseñados, mediante un proceso de traducción a un lenguaje que pueda ser interpretado por la máquina. La existencia de problemas resolubles algorítmicamente que necesitan una cantidad de *recursos* que excede las posibilidades de cualquier ordenador electrónico, proporciona una nueva *limitación*, en este caso, de la potencia de los modelos de computación práctica.

La necesidad de estudiar la mínima cantidad de recursos necesarios para resolver un problema abstracto da origen a la *Teoría de la Complejidad.* Su objetivo fundamental consiste en proporcionar una clasificación de los problemas en función de su resolubilidad en máquinas reales.

En el marco de la *computación práctica*, se plantea la necesidad de mejorar la velocidad de cálculo y la densidad de almacenamiento de información de los ordenadores. La aparición de *máquinas paralelas* (que permiten la ejecución de varias operaciones de manera simultánea) supone un notable avance y la *miniaturización* de las componentes físicas de la máquina se convierte en un objetivo primordial. Hacia finales de la década de los cincuenta, R. Feynman introduce el concepto teórico de computación a nivel molecular y lo postula como una innovación revolucionaria en la carrera por la miniaturización. En la década de los ochenta, R. Churchhouse establece las limitaciones físicas de la velocidad de cálculo de un ordenador convencional y, en consecuencia, proporciona una nueva *limitación*: esta vez para la computación de carácter práctico.

Ante esta situación surge la necesidad de buscar modelos de computación alternativos que permitan implementar nuevas máquinas que mejoren cuantitativamente el rendimiento de las máquinas clásicas. La posibilidad de desarrollar modelos de computación no convencionales abre un nuevo horizonte a la hora de amortiguar los efectos propios de las limitaciones inherentes a la computación práctica en modelos clásicos.

La *Computación Natural* está inspirada en el funcionamiento de los organismos vivos y tiene como objetivo fundamental la simulación e implementación de los procesos dinámicos que se dan en la Naturaleza y que son susceptibles de ser considerados como procedimientos de cálculo. Esta disciplina trata de estudiar cómo *calcula* la Naturaleza y de capturar nuevos modelos y paradigmas de computación que la Naturaleza lleva implementando desde hace millones de años.

Esta memoria está dedicada al estudio de dos tipos de modelos no convencionales dentro del marco de la Computación Natural: *modelos moleculares basados en ADN* y *modelos de computación celular con membranas.*

A principio de la década de los cincuenta, J. Watson y F. Crick descifran la

estructura de las moléculas de ADN, descubren el principio de complementariedad, demuestran que dichas moléculas codifican la información genética de los organismos vivos y justifican la posibilidad de usar ciertas técnicas para su manipulación. De esta manera surge la posibilidad de implementar algoritmos a través de moléculas de ADN, debido a que éstas se pueden usar como una especie de memoria, representando la información mediante una codificación basada en cuatro nucleótidos, y a que es posible su manipulación por medio de reacciones químicas controladas. En la década de los noventa, L. Adleman resuelve una instancia del problema del camino hamiltoniano, manipulando moléculas de ADN en el laboratorio, dando origen al nacimiento de la computación molecular basada en ADN.

A nivel celular, tienen lugar una serie de procesos (reacciones químicas y flujo de diferentes componentes químicas entre distintos compartimentos) que se pueden interpretar como procedimientos de cálculo. A finales de la década de los noventa, Gh. Păun introduce un nuevo modelo de computación no determinista, de tipo paralelo y distribuido, inspirado en el funcionamiento de las células en los organismos vivos: el *modelo de computación celular con membranas*. Este modelo puede ser adecuado para explorar la naturaleza computacional de las células, así como para conseguir una posible implementación artificial de sistemas que exploten dicha capacidad celular.

Estructura de la memoria

El objetivo fundamental de esta memoria consiste en desarrollar una primera aproximación a la verificación formal de procedimientos mecánicos en modelos no convencionales, en el marco de la Computación Natural.

Para ello, se han elegido dos modelos de computación que poseen características dispares. El primero de ellos, el *modelo sticker*, es un modelo de computación molecular que usa como sustrato físico el ADN y que se describe a través de un conjunto de operaciones básicas susceptibles de ser implementadas en el laboratorio con las técnicas actuales de biología molecular. Dichas operaciones, a modo de instrucciones de un lenguaje de programación, permiten que los procedimientos diseñados en el modelo adquieran la forma de programas. El segundo de los modelos elegidos es el denominado *computación celular con membranas* o *P sistemas*. En este modelo, los procedimientos se describen por medio de mecanismos o dispositivos similares a máquinas, de ejecución independiente.

Esta memoria está, básicamente, estructurada en dos partes bien diferenciadas:

la primera está dedicada al diseño y verificación de programas moleculares en el modelo sticker; la segunda parte está dedicada al estudio de soluciones de problemas, con su correspondiente verificación, mediante el uso de P sistemas. Es decir, a través de la resolución de problemas **NP**-completos, se trata de presentar una metodología que permita establecer la verificación formal de programas, en un lenguaje de computación molecular, y de P sistemas.

La memoria está estructurada en capítulos, cuyo contenido vamos a tratar de reseñar sucintamente.

En el **capítulo 1** se ofrece una breve introducción histórica de la Teoría de la Computación y la Complejidad, justificándose la necesidad de estudiar nuevos modelos de computación como alternativa a los modelos de computación que podríamos denominar *clásicos*. Además, se describen algunas consideraciones relativas a los diversos modelos de computación que se pueden enmarcar dentro de la *Computación Natural*.

En la primera sección del **capítulo 2** se presenta una breve introducción de la estructura química de las moléculas de ADN, así como de algunas de las operaciones que se pueden realizar sobre ellas, y que hacen posible el uso de dichas moléculas como sustrato físico en el que se puede almacenar y manipular información.

En la segunda sección de este capítulo se estudian los dos primeros experimentos de laboratorio que dieron lugar al nacimiento de la computación molecular basada en ADN: el experimento de L. Adleman, que resuelve una instancia del problema **HPP** del camino hamiltoniano en su versión dirigida y con dos nodos distinguidos, y el experimento de R. J. Lipton, que resuelve una instancia del problema **SAT** de la satisfactibilidad de la lógica proposicional.

El **capítulo 3** está dedicado a la presentación de tres modelos de computación molecular basados en ADN. El *modelo no restringido* de Adleman fue el primer modelo abstracto de computación molecular que se introdujo de manera explícita, y está inspirado en los experimentos de Adleman y de Lipton. Junto con el *modelo débil* de Amos, representan dos modelos de computación molecular *sin memoria*, ya que sus operaciones básicas no alteran la estructura interna de las moléculas de ADN.

El tercer modelo presentado, el *modelo sticker*, es más interesante desde el punto de vista abstracto, ya que hace uso de las moléculas de ADN como unidades de memoria susceptibles de ser modificadas *en tiempo de ejecución*. Por ello, en este capítulo se hace una exposición exhaustiva de dicho modelo, presentando lo que podría ser una implementación física del mismo en el laboratorio. Este modelo es el que se considera en la primera parte de la memoria, para el diseño y verificación de

programas que resuelven problemas **NP**–completos.

La última sección de este capítulo está dedicada al estudio de la universalidad de los modelos de computación molecular basados en ADN. Para ello, se describe una simulación de las máquinas de Turing a través de la manipulación de moléculas de ADN mediante unas operaciones elementales y una codificación molecular adecuada.

En el **capítulo 4** se presenta una metodología para la verificación de programas en el modelo sticker. Para ello, en la primera sección se describen los programas moleculares diseñados para resolver problemas de decisión, a través de sistemas formales. De tal manera que demostrar la verificación de un programa molecular sea equivalente a establecer la adecuación y completitud del sistema formal asociado.

Como aplicación de la metodología diseñada, en la segunda sección de este capítulo se presentan soluciones moleculares de una serie de problemas, relativos a conjuntos numéricos, y se establecen las verificaciones correspondientes. Los procedimientos diseñados serán usados como subrutinas en el capítulo siguiente, en el que se aborda la resolución de algunos problemas **NP**-completos en el modelo sticker.

El capítulo 4 finaliza con un análisis de los costes de los programas diseñados, en lo que respecta al número de operaciones moleculares, al número de tubos usados y al volumen molecular utilizado.

La primera parte de esta memoria, correspondiente a los modelos de computación molecular, finaliza con el **capítulo 5** que está dedicado al estudio (diseño, verificación y análisis) de soluciones moleculares de algunos problemas numéricos **NP**-completos. Los problemas que se estudian son los siguientes: el problema *Subset-Sum*, el problema de la *mochila 0/1* (en su versión acotada y no acotada), el problema *Set-Packing*, el problema *Exact-Cover* y el problema del *recubrimiento minimal*.

La segunda parte de la memoria comienza en el **capítulo 6** y aborda la verificación formal de P sistemas. En la primera sección de este capítulo se presenta una definición informal de los *P sistemas de transición*, como modelo de computación inspirado en el funcionamiento de las células. Se trata de un modelo diseñado a partir de sistemas autónomos que generan lenguajes de manera no determinista. Estos sistemas están basados en una *estructura* jerarquizada de *membranas* como modelo matemático de ordenación física de la célula. Cada una de las membranas pueden contener unos *objetos*, a modo de las distintas componentes químicas, y unas *reglas de evolución* que nos permitirán, eventualmente, modificar los objetos presentes en las membranas y/o comunicarlos entre las mismas.

En la segunda sección del capítulo 6 se presenta una formalización de los P sistemas de transición que nos permite abordar con garantías la verificación formal en este modelo. La formalización presentada se basa, principalmente, en la evolución

del árbol asociado a la estructura de membranas, con los multiconjuntos de objetos asociados, y en la especificación de los posibles multiconjuntos de reglas que se pueden aplicar en cada instante.

La tercera sección está dedicada al estudio de la versatilidad de los P sistemas de transición, que pueden ser interpretados como dispositivos generadores de lenguajes, reconocedores de predicados, o como máquinas de cálculo de funciones. A continuación se hace un breve estudio acerca de la universalidad de este modelo y se justifica la utilidad del mismo para atacar la conjetura **P=NP**, tanto para dar una respuesta positiva como negativa.

En la última sección del capítulo 6 se presentan algunas variantes de los P sistemas de transición, según el modo de considerar los objetos, según las características de las membranas o según el protocolo de las comunicaciones.

En su artículo fundacional, Gh. Păun ilustra el funcionamiento de los P sistemas de transición con varios ejemplos que reflejan las características principales del modelo y muestran las diversas interpretaciones de los mismos. En el **capítulo 7** se establece la verificación de los P sistemas diseñados por Gh. Păun (algunos con ligeras modificaciones) ilustrando el uso de algunas técnicas que pueden ser útiles para abordar, con cierto éxito, la verificación formal de P sistemas arbitrarios, y poniendo de manifiesto cómo el paralelismo distribuido inherente al modelo dificulta enormemente la tarea de verificación.

El **capítulo 8**, último de la segunda parte de la memoria, está dedicado al diseño, y posterior verificación, de P sistemas de transición que resuelven problemas **NP**-completos clásicos: el problema de la satisfactibilidad de la lógica proposicional y el problema del camino hamiltoniano en su versión dirigida y con dos vértices distinguidos. Las soluciones no deterministas presentadas resuelven los problemas de decisión citados en un sentido fuerte; es decir, los lenguajes generados por los P sistemas diseñados coinciden con los conjuntos de soluciones válidas (positivas) de los problemas planteados.

En el último capítulo de esta memoria, el **capítulo 9**, se destacan algunas conclusiones que se pueden extraer de los resultados obtenidos, y se presentan una serie de objetivos y trabajos futuros que marcan una línea de investigación en modelos de computación no convencionales, algunas de las cuales han comenzado a desarrollarse por 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. Desarrollo de un método de representación de conjuntos numéricos en el modelo sticker, usando la capacidad del modelo para identificar complejos de memoria con funciones booleanas.

2. Diseño de procedimientos moleculares en el modelo sticker que resuelven los siguientes problemas (y que serán usados como subrutinas):

   - **El problema del rellenado**: *Sea* $A = \{1, \ldots, p\}$. *Sea* $\mathcal{F}$ *una familia finita de subconjuntos de* A. *Determinar todos los pares ordenados* $(\mathcal{F}', B)$, *en donde* $\mathcal{F}'$ *es una subfamilia de* $\mathcal{F}$ *y* $B = \bigcup \mathcal{F}'$.
   - **El problema del recubrimiento**: *Sea* $A = \{1, \ldots, p\}$ *y* $\mathcal{F}$ *una familia finita de subconjuntos de* A. *Dada la colección de pares de la forma* $\{(\mathcal{F}', \bigcup \mathcal{F}') : \mathcal{F}'$ *subfamilia de* $\mathcal{F}\}$, *determinar* <u>todos</u> *los elementos de esta colección que constituyen un recubrimiento de* A.
   - **El problema de la ordenación según la cardinalidad**: *Sean* $A = \{1, ..., p\}$, $B \subseteq A$, *y* $\mathcal{F} \subseteq \mathcal{P}(A)$. *Ordenar los conjuntos de* $\mathcal{F}$ *de acuerdo con su cardinal relativo a* B.
   - **El problema de las familias disjuntas**: *Sea* $A = \{1, \ldots, p\}$. *Sea* $\mathcal{F}$ *una familia finita de subconjuntos de* A. *Determinar todas las subfamilias de* $\mathcal{F}$ *cuyos elementos son disjuntos dos a dos.*
   - **Funciones Peso sobre conjuntos**: *Dada una familia de subconjuntos de* $A = \{1, ..., p\}$ *y una función* $f : A \longrightarrow \mathbf{N}$, *hallar los pesos, respecto de la función* f, *de los elementos de la familia.*

3. Mejora de una solución molecular del problema del recubrimiento minimal, debida a S. Roweis y otros.

4. Diseño de las primeras soluciones moleculares de los siguientes problemas **NP**-completos:

   - **El problema Subset-Sum**: *Sea* $A = \{1, ..., p\}$ *y* $w : A \longrightarrow \mathbf{N}$ *una función peso. Sea* $k \in \mathbf{N}$ *tal que* $k \leq w(A)$. *Determinar si existe un subconjunto,* $B \subseteq A$, *cuyo peso sea, exactamente,* k.

- **El problema de la Mochila 0/1 acotado**: *Sean* $A = \{1, ..., p\}$ *un conjunto no vacío,* $w : A \longrightarrow \mathbf{N}$ *una función peso, y* $\rho : A \longrightarrow \mathbf{N}$ *una función aditiva de valores. Sean* $k, k' \in \mathbf{N}$ *tales que* $k \leq w(A)$ *y* $k' \leq \rho(A)$. *Determinar si existe un subconjunto* $B \subseteq A$ *tal que* $w(B) \leq k$ *y* $\rho(B) \geq k'$.
- **El problema de la Mochila 0/1 no acotado**: *Bajo las mismas condiciones que en el problema de la mochila 0/1 acotado, determinar un subconjunto* $B \subseteq A$ *que verifique* $\rho(B) = \max\{\rho(C) : C \subseteq A \wedge w(C) \leq k\}$.
- **El problema SET PACKING**: *Sean* $A = \{1, \ldots, p\}$ *un conjunto finito,* $\mathcal{F}$ *una familia de subconjuntos de A, y* $k \in \mathbf{N}$. *Determinar si existen k subconjuntos de* $\mathcal{F}$ *disjuntos dos a dos.*
- **El problema EXACT-COVER**: *Sean* $A = \{1, \ldots, p\}$ *un conjunto finito y* $\mathcal{F}$ *una familia de subconjuntos de A. Determinar si existe una subfamilia de* $\mathcal{F}$ *que sea partición de A.*
- **El Problema del recubrimiento Minimal**: *Sea* $A = \{1 \ldots, p\}$. *Sea* $\mathcal{F}$ *una familia finita de subconjuntos de A. Encontrar el menor recubrimiento de A por elementos de* $\mathcal{F}$.

5. Desarrollo de una metodología para la verificación de programas en un modelo de computación molecular con memoria.

6. Aplicación de la metodología anterior a los programas diseñados en el modelo sticker.

7. Elaboración de una formalización de los P sistemas de transición.

8. Primera aproximación a la verificación de P sistemas de transición.

9. Diseño de P sistemas de transición que proporcionan soluciones no deterministas, en sentido fuerte, de los problemas **SAT** y **HPP**.

El estudio sistemático de la verificación de programas moleculares es una tarea que ha sido iniciada, en el año 2000, por miembros del Grupo de Investigación en Computación Natural de la Universidad de Sevilla. En esta memoria se complementan los procesos de verificación para programas moleculares diseñados en modelos con memoria, y para sistemas en modelos celulares con membranas.

Esperamos que esta memoria proporcione métodos y técnicas que puedan ser de utilidad para estudios posteriores acerca de dichos modelos no convencionales, en

particular para el análisis de la robustez de procedimientos complejos descritos en dichos modelos y para la mecanización de los procesos de verificación en sistemas de razonamiento automático.

Capítulo 1

Computación Natural

1.1. Introducción

Desde el comienzo de los tiempos el hombre se ha planteado problemas, bien por una cuestión de supervivencia, por mejorar sus condiciones de vida o, simplemente, por una cuestión lúdica. El estudio de los problemas planteados le ha conducido, de manera natural, a la búsqueda de procedimientos sistemáticos que permitieran resolver dichos problemas realizando una serie de procesos elementales debidamente secuenciados. La mecanización de las soluciones facilitaba la transmisión del conocimiento de dichos problemas y el aprendizaje científico, en general.

La aparición de los primeros métodos formales de razonamiento (posiblemente en Mesopotamia, si bien sería sustancialmente mejorados en la civilización griega) proporcionó una herramienta interesante para poder expresar de forma precisa el concepto intuitivo de *procedimiento sistemático, procedimiento mecánico* o *algoritmo.* Es usual encontrar en un diccionario el término *algoritmo* como sinónimo de *cualquier método especial de resolución de un cierto tipo de problemas.* La palabra *algoritmo* debe su nombre al autor persa Abú Jáfar Mohanned 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.

Ahora bien, hasta muchos siglos después no empieza a surgir la necesidad de rigorizar el concepto de procedimiento mecánico. En primer lugar, cuando a finales del siglo XVII Leibnitz 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*). En segundo lugar, cuando D. Hilbert, en el Congreso Internacional de Matemáticos celebrado en Bolonia en 1928, formula tres cuestiones acerca de las matemáticas que marcarían

el devenir de la misma:

1. *¿Son completas?* Es decir, dado cualquier aserto matemático, ¿es posible probar dicho enunciado o su negación?

2. *¿Son consistentes?* Es decir, ¿es posible garantizar que dado cualquier aserto matemático no se puedan probar, simultáneamente, él y su negación?

3. *¿Son decidibles?* Es decir, ¿existe algún procedimiento mecánico que dado cualquier aserto matemático determine si es o no demostrable? (*Entscheidungsproblem*).

1.2. Lógica y Computación

En esta última cuestión aparecen implícitamente otras, como por ejemplo, ¿qué pueden calcular los distintos algoritmos? ¿cuáles son los requisitos que debemos exigir a los algoritmos para que tengan una potencia computacional adecuada? ¿qué clasificación puede realizarse entre los algoritmos en función de la cantidad de recursos que necesitan para su ejecución?

A finales del siglo XIX, la necesidad de resolver ciertos problemas, por parte de los matemáticos, les llevó a usar el formalismo y la metodología de la lógica matemática (que, como cualquier rama de las matemáticas, se puede considerar una colección de técnicas para resolver problemas y que se diferencia de otras por el hecho de usar lenguajes formales). En la *escuela formalista*, con B. Russell y D. Hilbert al frente, comienza a gestarse el concepto de *computabilidad efectiva* y uno de sus principales objetivos consistía en reducir todas las matemáticas a la manipulación formal de símbolos, que no es más que una forma de *computación.*

El *método axiomático* permite describir el comportamiento de una *teoría* o *sistema* y consta de tres ingredientes fundamentales: un *lenguaje*, unos *axiomas* y unas *reglas de inferencia.* El *lenguaje* permite describir los objetos que se van a estudiar, así como las propiedades relativas a dichos objetos. Los *axiomas* son asertos elementales, expresados en el lenguaje, que describen ciertas propiedades básicas de la *teoría* que es objeto de estudio. Las *reglas de inferencia* son una especie de reglas de juego que permiten obtener nuevos enunciados en la teoría (y que se denominan *teoremas*) a partir de los axiomas. De esta manera, en una teoría descrita a través de un sistema axiomático, se pueden generar nuevos resultados (es decir, teoremas de la teoría) realizando un número finito de operaciones elementales (que resultan directamente de las reglas de inferencia) a partir de los axiomas. Con otras palabras,

los teoremas de una teoría axiomática se pueden obtener, propiamente a partir de un procedimiento sistemático; es decir, mediante computaciones.

A principios del siglo XX, los matemáticos estaban a punto de realizar una serie de descubrimientos que marcarían el devenir de las ciencias, en general, y de las matemáticas, en particular. Había un convencimiento generalizado de que las matemáticas podían ser descritas a través de un sistema axiomático, de tal manera que bastaba encontrar el lenguaje, los axiomas y las reglas de inferencia adecuadas para deducir en él cualquier enunciado matemático. De esta forma, las matemáticas vendrían a ser una especie de sistema computacional en el que podría establecerse, mediante un procedimiento mecánico, la veracidad o falsedad de cualquier aserto matemático. Esta idea la había expresado D. Hilbert formulando la primera cuestión, antes descrita, acerca de la completitud de las matemáticas.

Todas estas suposiciones y conjeturas se derrumban cuando, en 1931, K. Gödel presentó sus teoremas de incompletitud. Estos resultados mostraban que era irrealizable el Programa de Hilbert (que trataba de justificar el uso de la metodología transfinita y, al mismo tiempo, proporcionar una formalización completa de las matemáticas). Para ello, K. Gödel consideró un sistema simple de la Aritmética para describir la matemática finitaria (aquella que proporciona métodos constructivos). En este sistema describió su propia sintaxis y construyó una proposición (*yo no soy demostrable*) que no se podía probar en el sistema (y siendo, por tanto, una fórmula verdadera). Este resultado es considerado como uno de los logros científicos más importantes del siglo XX y se trata, básicamente, de un resultado sobre *computabilidad.*

Posteriormente, Gödel demostró que dicho sistema no era capaz de demostrar su propia consistencia, dando también una respuesta negativa a la segunda cuestión planteada por Hilbert.

De esta manera resultó que no era posible encontrar una axiomatización completa de las matemáticas. Entonces surge una nueva cuestión: si no existe un sistema axiomático capaz de demostrar cualquier aserto matemático formulado en dicho sistema ¿qué puede conseguirse con los distintos sistemas axiomáticos?

1.3. Primeros modelos de computación

Cuando se estudian concienzudamente fenómenos de la vida real se puede percibir la existencia de una serie de rasgos fundamentales que son similares en el fondo, aunque parezcan diferentes en la forma. Entonces puede ser interesante construir un

modelo que capture, de la forma más simple posible, esos rasgos comunes subyacentes a los fenómenos analizados. Este es el *proceso de abstracción*, uno de los elementos fundamentales del progreso de la ciencia: hay que prestar atención a las propiedades importantes desprovistas de los detalles irrelevantes. Este proceso es inherentemente matemático.

Se han desarrollado modelos para máquinas (Turing, RAM, URM, etc.), para lenguajes formales (regulares, libres de contexto, recursivamente enumerables, etc.), para sistemas de programas (compiladores, sistemas operativos, etc.), para estructuras de datos (pilas, colas, montículos, etc.) y para bases de datos (relacionales, orientadas a objetos, etc.).

Definir un *modelo de computación* consiste, básicamente, en formalizar, rigorizar, cuáles son los procedimientos que serán considerados como mecánicos en el modelo (*sintaxis* del modelo), y en precisar cómo se ejecutan dichos procedimientos sobre unos datos de entrada (*semántica del modelo*). De esta manera, los procedimientos mecánicos determinarán, de manera natural, la clase de *funciones* que son consideradas *computables* en el modelo de computación (es decir, la clase de *problemas* que son *resolubles algorítmicamente*).

A la hora de formalizar el concepto de *algoritmo* se puede optar por varias vías cualitativamente distintas:

- Definir 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 este concepto, 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 modelo.

- Considerar un cierto conjunto de funciones 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 pueden ejecutar los algoritmos.

Los trabajos de K. Gödel, A. Church, S. Kleene y A. Turing, entre 1931 y 1936, proporcionan las primeras formalizaciones del concepto de *algoritmo*, dando lugar a la aparición de los primeros *modelos de computación*. De esta manera se define con rigor el concepto de *función computable*; es decir, función cuyos valores pueden ser *calculados* de una forma mecánica, automática o efectiva.

Concretamente, en 1931 K. Gödel [27] define el concepto de *relación recursiva* e introduce la clase de funciones que denominó *recursivas* (y que hoy se conocen como *primitivas recursivas*). Posteriormente, en 1934, el propio Gödel extiende la clase anterior a las funciones *general recursivas* (que son las que conocemos hoy con el nombre de *recursivas*). En 1931, A. Church y S. Kleene desarrollan el concepto de *λ-cálculo* relacionándolo directamente con el concepto intuitivo de *función computable*. En 1936, A. Turing [83] utiliza por primera vez el concepto abstracto de *máquina* como formalización del concepto de algoritmo y, por tanto, de las funciones computables. De esta manera, a mediados de la década de los treinta se tienen tres formalizaciones diferentes del concepto de procedimiento mecánico.

La idea de Turing al introducir su concepto de *computabilidad* a través de *máquinas abstractas*, consistía en desarrollar un sistema capaz de modelizar cualquier procedimiento susceptible de ser interpretado intuitivamente como un *proceso de cálculo*. Es decir, trataba de diseñar un *modelo de computación* en su sentido más literal: imaginó a un *ordenador-humano*, a una persona con papel y lápiz, resolviendo un determinado problema e intentó capturar los pasos más simples en los que se podía descomponer el proceso que estaba realizando.

1.4. Problemas indecidibles

En 1936, A. Church formula su famosa *tesis* acerca de la equivalencia entre la clase de funciones computables (desde el punto de vista intuitivo) y la clase de funciones λ-calculables. Esta tesis relaciona dos conceptos de naturaleza completamente distinta: existencia de un procedimiento mecánico que resuelve un problema (concepto informal) y existencia de una máquina de Turing que resuelve un problema (concepto formal). Por tanto, no tiene sentido buscar una prueba matemática de dicha tesis.

Además, A. Church [16] proporciona el primer ejemplo de un problema que no es resoluble algorítmicamente (según la formalización que da el λ-cálculo): *la indecidibilidad de la lógica de primer orden*. De esta manera, responde negativamente a la tercera cuestión formulada por D. Hilbert. Es decir, no existe un procedimiento mecánico capaz de obtener todos los asertos matemáticos que son demostrables, y sólo esos. Podemos deducir de aquí que la actividad matemática es, básicamente, un proceso creativo ya que no puede ser mecanizado (ni siquiera desde el punto de vista teórico).

La limitación de la potencia computacional no puede considerarse propiamente

como inherente al modelo de Church-Kleene. Se puede probar que en todo modelo computacional que verifique una serie de propiedades básicas (como, por ejemplo, que los algoritmos se puedan expresar mediante una cadena finita de símbolos) se pueden describir problemas abstractos que no son resolubles algorítmicamente en el modelo (básicamente, por el hecho de que la clase de funciones computables en el mismo es numerable).

Pocos meses después del resultado de Church, A. Turing [83] establece de manera independiente el mismo resultado pero en su modelo de computación (el Entscheidungsproblem tenía una solución negativa) probando que no existía una máquina de Turing que proporcione un método de decisión para la lógica de primer orden. Lo esencial de la prueba de Turing no radica en la estructura de su modelo computacional, sino en cómo utilizó dicho modelo.

Concretamente, Turing describió mediante una fórmula de primer orden la relación existente entre una configuración de la máquina y la siguiente; es decir, cada paso computacional de la máquina podía ser descrito por una fórmula. De este hecho y de la irresolubilidad algorítmica del *problema de la parada* (dada una máquina de Turing y una configuración inicial, determinar si la máquina para o no), resultado que acababa de probar, dedujo la indecidibilidad de la lógica de primer orden. Así aparece la primera reducción de un problema específicamente computacional (el problema de la parada) en un problema *híbrido* de lógica y computación (el Entscheidungsproblem).

En el trabajo antes citado, Turing establece la equivalencia de su modelo y el de las funciones λ-calculables, y anuncia la equivalencia entre la clase de funciones computables por máquinas de Turing y la clase de funciones recursivas, que probaría en 1937 [84]. De esta manera se tiene que los tres modelos de computación introducidos formalmente son equivalentes en el sentido de que todo aquello que es calculable en uno de esos modelos lo es en cualquiera de los otros dos.

Se piensa que las máquinas de Turing (y, por tanto, los modelos de computación de las funciones recursivas y del λ-cálculo) poseen una potencia computacional universal: ningún modelo de computación construido posteriormente puede realizar computaciones que no se puedan simular en dichas máquinas. Ahora bien, esta suposición se refiere propiamente a la potencia computacional y no a la eficiencia: por ejemplo, las máquinas con memoria de acceso aleatorio pueden realizar las mismas computaciones que las máquinas de Turing pero usando menos memoria y en menos tiempo.

Con el desarrollo de máquinas computacionales teóricas (antes aún de que la tecnología permitiera su construcción), los investigadores comienzan a centrarse en

el estudio de la potencia y limitaciones de dichas máquinas. Estas máquinas teóricas ayudarían de forma decisiva a la construcción de los actuales ordenadores, basados en el modelo conceptual de John von Neumann que, a su vez, está inspirado en los trabajos de Turing acerca de una máquina universal, programable y de propósito general.

Podemos decir que este momento marca, propiamente, el inicio de la *Teoría de la Computación*, cuyo objetivo principal es la clasificación de problemas abstractos según sean o no resolubles algorítmicamente. Como toda teoría interesante, la Teoría de la Computación aborda a la vez aspectos positivos (¿qué se puede calcular con los procedimientos mecánicos introducidos formalmente?) y negativos (¿existe algún problema que no se puede resolver de manera mecánica?).

La Teoría de la Computación trata cuestiones del siguiente tipo:

- Dado un problema abstracto ¿puede ser resuelto algorítmicamente?
- Si un problema abstracto es resoluble algorítmicamente
  - ¿pueden ser obtenidas sus soluciones en alguna máquina concreta?
  - ¿qué cantidad de recursos es necesario para obtener sus soluciones?
  - ¿cuál es la mínima cantidad de recursos necesarios para resolverlo en una máquina concreta?
  - la cantidad mínima de recursos necesarios para resolverlo ¿es tan grande que no se puede ejecutar en una máquina real?

1.5. Teoría de la Complejidad

En la década de los cincuenta se desarrollan los primeros *lenguajes de programación*, *traductores de lenguajes* y *sistemas operativos*. La potencia de los ordenadores en esa época estaba muy limitada por la excesiva lentitud de los procesadores y la escasa memoria de la que disponían para almacenar la información. Por ello, empiezan a desarrollarse teorías cuyo objetivo es la exploración del uso eficiente de los ordenadores, lo que conlleva de alguna manera el estudio de la *complejidad intrínseca* de problemas abstractos.

En la década de los sesenta se elaboran los primeros cimientos de la *Teoría de la Complejidad* con la clasificación de lenguajes y funciones (debidas a J. Hartmanis, P.M. Lewis y R.E. Stearns [29][41]), en función del tiempo y del espacio necesario para su generación o cálculo. Asimismo, se desarrollan métodos de análisis para

estudiar la eficiencia de los algoritmos y las estructuras de datos usadas en los mismos, la expresividad de los lenguajes formales, la capacidad computacional de las arquitecturas de los ordenadores y la clasificación de problemas según la cantidad de recursos necesarios para su resolución.

El problema de la decidibilidad de una teoría, T, de primer orden consiste en lo siguiente: *dada una fórmula en el lenguaje de T, determinar si la teoría prueba dicha fórmula.* Con la introducción de los modelos formales de computación, se dispone de las herramientas necesarias para atacar dicho problema. Si se dispone de un algoritmo de decisión para una teoría T, dada una fórmula, φ, sobre el lenguaje de dicha teoría, bastará ejecutar el algoritmo con dato de entrada φ para responder si T prueba o no prueba φ.

No obstante, los trabajos de P.C. Fischer, A.R. Meyer, M.O. Rabin, S. Cook y otros, en la década de los sesenta, ponen de manifiesto la existencia de teorías decidibles que eran, desde un punto de vista práctico, *semejantes* a una indecidible, en tanto en cuanto cualquier algoritmo de decisión requería una cantidad de tiempo y/o espacio tan elevada que hacía irrealizable la computación en la práctica.

A partir de este momento se hace imprescindible el estudio de la cantidad de recursos que un algoritmo necesita para su ejecución. Dicho análisis requiere el uso de técnicas matemáticas (inducción, ecuaciones de recurrencia, notaciones asintóticas, manipulación de sumas finitas, etc.) que, además, permitan un estudio comparativo de distintas soluciones algorítmicas de un mismo problema.

Ahora bien, a veces, el mejor algoritmo conocido que resuelve un problema puede tener un coste muy elevado. Entonces, parece natural plantearse la búsqueda de otros algoritmos que usen estrictamente menos recursos que el mejor conocido y que también resuelvan el problema. De esta manera se plantea una nueva cuestión: *dado un problema resoluble algorítmicamente, hallar el mejor algoritmo que lo resuelva.* El concepto de *mejor solución* estará referido a una cierta *medida de complejidad* que cuantifique los recursos.

Un procedimiento para determinar un *algoritmo óptimo* que resuelve un determinado problema consistiría en lo siguiente:

- Determinar una cota inferior de la cantidad de recursos que necesita para su ejecución cualquier algoritmo que resuelva dicho problema.
- Hallar un algoritmo que resuelva el problema y, además, la cantidad de recursos que utiliza es del orden de la cota inferior.

Si de un cierto problema abstracto se conoce un *algoritmo óptimo* que lo resuelve, entonces la cantidad de recursos que utiliza dicho algoritmo proporcionará, de manera

natural, la *complejidad computacional inherente* a dicho problema.

Es interesante hacer notar que la cuestión relativa a hallar un algoritmo óptimo que resuelve un problema, tiene un cierto paralelismo con la obtención de problemas irresolubles algorítmicamente: se trata de hallar un algoritmo que satisfaga una propiedad que implica a todos los algoritmos que resuelven dicho problema.

Como es fácilmente imaginable, la tarea de calcular un *algoritmo óptimo* que resuelva un problema suele ser árdua, complicada y, a veces, *imposible* de resolver. En efecto: si las medidas de complejidad que se consideran para cuantificar los recursos satisfacen unos requisitos mínimos (por ejemplo, los *axiomas de Blum* [11]), entonces existe algún problema resoluble algorítmicamente que carece de algoritmo óptimo, respecto a dichas medidas, que lo resuelve (*teorema de aceleración*). En consecuencia, no se puede definir de manera *individual* el concepto de complejidad computacional de un problema, ya que siempre existiría algún problema al que no se le podría asignar complejidad alguna de acuerdo con esta definición.

La alternativa que se considera es el estudio de la complejidad de los problemas de una manera *global*; es decir, a través del análisis de la complejidad de clases de problemas, dando lugar a las denominadas *clases de complejidad.*

La *Teoría de la Complejidad* proporciona herramientas para medir la dificultad de problemas abstractos, a la vez, en términos absolutos (complejidad inherente de un problema) y en términos comparativos con otros problemas (clases de complejidad).

El objetivo fundamental de la *Teoría de la Complejidad* es la clasificación de problemas en función de la *resolubilidad algorítmica práctica* de los mismos. Para ello, se define un concepto de *eficiencia* o resolubilidad práctica. 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 establece la frontera entre la resolubilidad algorítmica práctica (*tratabilidad*) y la no resolubilidad algorítmica práctica (*intratabilidad*). Desde este punto de vista, los problemas se clasifican en *tratables* e *intratables*, según sean o no resolubles de forma eficiente. La clase de complejidad de los problemas tratables se designa por **P**.

Ahora bien ¿qué motiva el hecho de que algunos problemas sean computacionalmente difíciles y otros sean fáciles? No siempre es fácil decidir qué problemas son tratables y cuáles no lo son. Más aún, existe una clase amplia de problemas de los que no sabemos si son tratables o no.

Con la idea informal de lo que es un *algoritmo no determinista* (en donde se admite una instrucción de elección), se puede obtener una nueva clase de complejidad

de problemas, que designaremos por **NP** y que contendrá aquellos problemas que son resolubles por un algoritmo no determinista en tiempo polinomial. Una de las cuestiones matemáticas más importantes, y que permanece abierta en la actualidad, es la relación existente entre las clases de complejidad **P** y **NP** (es decir, si **P**=**NP** ó **P**$\subsetneq$**NP**).

Dentro de la clase **NP** podemos destacar una subclase de problemas que tienen especial interés: los problemas que son los más *difíciles* de la clase, en el sentido de que cualquier otro problema de la clase **NP** puede ser resuelto a través de él con un coste en tiempo adicional de tipo polinomial (mediante una *m-reducción*). Es la clase de los problemas denominados **NP**-completos. El interés de esta clase radica en que pueden ser candidatos idóneos para atacar la conjetura **P**$\stackrel{?}{=}$**NP**. En efecto, es fácil probar que si existe **un** problema **NP**-completo que es tratable, entonces la respuesta a la conjetura es afirmativa; y si existe **un** problema **NP**-completo que no es tratable, entonces la respuesta es negativa.

En 1971, S.A. Cook [18] proporciona el primer ejemplo de un problema **NP**-completo: el problema **SAT** de la satisfactibilidad de la Lógica Proposicional. Un año después, teniendo presente que la m-reducibilidad permite generar nuevos problemas **NP**-completos a partir de otros conocidos, R.M. Karp [35] da 24 ejemplos nuevos de problemas **NP**-completos (entre los que destacan el problema del recubrimiento de vértices, el problema del recubrimiento exacto, el problema del número cromático, el problema del circuito hamiltoniano y el problema del viajante de comercio). En la actualidad se conocen muchos problemas **NP**-completos de disciplinas tan diversas como lógica, teoría de números, teoría de grafos, investigación operativa, etc. El libro [26] de M.R. Garey y D.S. Johnson constituye un catálogo exhaustivo de problemas **NP**-completos.

Cuando nos enfrentamos a un problema computacionalmente difícil hay varias maneras de abordarlo:

- Preguntarnos en qué aspecto del problema radica la razón de la dificultad.

- Intentar buscar una solución aproximada más simple en lugar de una solución exacta del problema.

- Tener presente que algunos problemas sólo son difíciles en el caso peor (que se podría dar poco) y, en cambio, ser fáciles en los restantes casos. Así se podría obtener un procedimiento mecánico ocasionalmente lento pero que muchas veces es rápido.

- Considerar otros modelos alternativos, no convencionales, de computación que, amplíe, en algún sentido el concepto de tratabilidad.

1.6. La Naturaleza como fuente de inspiración computacional

Muchos problemas interesantes que se pueden resolver por medio de algoritmos en un determinado modelo precisan de un alto coste para su resolución, ya sea en tiempo y/o en espacio, siendo habitual que el intento de disminuir una de las dos medidas provoca un crecimiento exponencial en la otra. Por ello, surge la necesidad de buscar nuevos modelos que sean capaces de reducir ambos parámetros o, al menos, de incluir procedimientos en los que un coste alto en una de las medidas sea *asimilado*, en cierto sentido, por el propio modelo en beneficio de una reducción considerable sobre la otra.

En este contexto, la búsqueda de nuevos modelos alternativos de computación está encaminada a la mejora cuantitativa en los resultados que proporciona la *Teoría de la Complejidad.*

En los últimos años, esta búsqueda ha dado como resultado la introducción de nuevos modelos de computación sustancialmente distintos de los clásicos o convencionales (máquinas de Turing, funciones recursivas, λ-cálculo, máquinas URM, modelo GOTO, etc.) que proporcionan una mejora importante en las medidas de complejidad y en el marco de una posible implementación práctica.

La *Computación Natural* surge como una de las posibles alternativas a la computación que podríamos denominar clásica, en la búsqueda de nuevos paradigmas que puedan proporcionar una solución *efectiva* a las limitaciones que poseen los modelos convencionales. Actualmente, dentro del concepto de *Computación Natural* se engloba un conjunto de modelos que tienen como característica común la simulación del modo en que la naturaleza actúa/opera sobre la materia (hay quien extiende este concepto hasta abarcar modelos tales como la *computación cuántica*, que no se ajusta fielmente a la interpretación anterior). Es decir, la Computación Natural estudia la forma en que las diversas leyes de la naturaleza producen modificaciones en determinados sistemas (desde hábitats hasta conjuntos de moléculas, pasando por organismos vivos) que pueden ser interpretados como procesos de cálculo sobre sus elementos. Así, un hábitat en el que varias especies de seres vivos conviven, sufre transformaciones con el paso de las generaciones y en donde la interacción entre las especies puede provocar cambios en los elementos que la forman (cambios

que pueden afectar desde la distribución de dichas especies en el hábitat hasta la morfología propia de cada especie). Es lo que entendemos como *evolución* de las especies; un conjunto de moléculas en un entorno con determinadas características (de temperatura, salinidad, etc) puede evolucionar hacia estados estructuralmente más complejos modificando las características de las mismas; es decir, propiciando reacciones químicas entre ellas que pueden llegar a producir elementos funcionalmente más complejos, como son las moléculas de ácido desoxirribonucléico (ADN).

Esta simulación que aborda la Computación Natural puede tener diversas interpretaciones a la hora de describir los nuevos modelos: que se utilice para el diseño de nuevos esquemas algorítmicos usando técnicas inspiradas en la naturaleza, o bien que sugiera la creación física de nuevos modelos experimentales en los que el medio electrónico de los ordenadores convencionales se sustituya por otro sustrato que pueda implementar ciertos procesos que aparecen en el modo de operar de la naturaleza.

Como ejemplo de la primera interpretación, podemos considerar los *Algoritmos Genéticos*, que se basan en el proceso genético de los seres vivos a través del cual evolucionan y cuyo elemento fundamental es el principio de selección natural.

Como ejemplo de la segunda interpretación, a finales de la década de los cincuenta el premio nobel R.P. Feynman [24] postula la necesidad de considerar operaciones *sub–microscópicas* como única alternativa revolucionaria en la carrera por la miniaturización de las componentes físicas de los ordenadores convencionales (basados en circuitos de silicio), y propone la computación a *nivel molecular* como posible modelo en el que implementar dichas operaciones. En 1987, T. Head [30] propone explícitamente el primer modelo teórico molecular basándose en las propiedades de la molécula de ADN. En noviembre de 1994, L. Adleman [1] realiza el primer experimento en un laboratorio que permite resolver una instancia concreta de un problema **NP**-completo a través de la manipulación de moléculas de ADN.

A continuación pasamos a estudiar con un poco más de detalle algunas cuestiones relativas a los principales modelos de Computación Natural.

1.6.1. Algoritmos Genéticos y Redes Neuronales

Los *Algoritmos Genéticos* y las *Redes Neuronales* tienen en común inspirarse en procesos que se dan en la naturaleza para diseñar nuevos tipos de algoritmos que pueden ser, posteriormente, implementados en ordenadores convencionales.

Los principios básicos de los *Algoritmos Genéticos* fueron propuestos por J.H. Holland [32] en 1975, y están inspirados en los procesos genéticos de los organismos vivos que usan el principio de selección natural para eliminar los individuos menos

dotados o adaptados. Para ello, se combinan los genes de los individuos (que pasarán a considerarse las posibles soluciones) presentes en una generación a fin de formar los genes de los individuos de la siguiente (incluso se permite la introducción de modificaciones aleatorias, *mutaciones*) y, de entre ellos, se seleccionan aquellos individuos que presenten las mejores características (de acuerdo con una medida establecida de antemano). Si el algoritmo genético es diseñado correctamente, entonces la evolución de la población convergerá a un individuo (*solución*) óptimamente adaptado.

Aunque usan técnicas aleatorias, los algoritmos genéticos son, propiamente, algoritmos de búsqueda que explotan eficientemente la información histórica de los individuos que se van obteniendo a fin de razonar sobre las propiedades de los mismos que pueden ser relevantes para la optimización de las soluciones.

Habitualmente, si para resolver un problema existen técnicas eficientes específicas que lo resuelven, éstas mejorarán los costes de aquellos que se puedan desarrollar por medio de un algoritmo genético. La gran baza que juegan estos algoritmos radica en su aplicación a problemas para los cuales no existen tales técnicas eficientes.

Las *Redes Neuronales Artificiales* surgieron originalmente como una simulación del sistema nervioso, formado por un conjunto de unidades (*neuronas*) conectadas entre sí, simulando a las *dendritas* y los *axones* en los sistemas nerviosos biológicos.

El primer modelo de red neuronal fue propuesto en 1943 por W.S. McCulloch y W. Pitts [49] como simulación de la actividad nerviosa, y, poco después, sirvió de ejemplo para los modelos posteriores de M. Minsky [51] y F. Rosenblatt [76], entre otros.

Las redes neuronales constituyen una herramienta de análisis estadístico que permite la construcción de un modelo de comportamiento a partir de una base de ejemplos de dicho comportamiento. La red neuronal, completamente *ignorante* al principio, efectúa un aprendizaje partiendo de los ejemplos para transformarse o evolucionar, a través de una serie de modificaciones sucesivas, en un modelo susceptible de predecir el comportamiento futuro del sistema simulado.

Esta capacidad para aprender todo aquello que tenga un cierto sentido (*aproximador universal*) ha sido establecida de manera rigurosa (*teorema de Kolmogorov*), por lo que las redes neuronales son consideradas hoy día como una herramienta de gran rigor cuyas bases teóricas han sido debidamente justificadas.

Una vez construida la red neuronal, se obtiene un modelo *a la medida* que actúa en función de lo que percibe. Si existe una correspondencia de causa–efecto entre las descripciones introducidas y los valores a prever, el modelo extraerá dicha relación para aplicarla en los casos sucesivos. Además, el modelo obtenido es robusto, en el siguiente sentido: la aparición de ejemplos no coherentes en la base de ejemplos es

reconocida por la red y no influyen en las conclusiones extraídas.

En relación con la similitud que presenta la red neuronal con la realidad biológica, se suele dar la siguiente clasificación:

- *Modelos de tipo biológico*, que tratan de simular los sistemas neuronales biológicos (y, más recientemente, funciones auditivas y algunas funciones básicas de la visión). Su objetivo principal es desarrollar un modelo para verificar hipótesis relativas a sistemas biológicos.

- *Modelos dirigidos a la aplicación*, que, en general, no presentan similitud con los sistemas biológicos sino que se diseñan atendiendo a las necesidades del problema que pretenden resolver.

Este modelo no se ha extendido hasta hace unos años por cuestiones de capacidad, ya que es ahora cuando se ha llegado a conseguir la potencia de cálculo necesaria para su aplicación práctica: los minutos necesarios para aplicar un aprendizaje en un ordenador actual equivalen a días en un ordenador de la década de los 70.

1.6.2. Computación Molecular

Como ya se ha comentado, a finales de la década de los cincuenta, el premio nobel R.P. Feynman [24] describe los ordenadores *sub-microscópicos* e introduce el concepto teórico de *computación a nivel molecular*, postulándolo como una innovación necesaria y revolucionaria en la carrera por la miniaturización. Las ideas de Feynman adquieren una especial relevancia a partir de 1983, cuando R. Churchhouse establece las limitaciones físicas de la velocidad de cálculo de un procesador convencional, demostrando que, bajo los principios de la física, existe una cota para la velocidad y el tamaño que los microprocesadores pueden alcanzar. Esta cota, además, impediría resolver con las técnicas actuales problemas que en la actualidad se consideran intratables, por precisar un tiempo muy elevado para resolver ejemplares de tamaño relativamente *grande*, imponiendo que por mucho que aceleremos los microprocesadores (incluso alcanzando dicha cota física) seguiríamos teniendo instancias de esos problemas que precisarían años o siglos para poder resolverlos en esas supermáquinas.

En 1987, T. Head [30] propone el primer modelo computacional abstracto basado en la manipulación de las moléculas de ADN, *el modelo splicing*. En este modelo la información es almacenada en cadenas de caracteres al modo en que lo hacen las moléculas de ADN, y las operaciones que se pueden realizar sobre dichas cadenas son similares a las que realizan ciertas enzimas sobre el ADN.

L.M. Adleman materializa esta similitud en noviembre de 1994 [1] mostrando que es posible usar procesos biológicos para atacar la resolubilidad de instancias de ciertos problemas matemáticos especialmente *difíciles*: mediante un experimento realizado en el laboratorio consiguió resolver una instancia concreta de un problema *computacionalmente intratable* usando técnicas de biología molecular para la manipulación del ADN. Aunque el experimento de Adleman no es propiamente una implementación práctica del modelo que diseñó T. Head, el tipo de sustrato utilizado (moléculas de ADN) así como las operaciones que usa sobre dicho sustrato son similares a las propuestas por el modelo splicing.

En julio de 2000, un equipo de científicos de la Universidad de California desarrolló un interruptor del tamaño de una millonésima de milímetro (un nanómetro), a partir de una molécula. Todo parece indicar que este interruptor puede representar una alternativa revolucionaria en relación con los actuales chips de silicio.

- En su funcionamiento sustituye la electricidad por una reacción química, lo que representa un importante ahorro en el consumo de energía.
- Estos nuevos interruptores podrían disponer de más de mil procesadores en el espacio ocupado hoy día por un sólo procesador (los actuales chips de silicio tienen una altura aproximada de cinco mil nanómetros).
- Se estima que estos interruptores podrían aumentar la velocidad de procesamiento de la información cien mil millones de veces la de un ordenador convencional, y podrían reproducir la capacidad equivalente a cien ordenadores convencionales en el tamaño de un grano de sal fina.

En la actualidad han surgido modelos moleculares alternativos al propuesto por T. Head. Algunos de esos modelos utilizan el ADN como molécula básica y la diferencia entre dichos modelos estriba en las distintas operaciones que se consideran primitivas o elementales. Otros modelos utilizan diferentes tipos de moléculas biológicas como dispositivo para almacenar la información (como el ARN) y enzimas específicas para su tratamiento. Ahora bien, todos esos modelos tienen como denominador común el uso de moléculas estructuralmente complejas que han demostrado su eficacia en la naturaleza, tanto en el almacenamiento de información como en la diversidad y potencia de las operaciones que se pueden realizar sobre ellas.

Los modelos moleculares constituyen una rama de la Computación en la que intervienen áreas del conocimiento humano muy distintas y, tradicionalmente, disjuntas, que van desde las disciplinas más abstractas de las matemáticas hasta las aplicaciones más novedosas que se puedan obtener en los laboratorios de biología

molecular. Esta característica hace que muchos de los avances en la creación de modelos eficientes pase ineludiblemente por la consecución de nuevas, eficientes y robustas técnicas de manipulación de las moléculas con las que se trabaja en el laboratorio.

1.6.3. Computación celular con Membranas

En las células tienen lugar una serie de reacciones químicas que provocan un transformación de las componentes químicas presentes en sus membranas, junto con un flujo de las mismas entre los distintos compartimentos que la integran. Estos procesos a nivel celular pueden ser interpretados como procedimientos de cálculo.

La *Computación celular con Membranas* ha sido hasta ahora el último modelo de Computación Natural. Fue introducido por Gh. Păun en octubre 1998 [56] como un modelo de tipo distribuido y paralelo, y está inspirado en el funcionamiento de la célula como organismo vivo capaz de procesar y generar información.

Los ingredientes básicos de un *P sistema* (que es como también se conocen los sistemas que utilizan la computación celular con membranas) son la *estructura de membranas*, que consiste en un conjunto de membranas (al modo de las vesículas que componen las células) incluidas en una *piel* exterior que las separa del entorno, junto con ciertos *multiconjuntos de objetos* (es decir, conjuntos en los que los elementos pueden aparecer repetidos) situados en las *regiones* que delimitan dichas membranas (al modo de los compuestos que hay en el interior de dichas vesículas). Estos objetos pueden transformarse de acuerdo con unas *reglas de evolución* que son aplicadas de una forma no determinista, paralela y maximal (al modo de las reacciones que se pueden producir entre dichos compuestos). Para simular la permeabilidad de las membranas celulares, las reglas de evolución no sólo pueden modificar los objetos presentes en una membrana, sino que pueden pasar a través de dos regiones adyacentes *atravesando* la membrana que las separa.

Este modelo de computación implementa un paralelismo masivo en dos niveles básicos: en un primer nivel, cada membrana aplica sus reglas de forma paralela sobre los objetos presentes en ella, produciendo los nuevos objetos y comunicándolos a las membranas adyacentes si procediera; en un segundo nivel, todas las membranas realizan esta operación en paralelo, trabajando simultáneamente, sin interferencia alguna de las operaciones que se estén produciendo en las demás membranas del sistema.

Desde la aparición de los primeros P sistemas han sido muchas las variantes introducidas buscando unas veces mayor eficiencia en la solución de problemas com-

plejos, y otras una mayor aproximación al modelo biológico real que trata de simular [58]. Entre las diversas variantes que se consideran en los P sistemas destacan las siguientes:

- El uso de cadenas de un determinado alfabeto como objetos básicos del modelo (al modo de las moléculas de ADN, ARN o de las proteínas en el interior de ciertas membranas), en lugar de considerar objetos atómicos sin estructura interna.
- La posibilidad de disolver, crear o duplicar membranas, y el uso de catalizadores (como objetos necesarios para la ejecución de una regla, pero que permanecen inalterables tras la ejecución de la misma).
- P sistemas que sólo admiten comunicación entre membranas pero no la generación o transformación de los objetos que atraviesan las mismas.

Además, se prueba que muchos de los modelos que se obtienen a partir de los nuevos conceptos son computacionalmente completos; es decir, que el modelo que se obtiene posee la misma potencia computacional que una máquina de Turing.

A diferencia de los modelos de Computación Natural reseñados anteriormente, la Computación Celular con Membranas no dispone en la actualidad de ninguna implementación real, ya sea en el laboratorio (como en el caso de la computación molecular) o bien a través de una adaptación en ordenadores convencionales (como en el caso de los algoritmos genéticos y las redes neuronales). Hasta el momento, todo lo realizado en esta dirección se reduce a una interpretación como P sistemas de las redes de ordenadores convencionales (por ejemplo, *internet*), o de los procesos de ciertas reacciones químicas que se producen en medio acuoso [31], y a una simple simulación de ejecuciones de los P sistemas a través de lenguajes de programación convencionales.

Capítulo 2

Computación molecular basada en ADN

A principios del siglo XX se descubre que los *cromosomas* (descritos por Hofmeister en 1848 y cuya denominación se debe a Waldeyer, 1888) están relacionados directamente con los mecanismos de la herencia, y son los portadores del material genético que determina las características de la descendencia en las especies. Entre 1943 (Claude, Porter) y 1947 (Mirsky) se empieza a conocer la estructura de los cromosomas, comprobándose que están compuestos, básicamente, por proteínas y ADN (ácido desoxirribonucléico). Durante estos años, y debido a la mayor complejidad en la estructura molecular de las proteínas, se tiene el convencimiento de que éstas deben ser las encargadas de transportar la información genética de padres a hijos.

Los trabajos de J. Watson y F. Crick de principios de la década de los cincuenta (entre 1951 y 1953) echarían por tierra el papel relevante atribuido a las proteínas en relación con la herencia. Watson y Crick

- descifran la estructura molecular del ADN;
- descubren el principio de complementariedad así como el direccionamiento o polaridad de dichas moléculas;
- demuestran que las moléculas de ADN codifican toda la información genética de los organismos vivos; y
- justifican la posibilidad de usar ciertas técnicas para su manipulación en el laboratorio.

A continuación estudiaremos brevemente la estructura del ADN, así como algunas de las operaciones que se pueden realizar en el laboratorio con cadenas del citado ácido.

2.1. Estructura del ADN

Tradicionalmente la molécula de ADN se ha representado como una molécula de estructura uniforme con apariencia de doble hélice (en forma de *escalera de caracol*), tal y como la describieron Watson y Crick inicialmente. Sin embargo, en los últimos 15 años los avances técnicos han permitido desterrar esta idea de uniformidad en su estructura, mostrando que se pueden producir variantes mucho más complejas. Pese a que es imprescindible conocer a fondo dicha complejidad para poder realizar implementaciones eficientes, y debido al uso que de las moléculas vamos a hacer, nos centraremos en la idea tradicional y mantendremos en mente la visión de una molécula con forma de doble hebra helicoidal (habitualmente, las estructuras más complejas en las moléculas de ADN se producen cuando se aumenta la longitud de las mismas).

El primer dato que llama la atención es que esta estructura helicoidal no responde tan solo a un resultado más o menos aleatorio de la naturaleza, sino que juega un papel importante en la protección de la información que va *escrita* en la molécula. Gracias a esta estructura, dicha información, codificada a través de la sucesión de bases nitrogenadas que conforman la molécula, queda hacia el interior (en la *zona media* de los peldaños de la escalera), de forma que las paredes exteriores de la hélice la protegen de posibles modificaciones que pudiesen producirse por reacciones con el entorno. El conjunto de enlaces de hidrógeno que mantienen unida la doble hélice son puntualmente lo suficientemente débiles como para no precisar de mucha energía en su ruptura, pero globalmente dotan a la molécula de una estructura que resulta muy estable y robusta.

A lo largo de millones de años, en el proceso evolutivo de los organismos vivos, se han desarrollado complejos procedimientos biológicos que permiten modificar, de forma controlada, la estructura interna de las moléculas de ADN. Muchas proteínas son capaces de reconocer diversos puntos dentro de las largas cadenas de ADN, quedando ancladas a ellos y provocando el comienzo de operaciones tan complejas como son la separación o unión de la doble hebra, copia de su totalidad o trozos específicos, modificación de alguna (o algunas) base concreta, etc.

Para comprender la estructura del ADN es preciso conocer los componentes in-

dividuales que lo forman. El ácido desoxirribonucleico (ADN) es un polímero que, en su estructura lineal, consta de una serie de monómeros denominados *desoxirribonucleótidos* (que llamaremos brevemente *nucleótidos*). A su vez, cada nucleótido consta de (veáse figura 2.1) :

(a) Un azúcar (*desoxirribosa*) que tiene cinco átomos de carbono enumerados del 1' al 5' , y que en el carbono 4' tiene un grupo hidroxilo (OH).

(b) Un grupo fosfato (P), unido al azúcar por el carbono 5'.

(c) Una base nitrogenada, unida al azúcar por el carbono 1'.

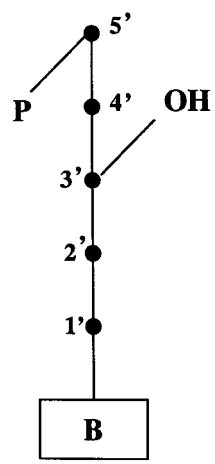


Figura 2.1. Estructura esquemática de un nucleótido

Generalmente se identifica cada nucleótido con la base nitrogenada que contiene. Existen cuatro tipos de bases nitrogenadas: *adenina*, *citosina*, *guanina* y *timina*, que se representan por las iniciales correspondientes: A, C, G, T. La adenina y la guanina pertenecen al grupo de las *purinas*, mientras que la citosina y la timina pertenecen al grupo de las *piramidinas*.

Los nucleótidos pueden enlazarse de dos maneras diferentes:

(a) Mediante un *enlace fosfodiester* (covalente): el grupo fosfato 5′ de un nucleótido se une al grupo hidroxilo 3′ de otro (ver la figura 2.2).

(b) Mediante un *enlace de hidrógeno* que se realiza a través de las bases nitrogenadas (ver la figura 2.3).

El enlace fosfodiester entre nucleótidos permite la construcción de *cadenas simples* de ADN, como se muestra en la figura 2.2, que poseen dos extremos de comportamiento muy diferente tanto química como biológicamente. Entre otras propiedades,

la diferencia existente entre ambos extremos proporciona una *polaridad* o *direccionamiento* a las moléculas de ADN; así hablaremos de la dirección $5' \rightarrow 3'$ o la dirección $3' \rightarrow 5'$. A la hora de trabajar con cadenas simples, si no se explicita su dirección se entenderá que ésta es $5' \rightarrow 3'$.

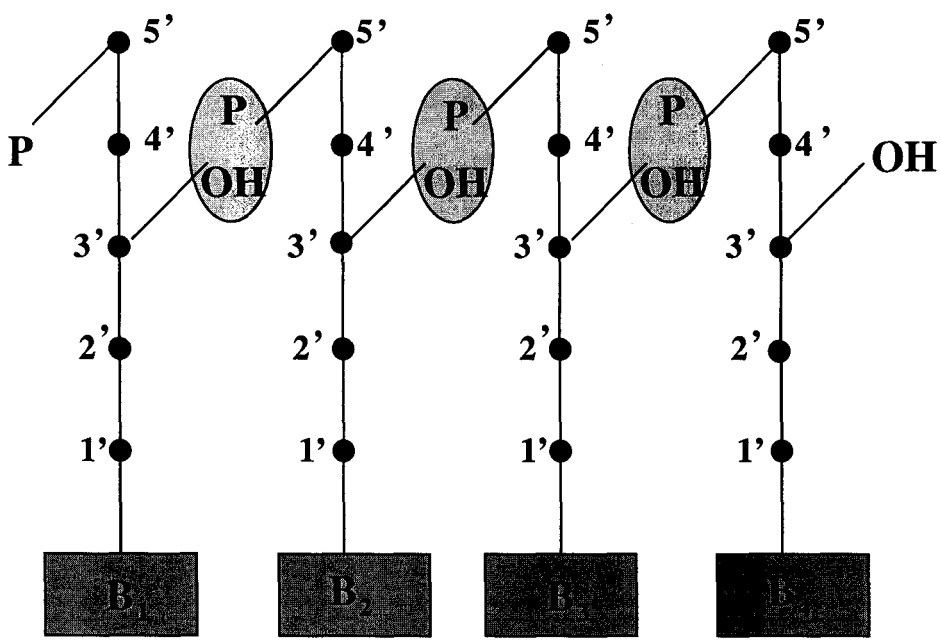


Figura 2.2. Enlace fosfodiester: cadena simple $5'$–$B_1B_2B_3B_4$

El enlace de hidrógeno es más débil que el fosfodiester y se rige por el principio de complementariedad: la adenina sólo se puede enlazar con la timina (y recíprocamente), y la citosina sólo con la guanina (y recíprocamente). Más aún, el enlace entre la adenina y la timina se establece a través de dos puentes de hidrógeno mientras que entre la citosina y la guanina se producen tres puentes de hidrógeno, lo cual proporciona un poco más de fuerza a este último enlace.

Combinando enlaces fosfodiester y enlaces de hidrógeno se obtienen *cadenas dobles* de ADN (véase la figura 2.3); es decir, *doble hebras* que se disponen espacialmente formando la estructura conocida como *doble hélice*: dos cadenas simples están alineadas de forma *antiparalela* (es decir, una con la dirección $5' \rightarrow 3'$ y la otra con dirección $3' \rightarrow 5'$) y unidas por enlaces de hidrógeno; los nucleótidos unidos por enlaces fosfodiester, con los grupos fosfatos orientados hacia el exterior de la hélice y las bases nitrogenadas próximas al centro, lo que proporciona una gran estabilidad a la molécula.

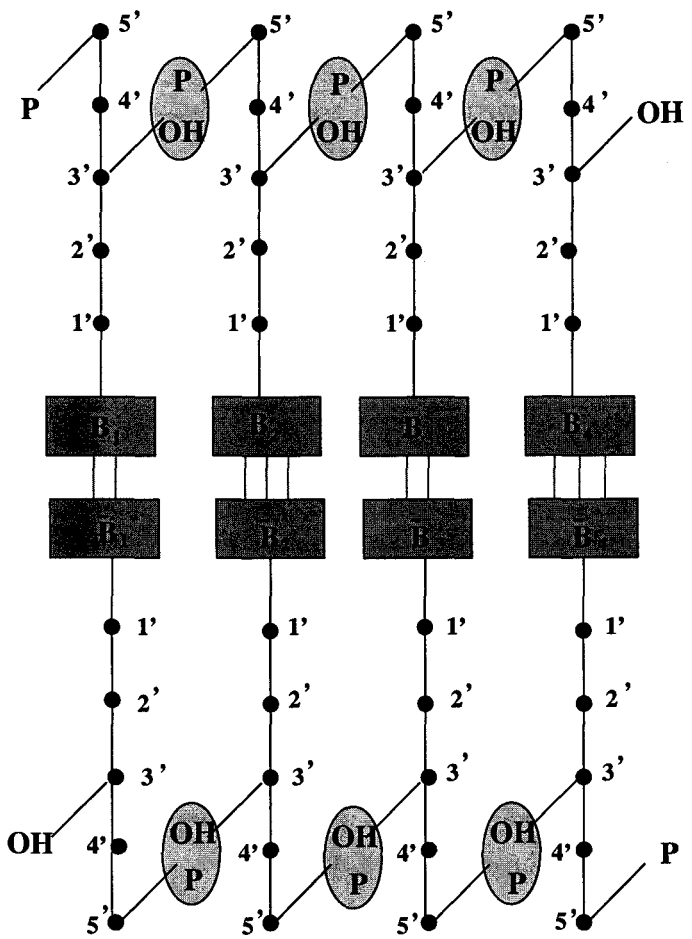


Figura 2.3. Enlace fosfodiester + Enlace de hidrógeno: cadena doble

Si γ es una cadena simple, notaremos $\overline{\gamma}$ la cadena formada por los nucleótidos complementarios (las cadenas complementarias se expresarán, en cambio, en la dirección $3' \rightarrow 5'$, salvo que se explicite lo contrario).

2.1.1. Operaciones con cadenas de ADN

La complejidad estructural del ADN, así como la diversidad en la formación de estas moléculas a partir de las distintas combinaciones de las bases nitrogenadas, va acompañada de una amplia gama de operaciones que sobre ellas se realizan en los seres vivos y que, en los últimos años, se han podido reproducir en el laboratorio. A continuación describimos, sin entrar en mucho detalle, algunas de estas operaciones, que constituyen la base tanto de la ingeniería genética como de la computación molecular basada en ADN. Para mayor detalle se remite al capítulo 0 de [62] y al capítulo 1 de [82].

Desnaturalización y Renaturalización

La estructura en doble hélice del ADN es especialmente sólida. La estabilidad de la doble hebra proviene, como hemos comentado anteriormente, de dos tipos de fuerzas químicas, los enlaces de hidrógeno y los enlaces fosfodiester, así como en la distribución espacial de las componentes. Además, la hélice, en estado natural, se encuentra cubierta por moléculas de agua que forman lo que podría considerarse como una especie de *escudo*.

El proceso biológico por el cual se rompen los enlaces de hidrógeno de una doble hebra dando lugar a dos cadenas simples de ADN, se denomina *desnaturalización* (*melting* o *reannealing*). Este proceso se puede simular en el laboratorio calentando la solución en la que estén las doble hebras de ADN, hasta una temperatura comprendida entre 85 y 94 grados centígrados.

El proceso inverso, denominado *renaturalización* (*annealing*), se consigue sometiendo la solución con cadenas simples complementarias a un enfriado lento hasta aproximadamente los 55 grados centígrados, a fin de permitir que las bases nitrogenadas complementarias se vayan uniendo a través de los correspondientes enlaces de hidrógeno.

Medida de la longitud de una molécula

La longitud de una cadena simple se define como el número de bases nitrogenadas que contiene, y se expresa en *mer*. La longitud de una cadena doble se define como el número de pares de bases nitrogenadas complementarias que contiene y se expresa en *bp* (pares de bases).

La técnica más usual que se utiliza para medir una molécula de ADN se denomina *electroforesis en gel*. El procedimiento se basa en el hecho de que toda molécula de ADN está dotada de una carga eléctrica negativa que es proporcional a su longitud. Además, la fuerza necesaria para desplazarla debe ser, igualmente, proporcional a su longitud. Por tanto, si un conjunto de moléculas de ADN es sometido a un campo eléctrico en una solución ideal, todas las moléculas se desplazarán hacia el electrodo positivo a igual velocidad, con independencia de la longitud que posean. Una forma de discriminar las moléculas entre sí, en función de su longitud, consiste en sustituir la solución ideal por un gel, que provoca un rozamiento mayor en las moléculas de más longitud. Para ello:

- Se coloca un gel adecuado en disolución y se ubica en un recipiente rectangular (habitualmente se consigue añadiendo al disolvente gel en polvo y calentando la disolución).

- Con la ayuda de un instrumento en forma de peine, y durante el proceso de enfriado, se realizan una serie de ranuras en el gel.
- En las distintas ranuras formadas por el peine se colocan las moléculas de ADN que se quieren medir.
- Se activa el campo eléctrico, situando el electrodo positivo en el lado opuesto a las ranuras, desactivándolo cuando la primera molécula llegue a él.

La longitud de cada molécula se calcula a partir de la distancia recorrida durante el proceso y el tiempo transcurrido.

Extracción de moléculas

Es posible seleccionar de un tubo que contiene en disolución cadenas simples de ADN, todas aquellas moléculas que contienen como subcadena una cierta cadena prefijada. Para realizar esta operación, dado un tubo de ensayo, T, que contiene una solución con cadenas simples de ADN, y γ, una cadena simple de ADN prefijada, se utiliza la técnica de las *sondas metálicas*, que consiste en lo siguiente:

- Se consideran unas microesferas de hierro que tienen adheridas la cadena $3'-\overline{\gamma}$, y se introducen en el tubo T.
- Se somete la solución a un proceso de renaturalización.
- Se coloca un imán a un lado del tubo, T, y se vierten en otro recipiente, T_1, las moléculas *no adheridas* a $3' - \overline{\gamma}$.
- Se retira el imán, se añade nuevo disolvente y se somete la solución a un proceso de desnaturalización.
- Se vuelve a colocar un imán a un lado del tubo, T, y se vierte en otro recipiente, T_2, las moléculas *no adheridas* a $3' - \overline{\gamma}$.
- El tubo T_1 estará formado por las moléculas del tubo inicial que **no** contienen a γ como subcadena, y T_2 por aquellas que **sí** contienen a γ como subcadena.

Alargar y copiar una cadena de ADN

La *ADN polimerasa* es la enzima responsable de la replicación del ADN, sintetizando una nueva cadena simple usando otra como patrón. Se suele decir que la polimerasa es la enzima de la vida. En cierto sentido podemos considerar que esta

enzima implementa la complementariedad de Watson–Crick. Para ello, se desliza sobre la cadena patrón, leyendo los nucleótidos y escribiendo de acuerdo con una cierta ley. Podemos resaltar la similitud entre la acción de esta enzima y la ejecución de una máquina de Turing: la cabeza de trabajo de la máquina se desliza a través de la cinta, va leyendo las casillas y escribe sobre la misma según le dicta la función de transición de la máquina, que también indica cómo debe continuar desplazándose. L. Adleman se percató de esta analogía e intentó simular una máquina de Turing a través de moléculas de ADN y de la enzima polimerasa. Tras fracasar en el intento (conseguido por Ehud Shapiro en noviembre de 2001), L. Adleman trató de resolver una instancia de un problema **NP**-completo mediante la manipulación de moléculas de ADN. Esta vez realizó con éxito el experimento.

Para copiar una cadena simple de ADN necesitamos tres elementos: (a) un *molde* o *patrón*, que es una cadena simple que codifica por complementariedad la cadena a copiar; (b) un *cebador* (*primer*), que suele ser una cadena corta enlazada al molde por el extremo 5'; y (c) nucleótidos suficientes en la solución. La polimerasa se activa cuando llega al extremo 3′ del cebador. El proceso de copiado se realiza en la dirección $5' \rightarrow 3'$, que se denomina *dirección de la vida.*

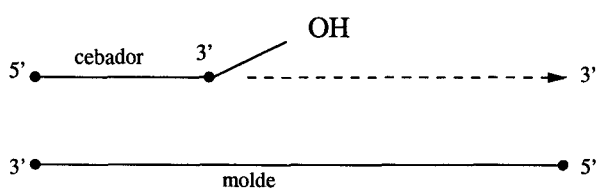


Figura 2.4. Acción de la polimerasa

Síntesis de una cadena de ADN

Esta operación permite fabricar cadenas (simples o dobles) de ADN a partir de una secuencia prefijada. La síntesis de una cadena simple se realiza en la dirección $3' \rightarrow 5'$. El primer nucleótido se adhiere a un soporte sólido por el extremo 3′ y se van añadiendo nucleótidos por el extremo 5′. Una cadena corta sintética de nucleótidos recibe el nombre de *oligonucleótido*, o más brevemente, *oligo*. Para sintetizar una doble hebra, en primer lugar se sintetiza la hebra $3' \rightarrow 5'$ y se considera un cebador; después se usa la polimerasa para completar la doble hebra tal y como se ha descrito en el apartado anterior. En la actualidad, este proceso puede realizarse de forma automática.

Alteración de nucleótidos en una cadena de ADN

Hay un tipo de enzimas, llamadas *de restricción*, que reconoce lugares determinados dentro de las cadenas de ADN, a las que se adhieren. De esta forma, dichas enzimas permiten la activación local de ciertos procesos moleculares que producen una desestabilización en las cadenas hasta el punto de provocar su ruptura. Existen otro tipo de enzimas, denominadas *de modificación*, que alteran las cadenas de ADN añadiendo o quitando oligos a las mismas. Este tipo de enzimas suele actuar en colaboración con una enzima de restricción que tiene asociado el mismo lugar de reconocimiento. Las enzimas de modificación son útiles para controlar algunas operaciones con ADN, especialmente aquellas que utilizan los organismos vivos para protegerse de ataques externos. Por ejemplo, para defenderse de un cierto virus, una determinada bacteria cuenta con una enzima de restricción cuyo objetivo es destruir cadenas de ADN del virus. Dicha enzima trabaja en colaboración con una de modificación, que tiene el mismo lugar de reconocimiento, y cuya misión consiste en alterar temporalmente los lugares correspondientes en las cadenas de ADN de la bacteria a fin de que ésta no quede afectada por la acción de su propia enzima de restricción, consiguiendo así destruir únicamente el material genético del virus. Posteriormente la enzima de modificación se encarga de restaurar los lugares de reconocimiento alterados.

Degradación de cadenas de ADN

Las enzimas *nucleasas* tienen la capacidad de destruir los enlaces fosfodiester. Estas enzimas se pueden clasificar en dos tipos:

- Las *exonucleasas*, que eliminan nucleótidos de una doble hebra, a la vez uno de cada extremo y en la dirección $3' \rightarrow 5'$. Estas enzimas son más flexibles que las polimerasas: algunas variedades sólo eliminan nucleótidos del extremo $5'$ y otras sólo del extremo $3'$. El resultado de una enzima de este tipo es un acortamiento de la cadena sobre la que actúa.

- Las *endonucleasas* pueden cortar cadenas (simples o dobles) de ADN por cualquier sitio. No obstante, existe una variedad (las *endonucleasas de restricción*) que sólo pueden cortar doble hebras por un sitio específico (su lugar de reconocimiento). Este tipo de enzimas está especializado según el lugar de reconocimiento y la forma de realizar el corte. Si existen varios lugares de reconocimiento en una doble hebra, entonces la endonucleasa de restricción realizará un corte por cada uno de ellos.

Rellenado de cadenas de ADN

Al cortar una doble hebra utilizando enzimas nucleasas se producen una serie de voladizos que, en determinadas condiciones, pueden volver a enlazarse entre sí. En este caso, así como en otros procesos, pueden aparecer unos pequeños huecos entre nucleótidos consecutivos de una cadena simple de ADN. La *ligasa* es una enzima que permite rellenar dichas hendiduras restaurando la fortaleza del enlace covalente y la estabilidad de la molécula.

Copias múltiples (amplificación) de cadenas de ADN

La técnica de la *reacción en cadena de la polimerasa* (PCR) fue descubierta por K. Mullis en 1985 y permite generar gran cantidad de copias de una cadena molde o patrón. Esta técnica es muy eficiente y tiene una gran cantidad de aplicaciones en ingeniería genética, análisis del genoma, diagnosis clínica, arqueología, paleontología, etc.

La técnica de la reacción en cadena de la polimerasa funciona como sigue:

- Supongamos que disponemos de una doble hebra, α, de la que queremos realizar copias idénticas, y cuyos bordes $3'$ son β (" borde superior") y γ ("borde inferior").

- En la fase inicial se prepara una solución que contiene la molécula α, oligos $3'-\overline{\beta}$, $5'-\overline{\gamma}$, nucleótidos en cantidades elevadas, y la enzima polimerasa (debido a que en un paso posterior se efectuará un calentamiento de la disolución, se usa una variedad de la polimerasa resistente a altas temperaturas, para que la energía proporcionada a la disolución no destruya los enlaces propios de la enzima).

- La fase de ejecución propiamente dicha consta de una serie de ciclos cada uno de los cuales comprende tres pasos:

  (a) *Desnaturalización*: se calienta la solución hasta una temperatura cercana a la de ebullición con el fin de romper los enlaces de hidrógeno.

  (b) *Renaturalización*: mediante un enfriado lento se producen enlaces entre los bordes y los oligos proporcionados inicialmente.

  (c) *Extensión*: se calienta nuevamente la solución hasta unos 72 grados con el fin de que la polimerasa extienda los bordes, reconstruyendo la cadena de ADN original por medio de la complementariedad de las bases.

- En cada uno de los ciclos anteriores se duplica el número de doble hebras α; por tanto, al reiterar este proceso n veces se obtienen 2^n copias de la molécula patrón.

Lectura de una cadena de ADN

El Proyecto Genoma Humano, iniciado en 1990 y coordinado por instancias públicas de E.E.U.U., tenía como objetivo descifrar antes del año 2003 la secuenciación correcta del ADN humano. Craig Venter, un científico heterodoxo que participaba en dicho Proyecto, creó en 1998 una empresa privada (PE Celera Genomics) en la que consiguió descifrar, en el año 2000, unos 3.120 millones de pares de nucleótidos que componen el ADN humano (aún quedan por transcribir unos 40.000 "huecos" y verificar varias veces los resultados de Venter, a fin de elevar a definitivo los resultados provisionales obtenidos). El trabajo consistió en la lectura correcta de una cadena de ADN extraordinariamente grande. Existen distintas técnicas en la actualidad para poder realizar la lectura de una cadena simple de ADN. A continuación describimos (sin entrar en detalles) una de ellas.

Se consideran ciertos nucleótidos que han sido modificados levemente, denominados *nucleótidos análogos*, que se denotan por ddA, ddC, ddG, ddT, asociados a los nucleótidos A, C, G, T, respectivamente. La operación de lectura de una cadena simple de ADN, α, se realiza como sigue:

- Se extiende α por el extremo $3'$ de acuerdo con un molde γ, formando una cadena simple β.
- Se preparan cuatro tubos (T_A, T_C, T_G, T_T) conteniendo cadenas β, oligos $5' - \overline{\gamma}$, polimerasa y nucleótidos en suficiente cantidad. Además, en el tubo T_X se añadirán nucleótidos análogos del tipo ddX, para cada $X \in \{A, C, G, T\}$.
- A partir de la cadena simple β y de los oligos $5' - \overline{\gamma}$, en cada tubo T_X se obtienen doble hebras parciales β'.
- En el tubo T_X, la polimerasa completa las doble hebras parciales β'. No obstante, en este proceso aparecerán doble hebras parciales no completas cuando se elija el nucleótido análogo ddX en lugar de X.
- Se somete la solución a un proceso de desnaturalización y se extraen de la solución las cadenas simples que contienen a $5' - \overline{\gamma}$.
- Se colocan las cadenas extraídas en una solución y se ordenan según su longitud, mediante la técnica de la electroforesis en gel.

- Por último, se leen los nucleótidos del extremo 3′ de las cadenas ordenadas: la cadena complementaria será α.

2.2. Los primeros experimentos moleculares

En la sección anterior se ha estudiado brevemente la estructura molecular del ADN así como algunas operaciones que se pueden realizar con dichas moléculas, gracias a las propiedades que se derivan del direccionamiento y del principio de complementariedad. En esta sección vamos a analizar los primeros experimentos prácticos realizados en el laboratorio (*in vitro*) que dieron lugar al nacimiento de la computación molecular basada en ADN como disciplina de estudio. Conviene recordar que, si bien existían resultados anteriores relativos a la potencia teórica del ADN como sustrato idóneo para la materialización de lo que debería ser la computación molecular, los experimentos de L. Adleman y R.J. Lipton constituyen las primeras demostraciones prácticas de su capacidad.

2.2.1. El experimento de Adleman

En noviembre de 1994, L. Adleman [1], matemático y doctor en Informática, resolvió una instancia concreta de un problema **NP**–completo (el *problema del camino hamiltoniano*, en su versión dirigida y con un par de nodos distinguidos [26]) a través de la manipulación de moléculas de ADN usando técnicas de biología molecular.

Dado un grafo G (dirigido o no), se dice que un *camino* de G es *hamiltoniano* si es simple (es decir, todas las aristas y todos los vértices del camino son distintos entre sí excepto, quizás, el primero y el último) y, además, pasa por todos y cada uno de los vértices de G.

El **problema del camino hamiltoniano** en su versión dirigida y con dos vértices distinguidos se formula como sigue: *Dado un grafo dirigido, $G = (V, E)$, y dos nodos distinguidos v_i, v_f, determinar si existe un camino hamiltoniano de G que va desde v_i hasta v_f.*

L. Adleman resolvió el problema anterior para el grafo concreto con siete vértices descrito en la figura 2.5, en donde los nodos distinguidos son 0 y 6, respectivamente.

Se puede comprobar fácilmente que dicha instancia posee un único camino hamiltoniano: el que, partiendo del vértice etiquetado por 0, pasa sucesivamente por los vértices $1, 2, 3, 4, 5$ hasta llegar al vértice 6. Esta solución puede ser obtenida por una persona en unos segundos y por un ordenador en menos de una millonésima de segundo.

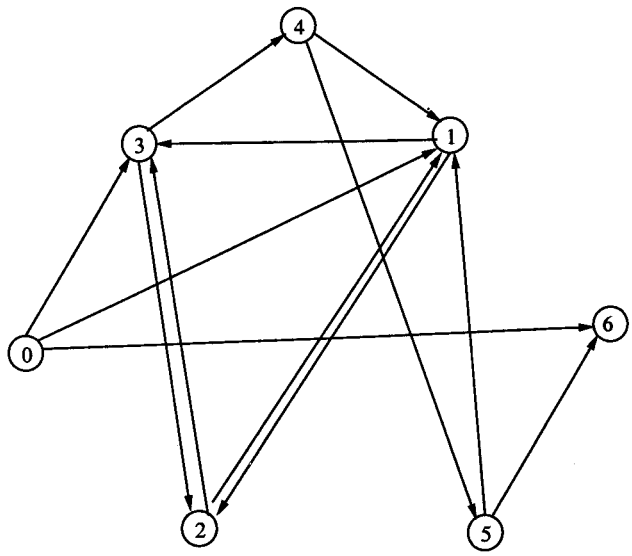


Figura 2.5 Grafo usado en el experimento de Adleman

Sin embargo, el experimento de Adleman necesitó siete días para su realización en el laboratorio. A pesar de ello, y analizando un poco más en profundidad cómo se materializó, podemos extraer las siguientes conclusiones acerca de dicho experimento:

- Representa el primer ejemplo práctico de computación a nivel molecular, en la línea esbozada por R. Feynman.
- Muestra por primera vez el uso potencial de moléculas de ADN como estructuras de datos físicas, susceptibles de ser usadas como medio de almacenamiento. En ellas destaca la extraordinaria densidad de información que son capaces de almacenar y el hecho de ser manipulables debido a la complementariedad de Watson-Crick.
- Ilustra la posibilidad de usar moléculas de ADN para resolver instancias de problemas de tipo combinatorio computacionalmente intratables.
- Muestra la capacidad que tienen las moléculas de ADN para simular computaciones de forma masivamente paralela, gracias a la acción simultánea de las enzimas sobre las cadenas de ADN en disolución, tal y como ocurre en el interior de los organismos vivos.

El problema del camino hamiltoniano se puede resolver de manera trivial a través del siguiente algoritmo de búsqueda exhaustiva, de fuerza bruta:

Entrada: $G = (V, E)$, grafo dirigido; v_i y v_f nodos distinguidos.

1. Generar todos los caminos de G.
2. Rechazar los caminos que no empiezan por v_i y terminan en v_f.
3. Rechazar los caminos que no contienen exactamente $|V|$ nodos.
4. Para cada $u \in V$, rechazar los caminos que no pasan por u.

Salida: **SI**, si queda algún camino; **NO** en caso contrario.

Evidentemente, cada uno de los pasos del algoritmo anterior conlleva un número elevado de operaciones que, debidamente explicitadas, pueden reflejar un número exponencial de pasos simples. El experimento de Adleman, realiza básicamente una implementación directa de este algoritmo. A grandes rasgos, procede como sigue:

- Se codifican los vértices y los arcos del grafo mediante oligos, de tal manera que los caminos del grafo se pueden obtener a partir de dichos oligos realizando operaciones de naturalización y desnaturalización.
- A partir de un tubo de ensayo inicial que contiene moléculas de ADN que codifican cualquier posible camino de G, se van seleccionando moléculas de acuerdo con los criterios indicados en cada uno de los pasos del algoritmo. Al final de todos estos pasos, las moléculas de ADN que permanezcan en el tubo de salida codificarán caminos hamiltonianos del grafo que van desde el nodo v_i hasta el nodo v_f.

Implementación de los pasos del algoritmo

A continuación, veamos con detalle la implementación que L. Adleman realizó en el laboratorio, de cada uno de los pasos del algoritmo de búsqueda exhaustiva descrito anteriormente.

Evidentemente, la implementación del paso 1 plantea un problema en sí mismo: generar todos los caminos de un grafo dirigido. En la solución dada en el experimento de Adleman este paso se resuelve elaborando un tubo de ensayo inicial que contenga moléculas que codifican cualquier posible camino del grafo considerado.

Para ello, a cada nodo del grafo, i ($0 \leq i \leq 6$), se le asocia un oligo, s_i, de longitud 20 mer (la longitud elegida depende realmente del número de elementos a codificar, debiendo ser suficiente para que no se produzcan enlaces no deseables, y, a la vez, lo más pequeña posible para reducir tiempo de ejecución y facilitar su creación). Para

cada vértice, i, se designa por s'_i (respectivamente, s''_i) el oligo formado por las diez primeras (respectivamente, últimas) bases de s_i.

Además de codificar en el tubo de entrada los vértices del grafo, también se necesita codificar información acerca de cómo llegar hasta un vértice a partir de otro. Para ello, a cada arco (i,j) del grafo se le asocia el siguiente oligo, e_{ij}:

$$e_{ij} = \begin{cases} s''_i s'_j & \text{si } i \neq 0 \ \wedge \ j \neq 6 \\ s_0 s'_j & \text{si } i = 0 \ \wedge \ j \neq 6 \\ s''_i s_6 & \text{si } i \neq 0 \ \wedge \ j = 6 \\ s_0 s_6 & \text{si } i = 0 \ \wedge \ j = 6 \end{cases}$$

Para formar todos los posibles caminos que se pueden dar en el grafo G, se parte de un tubo de ensayo que contiene disolvente y se añade una cierta cantidad (Adleman usó 50 pmoles) de oligos $\overline{s_i}$, para cada nodo i $(0 \leq i \leq 6)$, y la misma cantidad de oligos e_{ij} para cada arco $(i,j) \in E$. La solución así obtenida se somete a un proceso de renaturalización, usando la ligasa como enzima para rellenar los huecos que pudieran aparecer entre nucleótidos. De esta manera, se obtiene una serie de doble hebras que codifican caminos del grafo. Más aún, las cantidades iniciales consideradas garantizan, con una probabilidad suficientemente alta, que cada camino del grafo aparecerá codificado en el tubo inicial así elaborado.

Para implementar el paso 2 (`rechazar todos los caminos que no empiezan por 0 y terminan en 6`) se aplica la técnica de la reacción en cadena de la polimerasa que se ha descrito anteriormente, utilizando como bordes los oligos s_0 y $\overline{s_6}$, respectivamente. De esta forma, sólo serán amplificados los caminos del grafo que comienzan por el vértice 0 y terminan por el vértice 6.

A continuación, se utiliza la técnica de la electroforesis en gel para seleccionar las moléculas de longitud 140 bp, para implementar el paso 3 (`rechazar todos los caminos que no contienen exactamente siete vértices`).

El paso 4 (`para cada nodo del grafo, rechazar todos los caminos que no pasan por él`) se implementa de la siguiente forma: para cada i $(1 \leq i \leq 5)$ se extraen las moléculas que contienen al oligo s_i (obsérvese que, por el paso 2, no es necesario considerar aquí los casos $i = 0$ e $i = 6$).

Finalmente, para detectar si hay algún camino en el tubo de ensayo resultante, se amplifica el producto obtenido mediante la técnica PCR y se hacen circular las moléculas obtenidas a través de un gel.

Acerca del experimento de Adleman

Una vez descrito el experimento de Adleman, conviene tener presente las siguientes consideraciones:

- Las operaciones moleculares que tienen lugar en los tubos son aplicadas simultáneamente sobre todas las moléculas presentes en el mismo.
- En el tubo de ensayo inicial existe un número de cadenas que es *exponencial* en el tamaño del dato de entrada.
- El número de operaciones moleculares realizadas es *lineal* en el tamaño del dato de entrada.
- Como ya se indicó anteriormente, la elección de oligos de longitud 20 mer para codificar nodos y arcos (con la excepción de los arcos de extremos 0 y 6), así como las cantidades de cada oligo que se añaden a la solución, se realiza para garantizar que en el tubo de ensayo inicial estén todas las cadenas que codifican todos los caminos del grafo (y sólo esas). En el experimento de Adleman la cantidad considerada fue excesiva y aparecían codificados todos los caminos del grafo, pero en cantidades elevadas. Desde luego, la longitud seleccionada para los oligos no tiene porqué ser igual para todos ellos y, en general, será una variable que dependerá de la densidad y del tamaño del grafo. Sobre el problema de la codificación y elección adecuada de las longitudes de los oligos iniciales hay muchos estudios realizados, pero ninguno de ellos puede considerarse definitivo. Se trata de un problema complejo, pues una elección adecuada debe evitar la aparición de enlaces no deseables entre trozos de cadenas que no son complementarias, o entre distintas porciones de la misma cadena y, simultáneamente, no deben ser excesivamente grandes a fin de que los enlaces entre cadenas complementarias sean posibles.
- A lo largo de la ejecución del experimento aparecen errores debido a que las operaciones moleculares que se realizan en el laboratorio no son *perfectas*: en algún paso podemos seleccionar cadenas no deseadas, o bien rechazar alguna cadena que codifique una solución correcta del problema. No obstante, estos errores pueden ser controlados y amortiguados, en cierta medida, a fin de conseguir los efectos deseados con una "alta probabilidad" (ver [2] y [69] para un tratamiento más detallado de esta cuestión).
- D. Boneh, C. Dunworth y R. Lipton [8] conjeturan que el número máximo de moléculas de ADN que se pueden procesar en un experimento molecular

es del orden de 10^{21} (aproximadamente 2^{70}). De acuerdo con esta conjetura, el experimento de Adleman sólo podría ser simulado para grafos de tamaño menor o igual que 70 (como simple curiosidad conviene hacer notar que para simular el experimento de Adleman para un grafo de tamaño 200 y densidad media, se necesitaría un volumen molecular equivalente al peso de la tierra).

- Pese a lo anterior, podríamos resaltar algunas de las ventajas que, potencialmente, proporciona el experimento de Adleman en relación con el mejor supercomputador del momento:

  ⋆ *Velocidad de cálculo*: La máxima velocidad de cálculo alcanzada en el experimento fue de $1'2 \times 10^{18}$ operaciones por segundo (en la cincuentaava parte de una cucharadita de solución, se formaron más de 100 billones de caminos en un segundo), mientras que el mejor supercomputador de la actualidad alcanza un máximo de 10^{12} operaciones por segundo.

  ⋆ *Consumo de energía*: En el experimento, con 1 julio de trabajo se consiguió realizar hasta 2×10^{19} operaciones (de acuerdo con la segunda ley de la termodinámica, el número máximo de operaciones *irreversibles* por julio es de 34×10^{19}), mientras que el mejor supercomputador de la actualidad es capaz de realizar como máximo 10^9 operaciones por julio.

  ⋆ *Densidad de almacenamiento de información*: En el experimento se consiguió una densidad de almacenamiento de información de 1 bit por nanómetro cúbico, mientras que el mejor supercomputador de la actualidad tiene una densidad máxima de información del orden de 1 bit por 10^{12} nm^3 (un gramo de ADN ocupa en seco 1 centímetro cúbico, aproximadamente, y es capaz de almacenar la información equivalente a más de 1 billón de CD's convencionales).

Formulación abstracta

Pese a que el experimento de Adleman supone el primer acercamiento a la computación molecular, dista mucho de proporcionar un modelo de computación, puesto que no establece una base abstracta sobre la que poder diseñar, en un lenguaje propio, algoritmos que resuelvan problemas de forma general. Como primer paso hacia la creación de un modelo de computación que sea capaz de aprovechar las características ventajosas que se ponen de manifiesto en este experimento, y que se han detallado en el apartado anterior, podemos considerar una estructura de datos cuyos objetos son los *tubos* de ensayo, entendiendo como tal un multiconjunto finito de

cadenas del alfabeto $\Sigma_{ADN} = \{A, C, G, T\}$, y sobre ellos, las siguientes operaciones moleculares:

1. `Extraer` (T, γ): Dado un tubo, T, y una cadena, γ, sobre Σ_{ADN}, esta operación devuelve los tubos $+(T, \gamma)$ y $-(T, \gamma)$ cuyos elementos son, respectivamente, las moléculas de T que contienen (respectivamente, **no** contienen) a γ como subcadena.

2. `Inicial` (T, γ): Dado un tubo, T, y una cadena, γ, devuelve un tubo que contiene las moléculas de T que "comienzan" por la cadena γ.

3. `Final` (T, γ): Dado un tubo, T, y una cadena, γ, devuelve un tubo que contiene las moléculas de T que "terminan" con la cadena γ.

4. `Long` (T, n): Dado un tubo, T, y un número natural, $n \geq 1$, devuelve un tubo cuyos elementos son las moléculas de T que tienen longitud n.

5. `Detectar` (T): Dado un tubo, T, devuelve **SI** si existe alguna cadena de ADN en T, y devuelve **NO** en caso contrario.

Con la estructura de datos considerada y las operaciones moleculares descritas, el experimento de Adleman se puede expresar genéricamente como sigue:

```
Entrada: T (tubo de ensayo inicial)
    T ← Inicial(T, s_0)
    T ← Final(T, s_6)
    T ← Long(T, 140)
    para i = 1 hasta 5 hacer
        T ← +(T, s_i)
    detectar (T)
```

Conviene resaltar el hecho de que el experimento de Adleman no puede considerarse como un esquema algorítmico molecular, en el siguiente sentido: el tubo de ensayo inicial del experimento sólo se puede utilizar para el grafo concreto para el que fue diseñado; si se considera otro grafo, aunque tenga tamaño siete, habrá que elaborar nuevamente el tubo de ensayo inicial. No obstante, la formulación abstracta antes descrita nos permite diseñar un primer esbozo de programa molecular para resolver el problema del camino hamiltoniano para un grafo dirigido, G, de tamaño $n + 1$ con dos nodos distinguidos v_i y v_f, siguiendo las ideas de Adleman (es decir, codificando nodos y arcos por oligos de longitud l). Si s_0 es el oligo que codifica el nodo v_i y s_n es el oligo que codifica a v_f, entonces se obtiene el siguiente "programa molecular":

```
Entrada: (T, s_0, s_n)
        T ← Inicial(T, s_0)
        T ← Final(T, s_n)
        T ← Long(T, (n+1)·l)
        para i = 1 hasta n − 1 hacer
            T ← +(T, s_i)
        detectar (T)
```

En el capítulo siguiente veremos algunos modelos de computación molecular, en los que se explicitan las estructuras abstractas de datos sobre las que se va a trabajar, y las operaciones moleculares que actúan sobre ellas.

2.2.2. El experimento de Lipton

En abril de 1995, R.J. Lipton [42], siguiendo las ideas del experimento de Adleman, resuelve una instancia concreta del *problema* **SAT** *de la satisfactibilidad de la lógica proposicional.* Sin embargo, pese a la similitud del procedimiento usado, introduce la peculiaridad de que el tubo de ensayo inicial no depende del dato de entrada concreto, sino únicamente de su "tamaño" (el número de variables de la fórmula). De esta manera se consigue una solución molecular de cualquier instancia del problema **SAT** de tamaño prefijado, proporcionando el que se puede considerar realmente como primer esquema algorítmico molecular.

Antes de describir propiamente el experimento de Lipton, vamos a introducir algunos conceptos necesarios para enunciar el problema de la satisfactibilidad.

El problema SAT

El lenguaje de la lógica proposicional consta de:

(a) Un conjunto infinito numerable, **VP**, que denominaremos de variables proposicionales.

(b) Dos conectivas lógicas: $\neg$ (negación) y $\vee$ (disyunción).

(c) Dos símbolos auxiliares: "(" y ")".

A partir de la negación y de la disyunción, se definen de manera natural las conectivas lógicas $\wedge$ (conjunción), $\rightarrow$ (implicación) y $\leftrightarrow$ (doble implicación).

El conjunto **PForm** de las fórmulas proposicionales se define recursivamente (mediante un procedimiento generativo) como sigue: es el menor conjunto Γ que verifica, simultáneamente, las condiciones siguientes:

(a) $\mathbf{VP} \subseteq \Gamma$.

(b) Es cerrado bajo negación (es decir, si $\varphi \in \Gamma$ entonces $\neg\varphi \in \Gamma$).

(c) Es cerrado bajo disyunción (es decir, si $\varphi, \psi \in \Gamma$ entonces $\varphi \vee \psi \in \Gamma$).

Un *literal* es una variable proposicional o la negación de una variable proposicional. Una *cláusula* es una disyunción de literales. Se dice que una fórmula proposicional está en *forma normal conjuntiva* si es una conjunción de cláusulas; es decir, si es una conjunción de disyunción de literales.

Una *valoración* o *asignación de verdad* es una aplicación de **VP** en $\{0,1\}$. Toda valoración se extiende unívocamente de **VP** al conjunto **PForm** de manera natural (de acuerdo con las clásicas *tablas de verdad* de las conectivas lógicas).

Diremos que una valoración, σ, es *relevante* para una fórmula proposicional, φ, si y sólo si σ es una función booleana cuyo dominio contiene a $\mathrm{Var}(\varphi)$, siendo $\mathrm{Var}(\varphi)$ el conjunto de las variables proposicionales de φ. Obsérvese que el número de valoraciones relevantes, σ, para una fórmula proposicional φ tal que $\sigma(x) = 0$ para cada $x \in \mathbf{VP} - \mathrm{Var}(\varphi)$, es $2^{|\mathrm{Var}(\varphi)|}$. Así, a la hora de trabajar con las variables que afectan a una fórmula, sólo nos preocuparemos de los valores que se le asignan a cada una de sus variables.

Una fórmula proposicional se dice que es *satisfactible* si existe, al menos, una valoración que asigna 1 a dicha fórmula.

Problema de la Satisfactibilidad de la Lógica Proposicional: *Dada una fórmula proposicional en forma normal conjuntiva, determinar si es satisfactible.*

Como es bien sabido, el problema de la satisfactibilidad de la lógica proposicional es **NP**–completo [26].

Preparación del experimento

Sea $\varphi \equiv c_1 \wedge \ldots \wedge c_p$ con $c_i = l_{i,1} \vee \ldots \vee l_{i,r_i}$, una fórmula proposicional en forma normal conjuntiva, cuyo conjunto de variables es $\mathrm{Var}(\varphi) = \{x_1, ..., x_n\}$. Asociamos a la fórmula φ un grafo dirigido, $G_n = (V_n, E_n)$ (donde V_n es el conjunto de vértices y E_n es el conjunto de aristas), definido como sigue:

$$\begin{cases} V_n = \{a_i, x_i^j, a_{n+1} : \ 1 \leq i \leq n, \ 0 \leq j \leq 1\} \\ \\ E_n = \{(a_i, x_i^j), (x_i^j, a_{i+1}) : \ 1 \leq i \leq n, \ 0 \leq j \leq 1\} \end{cases}$$

El grafo G_n, descrito en la figura 2.6, verifica las siguientes propiedades:

- Existen 2^n caminos simples desde a_1 hasta a_{n+1} en G_n.
- Existe una biyección natural entre el conjunto de los caminos anteriormente citados y las valoraciones relevantes para φ, de acuerdo con el siguiente criterio: si $\gamma = a_1 x_1^{j_1} a_2 x_2^{j_2} ... x_n^{j_n} a_{n+1}$ es un camino desde a_1 hasta a_{n+1}, entonces la valoración asociada, σ, relevante para φ, viene caracterizada por $\sigma(x_i) = j_i$ $(1 \leq i \leq n)$.

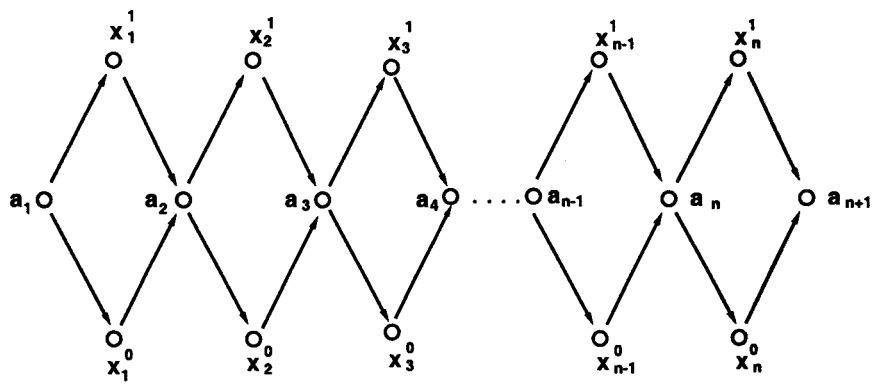


Figura 2.6. Grafo dirigido asociado a una fórmula proposicional con n variables

Para elaborar el tubo de ensayo inicial, T_0, se procede de manera similar a la seguida en el experimento de Adleman. A cada nodo, $i \in V_n$, del grafo se le asocia un oligo, s_i, de longitud fija (esta longitud debe depender de n). Para cada i se designa por s_i' (respectivamente, s_i'') la primera mitad (respectivamente, la última) de la cadena s_i. A cada arco, $(i,j) \in E_n$, del grafo se le asocia el oligo $e_{ij} = 3' - \overline{s}_i'' \, \overline{s}_j'$.

Se parte de un tubo de ensayo que contiene disolvente. Se le añade a la solución una cierta cantidad de oligos s_i, para cada nodo $i \in V_n$, la misma cantidad de oligos e_{ij}, para cada arco $(i,j) \in E_n$, y la misma de oligos $3' - \overline{s}_{a_1}'$ y $3' - \overline{s}_{a_{n+1}}''$. La solución se somete a un proceso de renaturalización. Finalmente se seleccionan las cadenas simples con bordes $5' - s_{a_1}$, para lo cual basta someter la solución a un proceso de desnaturalización y aplicar la operación `extraer` (T_0, s_{a_1}), que se ha descrito en el experimento de Adleman.

De esta manera, se obtienen una serie de doble hebras que codifican caminos del grafo que van desde a_1 hasta a_{n+1}. Las cantidades iniciales de oligos que se consideren deben garantizar que cada camino desde a_1 hasta a_{n+1} (y sólo esos) estén codificados en el tubo inicial. Además, debido a la simetría del grafo auxiliar considerado, todos los caminos aparecen repetidos el mismo número de veces, aproximadamente. Por tanto, en el tubo de ensayo inicial se obtiene un multiconjunto de moléculas que codifican todas las valoraciones relevantes para la fórmula φ.

Diseño del programa molecular

Para cada literal, $l_{i,j}$, que aparece en la fórmula φ, notaremos

$$l_{ij}^1 = \begin{cases} x_m^1 & \text{, si } l_{i,j} = x_m \\ x_m^0 & \text{, si } l_{i,j} = \neg x_m \end{cases}$$

Es decir, si una molécula contiene el oligo $l_{i,j}^1$ como subcadena, entonces el literal $l_{i,j}$ tiene asignado el valor 1 por la valoración que codifica dicha molécula.

La idea del experimento de Lipton es la siguiente: a partir del tubo inicial, T_0, que codifica todas las valoraciones relevantes para la fórmula de entrada, se procede como sigue:

- Se elabora un nuevo tubo, T_1, seleccionando de T_0 todas las valoraciones que hacen verdadera la cláusula c_1. Para ello:
  - ⋆ Se eligen las que hacen verdadero el literal $l_{1,1}$.
  - ⋆ De las que hacen falso $l_{1,1}$, se eligen las que hacen verdadero el literal $l_{1,2}$.
  - ⋆ De las que hacen falso $l_{1,1} \vee l_{1,2}$, se eligen las que hacen verdadero el literal $l_{1,3}$.
  - ⋆ Y así sucesivamente con todos los literales de c_1.
- Se elabora un nuevo tubo, T_2, seleccionando de T_1 todas las valoraciones que hacen verdadera la cláusula c_2 (así dichas valoraciones hacen verdadera la fórmula $c_1 \wedge c_2$). Para ello:
  - ⋆ Se eligen las que hacen verdadero el literal $l_{2,1}$.
  - ⋆ De las que hacen falso $l_{2,1}$, se eligen las que hacen verdadero el literal $l_{2,2}$.
  - ⋆ De las que hacen falso $l_{2,1} \vee l_{2,2}$, se eligen las que hacen verdadero el literal $l_{2,3}$.
  - ⋆ Y así sucesivamente con todos los literales de c_2.
- El proceso se reitera p veces hasta obtener el tubo de salida T_p, a partir de T_{p-1}, que contiene todas las valoraciones que hacen verdadera la fórmula $c_1 \wedge \ldots \wedge c_p$.

Estas ideas sugieren el diseño del siguiente programa molecular:

```
Entrada: T_0
    para i = 1 hasta p hacer
        T_1 ← T_0;   T_0 ← ∅
        para j = 1 hasta r_i hacer
            T' ← +(T_1, l^1_{i,j})
            T_1 ← −(T_1, l^1_{i,j})
            T_0 ← T_0 ∪ T'
    detectar(T_0)
```

En donde $T_0 \cup T'$ indica la **unión** de los tubos T_0 y T' como multiconjuntos, y las operaciones **extraer** y **detectar** funcionan tal como se han descrito en la formulación abstracta del experimento de Adleman.

El número de operaciones moleculares que se ejecutan en el programa es lineal en el número de literales, k, de la fórmula de entrada (en concreto, k operaciones de **extracción**, k operaciones de **unión** y 1 operación de **detección**). Además, el número total de tubos usados es

$$1 + \sum_{i=1}^{p}(3 + 3 \cdot r_i) = 1 + 3p + 3k \in O(k)$$

Extensión del experimento de Lipton

En el experimento de Lipton se proporciona una solución molecular del problema de la satisfactibilidad de fórmulas proposicionales en forma normal conjuntiva. Veamos seguidamente cómo es posible generalizar dicho experimento a fin de decidir la satisfactibilidad de una fórmula proposicional arbitraria.

Proposición 2.1 *Sea φ una fórmula proposicional con k conectivas lógicas. Sea T un tubo que contiene moléculas que codifican valoraciones relevantes para φ. Entonces, con $O(k)$ operaciones de extracción, unión y amplificación, se pueden generar los tubos siguientes:*

$$T^1(T,\varphi) = \{\sigma \in T : \ \sigma(\varphi) = 1\} \ \ y \ \ T^0(T,\varphi) = \{\sigma \in T : \ \sigma(\varphi) = 0\}$$

Demostración:
Por inducción fuerte sobre el número de conectivas lógicas, k, de la fórmula.

- Si $\varphi \equiv x_i$ y T es un tubo que contiene codificaciones relevantes para φ, entonces $T^1(T,\varphi) = +(T, x_i^1)$ y $T^0(T,\varphi) = -(T, x_i^1)$.

  Si $\varphi \equiv \neg x_i$ y T es un tubo que contiene codificaciones relevantes para φ, entonces $T^1(T,\varphi) = +(T, x_i^0)$ y $T^0(T,\varphi) = -(T, x_i^0)$.

- Supongamos cierto el resultado para cada $r \leq k$ y sea φ una fórmula con $k+1$ conectivas lógicas. Sea T un tubo que codifica valoraciones relevantes para φ.

  Si $\varphi \equiv \neg\varphi_1$, entonces por hipótesis de inducción con $O(k)$ operaciones de extracción, unión y amplificación, se pueden obtener los tubos

$$\begin{cases} T^1(T,\varphi_1) = \{\sigma \in T:\ \sigma(\varphi_1) = 1\} \\ T^0(T,\varphi_1) = \{\sigma \in T:\ \sigma(\varphi_1) = 0\} \end{cases}$$

  Entonces basta devolver los tubos

$$\begin{cases} T^1(T,\varphi) = T^0(T,\varphi_1) \\ T^0(T,\varphi) = T^1(T,\varphi_1) \end{cases}$$

  Sea $\varphi \equiv \varphi_1 \vee \varphi_2$. Sea s_i el número de conectivas lógicas de φ_i. Por hipótesis de inducción, con $O(s_1)$ operaciones de extracción, unión y amplificación se pueden obtener los tubos

$$\begin{cases} T^1(T,\varphi_1) = \{\sigma \in T:\ \sigma(\varphi_1) = 1\} \\ T^0(T,\varphi_1) = \{\sigma \in T:\ \sigma(\varphi_1) = 0\} \end{cases}$$

  Amplificamos el tubo $T^0(T,\varphi_1)$ obteniendo dos copias. Entonces, por hipótesis de inducción, con $O(s_2)$ operaciones de extracción, unión y amplificación obtenemos los tubos $T^1(T^0(T,\varphi_1),\varphi_2)$ y $T^0(T^0(T,\varphi_1),\varphi_2)$. En tal situación, basta devolver los tubos

$$\begin{cases} T^1(T,\varphi_1 \vee \varphi_2) = T^1(T,\varphi_1) \cup T^1(T^0(T,\varphi_1),\varphi_2) \\ T^0(T,\varphi_1 \vee \varphi_2) = T^0(T^0(T,\varphi_1),\varphi_2) \end{cases}$$

□

Corolario 2.2 *El problema de la satisfactibilidad para fórmulas proposicionales arbitrarias se puede resolver realizando una operación de detectar y $O(k)$ operaciones de extracción, unión y amplificación (en donde k es el número de conectivas lógicas de la fórmula).*

Demostración:

Dada una fórmula proposicional, φ, con n variables y k conectivas lógicas se procede, al igual que en la preparación del experimento de Lipton, diseñando el grafo dirigido asociado, G_n, y elaborando, a partir de éste, el tubo de ensayo inicial, T_0, que contiene cadenas de ADN que codifican todas las valoraciones relevantes para la fórmula de

entrada. De la proposición anterior se deduce que realizando $O(k)$ operaciones de extracción, unión y amplificación se obtienen los tubos

$$\begin{cases} T^1(T_0, \varphi) = \{\sigma \in T_0 : \ \sigma(\varphi) = 1\} \\ T^0(T_0, \varphi) = \{\sigma \in T_0 : \ \sigma(\varphi) = 0\} \end{cases}$$

Entonces basta aplicar la operación `detectar` $(T^1(T_0, \varphi))$.

□

Capítulo 3

Modelos de computación molecular

Como ya hemos comentado en el capítulo anterior, el experimento de Lipton resuelve una instancia concreta del problema de la satisfactibilidad de la lógica proposicional, a la vez que proporciona un primer esquema algorítmico dentro del campo de la computación molecular, ya que, prefijado el tubo de ensayo inicial a través del grafo dirigido G_n, se puede realizar el experimento para cada fórmula proposicional dada en forma normal conjuntiva que tenga a lo sumo n variables. En este capítulo presentaremos un marco formal en el que dichos experimentos pueden ser considerados como la ejecución de un procedimiento mecánico; es decir, presentaremos los primeros modelos abstractos de computación molecular que permiten la escritura de un *programa*, en un determinado lenguaje, cuya ejecución resuelva problemas, independientemente de la instancia concreta que reciban como dato de entrada.

Hemos de recordar que un modelo de computación, M, se caracteriza, básicamente, por los siguientes elementos:

1. Una estructura de *datos*, $\mathcal{D}_M$.
2. Un conjunto de *programas*, $\mathcal{P}_M$.
3. Una *función semántica*, $\mathcal{S}_M$, que establece cómo se ejecutan los programas del modelo.

Habitualmente, el principal *tipo de dato* que manejan los modelos de computación molecular es el objeto denominado *tubo*, que se puede identificar con el conjunto (aunque como veremos, deberíamos considerar *multiconjuntos*) de moléculas sobre el que se realizan las operaciones moleculares que caractericen al modelo. Para dar una representación abstracta de estas moléculas, será habitual considerarlas como cadenas sobre un alfabeto prefijado.

Las *instrucciones básicas* de un modelo de computación molecular pueden ser de dos tipos:

- **Moleculares**: operaciones basadas en propiedades de las moléculas que sirvan de sustrato.
- **Robóticas**: operaciones secuenciales estándar (bucles, condicionales, asignaciones, etc.) que, básicamente, proporcionan la secuenciación de las operaciones moleculares. Estas operaciones no actúan directamente sobre la estructura de las moléculas que sirvan de sustrato.

La *función semántica* del modelo se define de manera natural, siendo para las instrucciones moleculares, el resultado correspondiente de la operación molecular que representa; y para las instrucciones robóticas, la semántica secuencial convencional.

El concepto de *programa* en un modelo de computación molecular se introduce de manera natural como una sucesión finita de instrucciones básicas del modelo.

Se considera como condición indispensable para la elección de las operaciones moleculares primitivas de los modelos que todas ellas deben ser implementables en el laboratorio usando técnicas actuales de biología molecular (aun que se permite un cierto margen de error en las mismas).

En un modelo de computación molecular, conviene distinguir entre lo que es el aparato formal del mismo y lo que representa la implementación práctica que se puede realizar en el laboratorio. Denominaremos *sustrato computacional* de un modelo a la sustancia molecular sobre la que se materializa la implementación del mismo.

El modelo *no restringido* de L. Adleman y el modelo *débil* de M. Amos, que se detallan en las próximas secciones, son modelos de computación molecular basados en procedimientos de *filtrado*. Es decir, cualquier computación en dichos modelos comienza con un tubo de ensayo inicial que contiene todas las posibles soluciones del problema y, mediante una serie de procedimientos de selección (a modo de sucesivos filtrados), devuelve un tubo de salida que contiene todas las soluciones correctas del mismo (y sólo esas). Así pues, las computaciones en dichos modelos no alteran la estructura interna de las moléculas de ADN que componen los tubos, simplemente las moléculas son rechazadas o no, de acuerdo con un determinado test. Por tanto, podemos interpretar que estos modelos de computación carecen de memoria de acceso aleatorio, y suelen agruparse, por tanto, en los denominados *modelos moleculares sin memoria.*

A la hora de trabajar en estos modelos hay que distinguir claramente una *fase de inicialización*, en la que se elabora el tubo de ensayo inicial (se ha de garantizar que

en ese tubo estén todas las moléculas que codifican posibles soluciones del problema), y una *fase de ejecución*, en la que se van filtrando del tubo inicial las moléculas que codifican soluciones correctas del problema (y sólo esas).

El modelo *sticker*, sin embargo, opera de forma sustancialmente distinta, ya que las soluciones a los problemas que se dan en este modelo pasan por una etapa de construcción. Este modelo posee operaciones moleculares que alteran la estructura interna de las moléculas de ADN que componen los tubos, lo que permite usar dichas moléculas a modo de unidades de memoria en las que la información puede ser almacenada y modificada en tiempo de ejecución del programa. De ahí que este modelo molecular se agrupe con aquellos denominados *modelos moleculares con memoria de acceso aleatorio.*

Recordemos brevemente que un alfabeto V es un conjunto no vacío. Una cadena sobre un alfabeto V es una sucesión finita de elementos de V. Se representa por V^* el conjunto de todas las cadenas sobre V, y por V^+ el conjunto de todas las cadenas sobre V distintas de la cadena vacía.

3.1. Modelo no restringido de Adleman

Este modelo de computación molecular fue el primero que se introdujo [2]. La estructura básica de información del modelo es el *tubo*, que es la representación abstracta del tubo de ensayo que se utiliza en el laboratorio, y que contendrá las moléculas de ADN sobre las que se efectuarán la sucesión de operaciones que indique el programa correspondiente del modelo.

Definición 3.1 *Un* tubo *en el modelo no restringido es un multiconjunto finito de cadenas del alfabeto* $\Sigma_{ADN} = \{A, C, G, T\}$.

Las cadenas del alfabeto Σ_{ADN} pueden ser implementadas de manera natural en el laboratorio a través de moléculas de ADN, gracias al direccionamiento de las mismas. En los modelos moleculares que tienen al ADN como sustrato computacional, el alfabeto habitual es Σ_{ADN}. No obstante, con frecuencia y por cuestiones técnicas, se usa como alfabeto de estos modelos otro distinto, de tal manera que cada símbolo del nuevo alfabeto está codificado por un determinado oligo en Σ^*_{ADN}.

Las operaciones básicas que se pueden realizar sobre los tubos, es decir, las instrucciones moleculares básicas del modelo no restringido, son las siguientes:

- `Extraer`(T, γ): dado un tubo, T, y una cadena, γ, de Σ_{ADN}, devuelve dos tubos:

$$\begin{cases} +(T,\gamma) = \{\{\sigma \in T : \ \gamma \text{ es subcadena de } \sigma\}\} \\ -(T,\gamma) = \{\{\sigma \in T : \ \gamma \text{ no es subcadena de } \sigma\}\} \end{cases}$$

- `Mezclar`(T_1, T_2): dados dos tubos, T_1 y T_2, devuelve un nuevo tubo, $T_1 \cup T_2$, que es la unión de ambos, como multiconjuntos.

- `Amplificar` $(T, \{T_1, T_2\})$: dado un tubo T, devuelve dos tubos, T_1 y T_2, que son copias exactas de T.

- `Detectar`(T): dado un tubo, T, devuelve **SI**, en el caso en que T contenga alguna molécula de ADN, y **NO** en caso contrario.

Hay que notar que las operaciones que aquí (y en el resto de los modelos) se describen actúan sobre la totalidad del contenido del tubo; es decir, sobre todas las moléculas del tubo simultáneamente. Sin embargo, en el modelo no restringido de Adleman únicamente la operación molecular **extraer** implementa el paralelismo masivo (y realmente proporciona dos operaciones $+(T, \gamma)$ y $-(T, \gamma)$). Además, debe observarse que, en efecto, las operaciones descritas no implican cambios en la estructura interna de las moléculas que integran los tubos, sino que, a lo sumo, se limitan a *seleccionar* de entre estos elementos aquellos que satisfacen ciertas propiedades. Es por ello que anteriormente nos referimos a este modelo como un *modelo de filtrado.*

Este modelo intenta reflejar, de una forma más o menos fiel, las operaciones básicas que se pueden realizar en el laboratorio, por lo que su implementación física no plantea problemas. Sin embargo, hay que resaltar la importancia que tiene la elaboración del tubo de entrada que recibe un programa en este modelo. En dicha construcción hay que generar todas las posibles soluciones del problema que se pretende resolver.

Definición 3.2 *Dado un programa, P, en el modelo no restringido y un tubo de ensayo inicial, T_0, la computación de P a partir de T_0 es una sucesión finita de tubos $T_0 \vdash_P T_1 \vdash_P \cdots \vdash_P T_r$ en donde T_{i+1} es el tubo resultante de T_i al aplicar una operación básica del modelo.*

En la última sección de este capítulo se justificará que este modelo de computación tiene la misma potencia computacional que las máquinas de Turing.

3.2. Modelo débil de Amos

Definición 3.3 *Un tubo en el modelo débil es un multiconjunto finito de cadenas del alfabeto* $\Sigma_{ADN} = \{A, C, G, T\}$.

Las instrucciones moleculares primitivas del modelo débil son las siguientes:

- `Quitar`$(T, \{\gamma_1, ..., \gamma_k\})$: Dado un tubo, T, y un número finito de cadenas, $\gamma_1, ..., \gamma_k$, de Σ, devuelve el tubo obtenido de T eliminando todas aquellas cadenas que contengan, al menos, una ocurrencia de alguna de las cadenas $\gamma_1, ..., \gamma_k$ (téngase presente que $T = \text{quitar}(T, \emptyset)$).

- `Copiar`$(T, \{T_1, ..., T_k\})$: Dado un tubo, T, y un número natural, $k \geq 2$, devuelve k tubos, $T_1, ..., T_k$, que son copias exactas de T.

- `Unión`$(\{T_1, ..., T_k\}, T)$: Dados los tubos $T_1, ..., T_k$, con $k \geq 2$, devuelve un tubo, T, cuyo contenido es la unión de los tubos $T_1, ..., T_k$, como multiconjuntos.

- `Selección`(T): Dado un tubo, T, selecciona aleatoriamente un elemento de T en el caso en que $T \neq \emptyset$; en caso contrario, devuelve **NO**.

Obsérvese que en el modelo débil únicamente la operación molecular `quitar` implementa el paralelismo masivo y que, además, las instrucciones primitivas no alteran la estructura interna de las moléculas sobre las que actúan.

Definición 3.4 *Dado un programa, P, en el modelo débil y un tubo de ensayo inicial, T_0, la computación de P a partir de T_0 es una sucesión finita de tubos $T_0 \vdash_P T_1 \vdash_P \cdots \vdash_P T_r$ en donde T_{i+1} es el tubo resultante de T_i al aplicar una operación básica del modelo.*

Al igual que con el modelo anterior, los resultados que se presentan en la última sección de este capítulo justificarán también que el modelo débil tiene la misma potencia computacional que las máquinas de Turing.

3.3. El Modelo Sticker

A diferencia de los otros modelos de computación molecular basados en el ADN que se han presentado, el modelo sticker tiene memoria de acceso aleatorio (que no requiere extender las cadenas), no usa enzimas, y, al menos teóricamente, sus materiales son reutilizables. Este modelo fue presentado en 1998 por S. Roweis, E. Winfree y otros [77].

Gran parte del interés creciente que está adquiriendo la computación molecular se debe a la esperanza de conseguir algún día una plataforma computacional masivamente paralela, capaz de atacar problemas cuya solución (para ejemplares de tamaño relativamente grande) se ha resistido, hasta ahora, a las arquitecturas convencionales. Se han propuesto muchas arquitecturas (L. Adleman, R.J. Lipton, M. Amos, D. Boneh, D. Beaver, P. Rothemund, etc.) que sugieren que los computadores basados en ADN pueden ser lo suficientemente flexibles como para atacar un amplio espectro de problemas difíciles, aunque algunas cuestiones básicas como la escala volumétrica de los materiales y la fiabilidad de las operaciones de laboratorio permanezcan aún sin resolver.

Actualmente se considera que el modelo sticker es uno de los más interesantes que se han dado en la computación molecular basada en ADN, y será con el que vamos a trabajar en la primera parte de esta memoria. Por ello, haremos un estudio más detallado de sus características fundamentales, estudiando cómo se pueden implementar sus operaciones básicas en el laboratorio, e incluso detallaremos una máquina que podría funcionar a modo de computador molecular basado en este modelo, siguiendo las ideas originales de S. Roweis, E. Winfree y otros.

3.3.1. Representación de la información en el modelo sticker

El modelo sticker utiliza, básicamente, dos tipos de moléculas de ADN: las cadenas de memoria y los stickers.

Definición 3.5 *Una* cadena de memoria de tipo (n, k, m), *con* $n \geq km$, *es una cadena simple de ADN de longitud* n *que consta de* k *subcadenas de longitud* m, *que denominaremos regiones.*

Cada una de las regiones de una cadena de memoria debe estar claramente diferenciada de las restantes y se identificará con una variable booleana, proporcionando una unidad de información.

Figura 3.1. Cadena de memoria para $k = 4$, $m = 5$ y $n \geq 20$

Definición 3.6 *Dada una cadena de memoria del tipo* (n, k, m), *un* sticker *asociado a dicha cadena es una cadena simple de longitud* m *que es complementaria a una y sólo una región de la cadena de memoria.*

Figura 3.2. Stickers asociados a la cadena de la figura 2.1

Definición 3.7 *Sea* $(\gamma_1, \ldots, \gamma_k)$ *una cadena de memoria del tipo* (n, k, m) *cuyas regiones con* $\gamma_1, \ldots, \gamma_k$. *Diremos que una región,* γ_i, *está* activada *(o* activa*) si dicha región está complementada por algún sticker asociado. Caso contrario, diremos que la región está* desactivada *(o* inactiva*).*

Definición 3.8 *Un* complejo de memoria del tipo (n, k, m) *es una doble hebra parcial compuesta por una cadena de memoria del tipo* (n, k, m) *que, eventualmente, está complementada por algunos stickers asociados.*

De manera natural, identificaremos un complejo de memoria del tipo (n, k, m) con un número binario de k dígitos $\{0, 1\}$ de acuerdo con el siguiente criterio: el i-ésimo dígito de dicho número es 1 (respectivamente 0) si y sólo si la región i-ésima de la cadena de memoria que determina el complejo está activada (respectivamente, desactivada).

En la figura 3.3 aparecen dos ejemplos de complejos de memoria del tipo $(30, 6, 5)$, con el número binario que determinan.

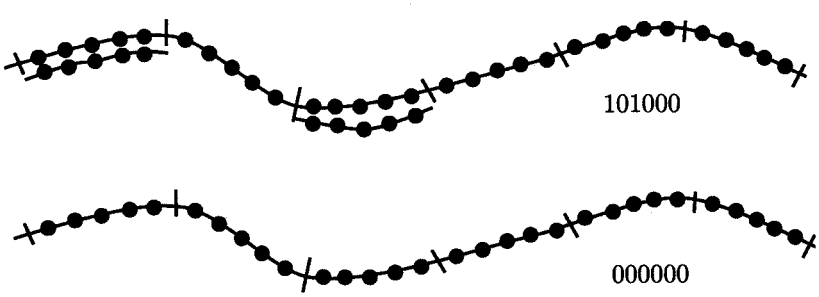


Figura 3.3. Ejemplos de complejos de memoria

Una forma muy útil de seleccionar las regiones del complejo puede ser la siguiente: las regiones impares están determinadas por Purinas (A, G), y las regiones pares están determinadas por Pirimidinas (C, T).

Esta forma de seleccionarlas tiene una gran ventaja: proporciona de manera natural una especie de frontera o barrera entre las distintas regiones, lo cual hace imposible (o muy difícil) que un sticker se adhiera al complejo de forma no deseada, como podría ser la siguiente:

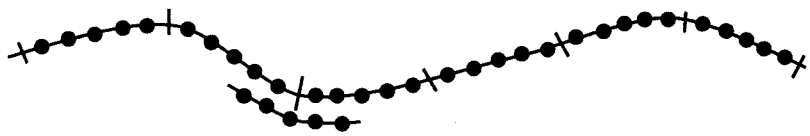


Figura 3.4. Ejemplo de enlace no deseado

Es interesante comparar los mecanismos de representación de la información en el experimento de L. Adleman (o de R.J. Lipton) y en el del modelo sticker.

Ambos están basados en el paradigma de la complementariedad de Watson-Crick y la direccionalidad.

1. En el experimento de L. Adleman se parte de cadenas simples y cortas que se van enlazando paso a paso formando doble hebras que, eventualmente, poseen unos pequeños extremos sin aparear (voladizos). Las doble hebras así construidas carecen de huecos.

2. En el modelo sticker se parte de una cadena simple de memoria (que suele ser larga) y una serie de stickers (que son cadenas simples cortas). Los apareamientos dan lugar a los complejos: doble hebras parciales cuya longitud es la de la cadena de memoria. Los complejos de memoria pueden tener huecos (considerados como doble hebras).

Ahora bien, en ambos casos la densidad de información es similar ya que:

- En el experimento de Adleman, representando cada nodo y cada arco por una cadena de longitud 20, la densidad de información es de 1/20 bits por nucleótido (es decir, 1 bit por cada 20 nucleótidos).

- En un modelo sticker cuyas cadenas de memoria sean del tipo (n, k, m), la densidad de información es de $1/m$ bits por cada nucleótido (es decir, 1 bit por cada m nucleótidos).

Desde luego, la densidad máxima de representación de la información mediante moléculas de ADN será de 2 bits por cada nucleótido. Con esa densidad, un ordenador molecular que use extracción o separación (en alguna de las acepciones) debería tener la capacidad de separar moléculas usando únicamente una base. Eso es imposible de conseguir en la práctica, por ello hay que sacrificar densidad de información para que el modelo sea experimentalmente plausible.

3.3.2. Computaciones en el modelo sticker.

Las operaciones primitivas del modelo sticker operan sobre tubos de ensayo que contienen complejos de memoria, realizando la misma operación sobre todos los complejos presentes en el tubo. Por ello, también en este modelo se consideran a los tubos como la estructura de datos básica.

Definición 3.9 *Un* tubo *en el modelo sticker es un multiconjunto finito de complejos de memoria del mismo tipo.*

A continuación definimos las instrucciones básicas del modelo. Como operaciones moleculares primitivas, se consideran las siguientes:

1. `Mezclar`(T_1, T_2): dados dos tubos, T_1 y T_2, devuelve un nuevo tubo cuyo contenido es la unión de T_1 y T_2 como multiconjuntos.

2. `Extracción`(T, i) (o `separación`(T, i)): dado un tubo, T, de complejos del tipo (n, k, m), y un número natural i $(1 \leq i \leq k)$, devuelve dos nuevos tubos:

$$+(T, i) = \{\{\sigma \in T : \text{ la región } i\text{-ésima de } \sigma \text{ está activa}\}\}$$

$$-(T, i) = \{\{\sigma \in T : \text{ la región } i\text{-ésima de } \sigma \text{ está inactiva}\}\}$$

3. `Activar`(T, i): dado un tubo, T, de complejos del tipo (n, k, m), y un número natural i $(1 \leq i \leq k)$, devuelve un nuevo tubo que contiene todos los complejos de T a los que se les ha activado la región i-ésima (si estaba inactiva).

4. `Desactivar`(T, i): dado un tubo, T, de complejos del tipo (n, k, m), y un número natural i $(1 \leq i \leq k)$, devuelve un nuevo tubo que contiene todos los complejos de T a los que se les ha desactivado la región i-ésima (si estaba activa).

5. `Leer`(T): dado un tubo, T, determina si $T \neq \emptyset$, en cuyo caso aísla un complejo y lee su contenido (es decir, decodifica dicho complejo).

Nota 3.10 Obsérvese que la operación `extracción`(T, i) proporciona realmente dos operaciones distintas: $+(T, i)$ y $-(T, i)$. Además, en este modelo las operaciones `extracción`, `activar` y `desactivar` implementan el paralelismo masivo.

A continuación vamos a introducir los datos de entrada que se suelen usar en las computaciones del modelo sticker.

Definición 3.11 *Sean n, k, l, m tales que $n \geq km$ y $1 \leq l \leq k$. Una (n, k, l, m)-librería es el conjunto de todos los complejos de memoria del tipo (n, k, m) tales que las últimas $k - l$ regiones del complejo están desactivadas.*

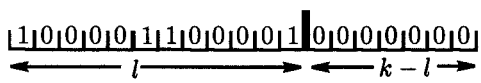


Figura 3.5. Elemento de una (k, l)-librería

En lo que sigue se supondrán fijados n y m, por lo que hablaremos simplemente de (k, l)-librerías. Por tanto, una (k, l)-librería es un conjunto que consta exactamente de 2^l complejos de memoria tales que cada uno de ellos codifica un número binario de k dígitos y tales que los últimos $k - l$ dígitos son iguales a cero.

Con otras palabras, una (k, l)-librería se puede describir, vía la codificación, mediante el siguiente conjunto de palabras sobre $\{0, 1\}$:

$$\{x0^{k-l} : x \in \{0, 1\}^l\}$$

Definición 3.12 *Un* tubo madre del tipo (n, k, l, m) *(con $n \geq km$ y $1 \leq l \leq k$) es una (n, k, l, m)-librería.*

Los datos de entrada de las computaciones en el modelo sticker suelen ser los tubos madre.

Definición 3.13 *Dado un programa, P, en el modelo sticker y un tubo madre, T_0, la* computación *de P a partir de T_0 es una sucesión finita de tubos*

$$T_0 \vdash_P T_1 \vdash_P \cdots \vdash_P T_r$$

en donde T_{i+1} es el tubo resultante de T_i al aplicar una operación básica del modelo.

Nota 3.14 En este concepto de computación se obvian las operaciones robóticas y únicamente se considera la traza de los tubos a lo largo de la ejecución del programa. Cuando se introduzcan operaciones robóticas (como podrían ser los bucles) con el fin de automatizar determinadas operaciones sobre los tubos, se considerará que no aparecen problemas de parada con respecto a las mismas.

Es interesante observar que el modelo sticker es capaz de simular teóricamente, en paralelo, máquinas universales independientes: una por cada complejo de memoria, y bajo la hipótesis de que exista una cantidad no acotada de stickers.

Conviene también observar que el modelo sticker es universal (todo lo computable por una máquina de Turing es computable en el modelo sticker), incluso si eliminamos la operación desactivar (que es la más complicada de implementar experimentalmente en el laboratorio). No obstante, se añade dicha operación en el modelo por cuestiones metodológicas, a fin de expresar de manera más simple determinados algoritmos.

El modelo sticker es un marco adecuado para atacar la resolución de problemas **NP**-completos mediante búsquedas exhaustivas: para cada dato de entrada del problema de longitud l basta considerar un tubo madre del tipo (k, l), en donde $k - l$ es la memoria de trabajo que se necesita para resolver la instancia, y procesar en paralelo todas las posibles 2^l entradas.

3.3.3. Implementación física

A continuación vamos a analizar cómo se puede realizar una implementación física del modelo sticker en el laboratorio.

Para ello, cada una de las operaciones moleculares antes descritas se ha de interpretar en términos de moléculas de ADN. Supondremos que los tubos están formados por complejos de memoria de tipo (n, k, m), con $n \geq km$.

Mezcla en el modelo sticker.

Dados dos tubos, T_1 y T_2, se trata de hallar un nuevo tubo cuyo contenido sea todas las moléculas de T_1 junto con todas las moléculas de T_2.

La implementación física de esta operación se puede realizar como sigue:

1. Hidratar el contenido de los tubos, si éstos no estuvieran aún en solución.

2. Verter uno de los tubos en el otro, para formar un nuevo tubo.

Hay que tener presente que en el vertido hay que contar con la cantidad de ADN que puede permanecer almacenada en las paredes de tubos, pipetas, etc, y que se pierden en la computación. Incluso en el caso de que esta pérdida sea pequeña en relación con la cantidad total de ADN, puede resultar problemática ya que algunas de las moléculas perdidas podrían, presumiblemente, codificar alguna solución relevante del problema.

Extracción en el modelo sticker.

Dado un tubo, T, y una posición i tal que $1 \leq i \leq k$, la operación `extracción` devuelve dos tubos:

$$+(T,i) = \{\{\sigma \in T : \text{la región } i\text{-ésima de } \sigma \text{ está activa}\}\}$$

$$-(T,i) = \{\{\sigma \in T : \text{la región } i\text{-ésima de } \sigma \text{ está inactiva}\}\}$$

Para implementar físicamente esta operación en el modelo sticker se procede como sigue:

1. En el tubo T se introducen gran cantidad de muestras que codifican stickers de la región i-ésima (recordemos que cada región de T debe estar unívocamente determinada, a priori, por una sucesión de nucleótidos que la identifica).

2. Enfriando lentamente la solución contenida en el tubo T conseguimos que:

   - Los complejos cuya región i-ésima esté desactivada, enlacen dicha región con un ejemplar de la muestra.
   - Los complejos cuya región i-ésima esté activada, permanezcan igual.

3. A continuación separamos de la solución los complejos de $+(T,i)$. Para ello, basta que, inicialmente, las muestras tengan adosada una pequeña microesfera magnética. La acción de un campo magnético atraería a las moléculas de $-(T,i)$, con las bolitas magnéticas adosadas (también se podrían haber adherido a un soporte sólido las muestras introducidas para así atraer a las moléculas de $-(T,i)$).

4. Finalmente, los complejos pertenecientes a $-(T,i)$ son liberados de las muestras mediante un proceso de calentamiento de la solución. Esta fase es bastante delicada, ya que al calentar podrían desprenderse del complejo no sólo las muestras recientemente adheridas, sino también los stickers restantes del complejo. Por ello, conviene crear un diferencial entre la temperatura de rotura de enlaces de muestras y las de los stickers. Una forma de conseguir esto podría ser la siguiente: hacer que las muestras no sean exactamente complementarias a sus regiones, o bien que sean complementarias pero más cortas que éstas.

   De esta manera se necesita menos temperatura para romper los enlaces de las muestras que los enlaces de los stickers (otra forma de conseguir esto sería la siguiente: que las muestras fuesen complementarios exactos de las regiones correspondientes, pero que, en cambio, los stickers fuesen de un material más resistente a la temperatura, tipo PNA o DNG, [22]).

Activar y desactivar en el modelo sticker.

Dado un tubo, T, y una posición i tal que $1 \leq i \leq k$, la operación `activar`(T, i) procede así: en cada complejo de T se activa la posición i-ésima, si es que no está activada. La operación `desactivar`(T, i) procede como sigue: en cada complejo de T se desactiva la posición i-ésima si es que está activada.

Veamos cómo implementar físicamente la operación `activar`(T, i).

1. Se introduce en el tubo T una cantidad suficiente de stickers asociados a la región i-ésima de los complejos de T.

2. Enfriando lentamente la solución, aquellos complejos de T que tengan desactivada la región i-ésima, enlazarán esa región con alguno de los stickers introducidos.

3. Separamos los stickers sobrantes (que no se han enlazado a ningún complejo) mediante filtración, o bien separando todos los complejos. Esto se podría conseguir incluyendo inicialmente una región universal común a toda cadena de memoria (bien al principio, o bien al final). Dicha región nunca será recubierta o enlazada a ningún sticker.

   Entonces diseñamos una muestra complementaria de esta región universal que nos permitirá capturar los complejos del tubo, en cualquier momento.

Veamos cómo implementar la operación `desactivar`(T, i):

Necesitamos romper los enlaces de los stickers correspondientes a la región i-ésima. Desde luego, esto no se puede conseguir simplemente calentando la solución, ya que ello haría saltar los enlaces de todos los stickers.

Una posibilidad sería designar ciertas regiones *débiles* de tal manera que los stickers asociados a esas regiones se pudieran desprender del complejo con más facilidad que las restantes.

Así, calentando a una temperatura intermedia sería posible romper los enlaces de los stickers *débiles*, y sólo esos. Claro está, de esta manera se implementaría la operación `desactivar`(T, i), pero únicamente para el caso en que la región i-ésima fuese una *región débil.*

Otra manera de implementar la operación `desactivar`(T, i) está basada en el fenómeno de la invasión de una cadena de **PNA** por formación de una triple hélice [52]. Se ha probado que en condiciones adecuadas,

dos cadenas simples, oligos de todas las pirimidinas PNA, invadirán una doble hebra de ADN formando una triple hélice $(\mathbf{PNA})_2/\mathbf{ADN}$, desplazando las pirimidinas.

Al parecer, este proceso se podría implementar utilizando un tipo de PNA: las grapas PNA [22]. Si, por ejemplo, un sticker consta de 21 nucleótidos se podría usar una grapa de PNA de longitud 14 que formaría una triple hélice con los 7 nucleótidos centrales del sticker.

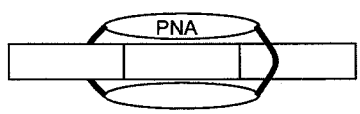


Figura 3.6. Una grapa de PNA alrededor de un complejo

Así, mezclando las grapas específicas de PNA relativas a la región i-ésima correspondiente, las grapas de PNA formarán una triple hélice en el núcleo del sticker, lo que provoca una desestabilización del mismo, con lo cual es necesaria una menor temperatura para romper los enlaces de dicho sticker respecto de su complejo, sin que queden afectados los restantes stickers del mismo.

Sin lugar a dudas, en términos de implementación física, la operación `desactivar` es la más problemática. No obstante, cabe señalar que puede eliminarse esta operación del modelo sin que se pierda, teóricamente, potencia computacional.

Inicialización en el modelo sticker.

Toda computación en el modelo sticker comienza con un proceso de inicialización del tubo de partida o tubo madre, que será una de las librerías definidas anteriormente.

En nuestro paradigma de computación resolveremos problemas **NP**-completos en este modelo mediante la búsqueda exhaustiva sobre entradas de longitud l, lo que nos llevará a procesar en paralelo todas las posibles 2^l entradas, eliminando, si fuera el caso, todas aquellas que no satisfagan algún criterio.

Para construir una (k, l)-librería se procede como sigue:

1. Se sintetizan 2^l copias idénticas de una cadena de memoria con k regiones $(k \geq l)$ que colocamos en un tubo (en solución), T.

2. Se divide el tubo anterior en otros dos, T' y T'', que contengan cada uno la mitad del volumen, aproximadamente.

3. En el tubo T' se introduce una gran cantidad de stickers correspondientes a cada una de las regiones i-ésimas, para $1 \leq i \leq l$.

4. Enfriando adecuadamente la solución, se producirán los correspondientes enlaces, consiguiendo que todas las cadenas de memoria de T' den lugar a complejos que tienen complementadas las regiones $1, 2, \ldots, l$, respectivamente.

5. Los stickers que no han sido usados en la operación anterior se eliminan de T', bien por filtración o bien por separación, a través de la creación inicial de regiones universales en las cadenas de memoria en la forma que se ha indicado anteriormente.

6. Se mezclan los contenidos de los tubos T' y T'' en un nuevo tubo, T'''. Se calienta la solución a fin de conseguir que se rompan los enlaces de las moléculas de T'.

7. Se enfría poco a poco la solución del tubo T''' permitiendo que los stickers de la solución se enlacen aleatoriamente con las cadenas de memoria de T'''.

   Cada región i-ésima, con $1 \leq i \leq l$, tiene 2^{l-1} correspondientes stickers y en T''' hay en total 2^l regiones i-ésimas. Por tanto, cada una de ellas tiene una proporción de 1 sticker para 2 regiones. En consecuencia, la probabilidad de que la región i-ésima de un complejo resultante esté activada es $1/2$.

   Dado un número binario de longitud l y un complejo arbitrario resultante, la probabilidad de que dicho complejo no codifique ese número binario será $1 - \frac{1}{2^l}$. Teniendo presente que existen en total 2^l complejos, resulta que la probabilidad de que un número binario dado de longitud l no esté codificado por ningún complejo en el tubo resultante es

   $$(1 - \frac{1}{2^l})^{2^l}$$

   Para valores suficientemente grandes, dicho número tiende a $\frac{1}{e}$.

Como conclusión, en el tubo resultante cada uno de los complejos deseados serán fabricados con una probabilidad del 63 %, aproximadamente. Desde luego, este porcentaje puede ser incrementado con tal de sintetizar inicialmente más de 2^l cadenas de memoria (a través de la técnica del PCR).

Salida final de una computación en el modelo sticker.

Para determinar un complejo solución del tubo obtenido al final de la computación, se necesita:

1. Detectar la presencia de complejos en dicho tubo.
2. En caso de que existan, aislar uno de esos complejos.
3. Identificar el complejo aislado a través del análisis de los stickers enlazados al mismo.

Veamos cómo implementar cada uno de estos pasos.

- La detección de complejos en el tubo de salida se puede conseguir poniendo un etiquetado fluorescente en las cadenas de memoria (en la fase inicial), y haciendo que la solución pase a través de un tubo capilar (muy fino). Esta técnica puede servir también para aislar un complejo, en el caso en que el tiempo transcurrido entre la detección de un complejo y otro sea *suficientemente grande.*

- El proceso final de identificación de los stickers enlazados al complejo aislado, se puede realizar con la ayuda de un microscopio electrónico, teniendo presente que se conocen a priori el orden de las regiones del complejo (incluso leyendo las regiones desactivadas).

En 1995, Meade [50], implementó esta operación aislando los stickers vertiendo la solución a través de un malla de hibridación.

3.3.4. Diseño de las cadenas de memoria y los stickers

A la hora de diseñar las cadenas de memoria y los stickers, hay que tener presente una propiedad fundamental: cada sticker debe estar especializado en una única región de la cadena, con la que se podrá enlazar. Se trata, pues, de diseñar sucesiones de longitud n tales que existan k sucesiones de longitud m verificando la siguiente propiedad:

Cada una de las regiones tiene, al menos, d_1 *posiciones que se aparean mal con cualquier otra región de la cadena.*

Es decir: d_1 es el número mínimo de malos emparejamientos que se necesitan para que un sticker no se enlace a una cadena.

Hemos de prevenir, también, que la propia cadena se enlace con ella misma. Esto se puede conseguir diseñando una sucesión de longitud n tal que el complementario de cada subsucesión de longitud m tenga, al menos, d_2 malos emparejamientos con cualquier otra subsucesión de longitud m. Es decir: d_2 es el número mínimo de malos emparejamientos que son necesarios para prevenir que una cadena de memoria se enlace consigo misma.

Finalmente, es necesario diseñar muestras que puedan enlazar con unas correspondientes regiones de la cadena, y de tal manera que tengan menor afinidad que los stickers asociados: así se asegura la existencia de temperaturas que permiten la rotura de enlaces de las muestras mientras permanecen los enlaces de los stickers.

Se exige que las muestras tengan, al menos, d_3 malos emparejamientos con la correspondiente región y, al menos, d_4 (con $d_4 > d_3$) malos emparejamientos con cualquier otro sitio de la cadena.

Estos criterios pueden parecer desalentadores, sin embargo, existen formas potencialmente simples de realizar estas tareas.

Para ello, obsérvese que, en general, podemos dejar *vacías* algunas porciones de la cadena de memoria: basta no identificar esas porciones como regiones (por ello, no exigimos que $n = km$, sino que $n \geq km$).

En 1996, K. Mir propone usar sólo Pirimidinas (o respectivamente Purinas) para las cadenas y sólo Purinas (respectivamente Pirimidinas) para los stickers, con el fin de evitar que una cadena de memoria se enlace con ella misma.

A modo de conclusión podemos decir que aunque el diseño de las cadenas y stickers es bastante complicado, resulta que una vez que disponemos de una cadena de memoria con k regiones, esta puede ser reutilizada para cualquier problema que necesite, a lo sumo, k bits de memoria.

3.3.5. Una propuesta de máquina sticker

Siguiendo [77], a continuación describimos una posible máquina que implementa computaciones usando el modelo sticker. La máquina que describimos es una especie de estación de trabajo robótica y paralela para la computación molecular, en la que varios robots y el flujo de líquido es controlado por un ordenador central electrónico programable.

Consta de un estante que contiene muchos tubos de ensayo, una pequeña cantidad de robots, algunas bombas de líquidos, calentadores y enfriadores, así como algunos aparatos microelectrónicos convencionales.

La estación de trabajo almacena todo el ADN (que representa información du-

rante la computación) en los llamados tubos de datos.

Un *tubo de datos* es un cilindro cerrado que tiene unos manguitos en sus bases (lo que permite que fluya líquido a través de ellos).

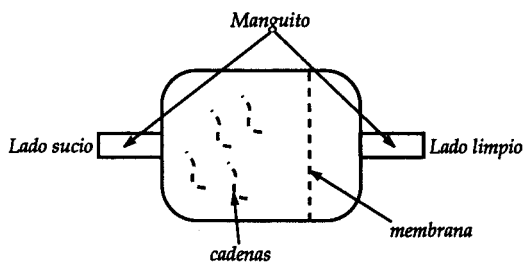


Figura 3.7. Tubo de datos

Próximo a una de las bases hay una membrana que únicamente permite que pase a través de ella el disolvente (y no los stickers o las cadenas de memoria). Esta membrana proporciona una polaridad al tubo de datos: el manguito próximo a ella se dice que es el lado *limpio*, y el opuesto se dice que es el lado *sucio*.

Cuando un tubo de datos no se está usando, se coloca la membrana en la parte inferior. Todo el ADN estará en la parte del cilindro situada por encima de la membrana.

Los tubos de datos (que pueden estar vacíos) contienen un conjunto de complejos de memoria y un conjunto de stickers no enlazados. Cada conjunto de números binarios (de longitud prefijada, l) tiene asociado un tubo de datos que contiene las cadenas de memoria y los stickers que representan los números binarios del conjunto.

Asimismo, cada bit (posición o región) tiene asociado un tubo de datos que contiene una cantidad suplementaria de stickers que codifican dicha posición o región.

Cada vez que se crea un conjunto de complejos (por ejemplo, a partir de la operación de extracción/separación) se coloca en la estantería un nuevo tubo de datos.

Cada vez que se destruye un conjunto de complejos (por ejemplo, a partir de la operación mezcla), el tubo de datos es desechado (o quizás, limpiado, esterilizado y reutilizado posteriormente).

Además de los tubos de datos, existen los *tubos operadores* cuya estructura es similar a la de un tubo de datos, pero con diferente contenido interno (ambos lados/extremos de los tubos operadores se consideran sucios).

Existen tres tipos de tubos operadores:

1. *Tubos operadores en blanco*, que son simplemente tubos vacíos con manguitos en sus bases.

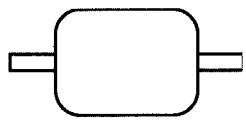


Figura 3.8. Tubo operador en blanco

2. *Tubos operadores sticker*, que son tubos con manguitos en sus bases y una membrana próxima a una de ellas, que permite pasar los stickers pero no las cadenas de memoria.

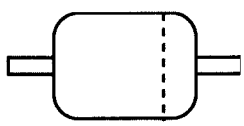


Figura 3.9. Tubo operador sticker

3. *Tubos operadores de extracción*, que son tubos que contienen muchas copias idénticas de oligo-muestras de un determinado bit o región. Esos oligos son adheridos a un soporte sólido o tienen adosados una bolita/sonda que no permite que pasen a través de los filtros por los que, en cambio, sí pueden pasar las cadenas de memoria.

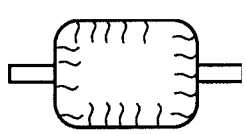


Figura 3.10. Tubo operador de extracción

Para cada bit o región, existe un tubo operador de extracción diferente.

A lo largo de la ejecución de la máquina, algunos tubos serán usados y otros no. Los tubos no usados se almacenarán en una estantería.

Una operación genérica se realiza en la máquina como sigue:

1. Bajo el control de un ordenador electrónico, un robot selecciona y escoge dos tubos de datos de la estantería, así como un tubo operador.

2. Los conectores (o extremos) sucios de los tubos de datos se conectan a los extremos (sucios) del tubo operador. Los conectores limpios de los tubos de datos se conectan entre sí a través de una manguera.

3. La solución circulará a través de los tres tubos.

4. La dirección y duración del flujo, así como la temperatura, es controlada por el ordenador.

5. Cuando el flujo se detiene, uno o más tubos son desconectados y colocados de nuevo en la estantería, por un robot.

6. Nuevos tubos son seleccionados por el robot de la estantería y se reitera el proceso.

Obsérvese que los conectores limpios nunca están en contacto entre sí. Éstos únicamente están en contacto a través de la manguera.

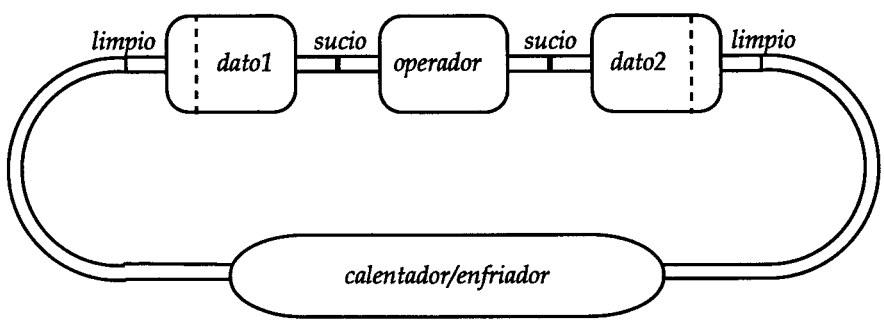


Figura 3.11. Máquina Sticker

A continuación, veamos cómo pueden ser simuladas en la máquina antes descrita cada una de las cuatro operaciones fundamentales del modelo sticker.

Mezcla (T_1, T_2)**.**

Seleccionamos de la estantería dos tubos de datos correspondientes a T_1 y T_2, respectivamente, y un tubo operador en blanco. Hacemos que la solución fría fluya hacia, por ejemplo, el primer tubo de datos.

De esta manera, todos los complejos de memoria irán a parar al tubo de datos T_1, mientras que el tubo de datos T_2 y el tubo operador blanco son desechados.

Extracción(T, i)**.**

Seleccionamos de la estantería el tubo de datos correspondiente a T, así como un tubo de datos vacío. Seleccionamos el tubo operador extracción correspondiente a la región i-ésima. Durante un tiempo hacemos que la solución fría fluya en ambas direcciones (esto permite que las muestras que codifican la región i-ésima se enlacen con aquellos complejos que tienen desactivada dicha región).

A continuación hacemos que el líquido fluya hacia el tubo de datos vacío, forzando que todos los complejos que no se han enlazado con stickers del tipo i-ésimo pasen a dicho tubo.

Desconectamos este tubo (que inicialmente estaba vacío y ahora contiene todos los complejos de T que tenían activadas la región i-ésima), y lo colocamos en la estantería: será el tubo $+(T, i)$. Lo reemplazamos por otro tubo vacío.

Calentamos la solución y hacemos que ésta fluya hacia el nuevo tubo de datos que estaba vacío. Con esto se consigue que los complejos que tenían adheridas las muestras/stickers de la región i-ésima, rompan ese enlace y se vayan al nuevo tubo. Despegamos este tubo, y lo colocamos en la estantería: es el tubo $-(T, i)$. Desechamos el tubo de datos original y volvemos a colocar en la estantería el tubo operador extracción i-ésima que ha sido usado.

Activar(T, i).

Seleccionamos de la estantería el tubo de datos correspondiente a T, así como el tubo de datos que contiene stickers suplementarios de la región i-ésima. Seleccionamos de la estantería el tubo operador sticker.

Hacemos que la solución fría fluya en ambas direcciones durante un momento. Esto permite que los stickers se enlacen con los complejos que tengan desactivada la región i-ésima. A continuación hacemos que la solución fría fluya hacia el tubo de datos sticker.

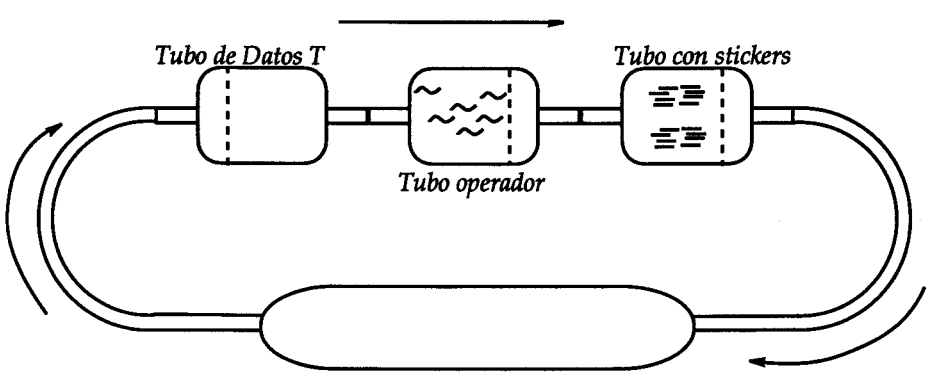


Figura 3.12. Proceso de activación

De esta manera todos los stickers no usados pasarán al tubo de datos stickers y los complejos quedarán retenidos en el filtro del tubo operador.

Desconectamos el tubo de datos stickers y lo colocamos en la estantería. Lo reemplazamos por un nuevo tubo de datos vacío. Hacemos ahora que la solución fría fluya hacia el tubo de datos T. Esto hace que todos los complejos pasen al tubo T.

Colocamos el tubo T en la estantería y desechamos el tubo operador y el tubo de datos vacío.

Desactivar(T, i).

Seleccionamos de la estantería el tubo de datos correspondiente a T, así como el tubo de datos que contiene anti-stickers correspondientes a la región i-ésima (es decir, cadenas PNA que, por invasión, forman una triple hélice que desestabiliza la región i-ésima). Seleccionamos de la estantería el tubo operador sticker.

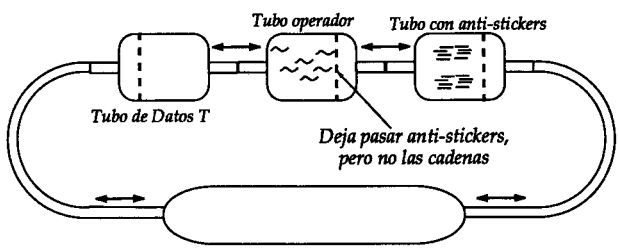


Figura 3.13.

Hacemos que la solución fría fluya en ambas direcciones durante un momento. Esto permite la invasión de las cadenas de PNA dando lugar a triple hélices en la región i-ésima.

A continuación hacemos que la solución fría fluya hacia el tubo de datos de los anti-stickers. De esta manera todos los anti-stickers no usados se depositan en el tubo que contiene anti-stickers.

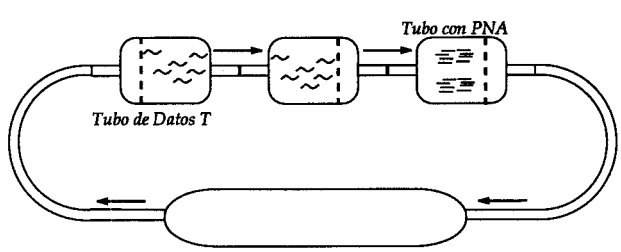


Figura 3.14.

Desconectamos este tubo y lo colocamos en la estantería. Ponemos en su lugar un tubo vacío.

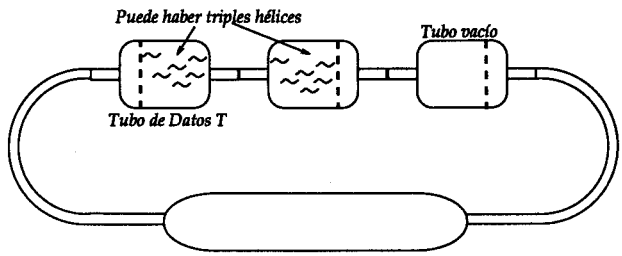


Figura 3.15.

Calentamos un poco hasta conseguir que se rompan las triple hélices pero no otros enlaces y hacemos que la solución fluya hacia el tubo vacío. Entonces en este tubo se depositarán todos los anti-stickers PNA que habían formado las triple hélices.

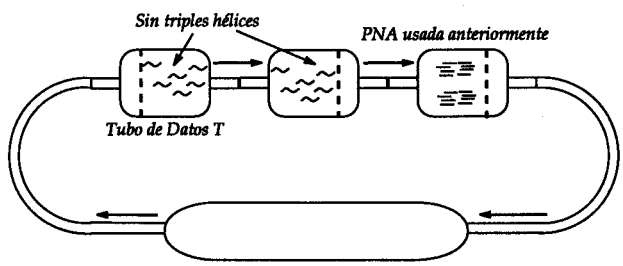


Figura 3.16.

Desconectamos este tubo, lo colocamos en la estantería y lo sustituimos por otro vacío.

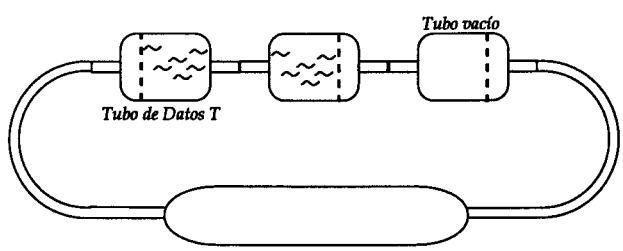


Figura 3.17.

Entonces hacemos que la solución fluya hacia el tubo T, en donde quedarán depositados todos los complejos que tienen desactivada la región i-ésima.

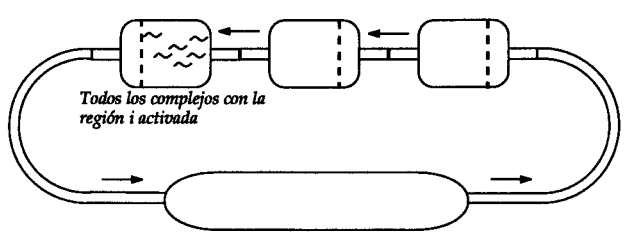


Figura 3.18.

Colocamos el tubo T en la estantería y desechamos el tubo operador y el tubo de datos vacío.

Nota 3.15 El paralelismo se puede conseguir de muchas maneras. Por ejemplo, las operaciones de activar y desactivar un determinado bit pueden ejecutarse simultáneamente por la máquina que hemos descrito, basta apilar los tubos de datos correspondientes tras el tubo operador:

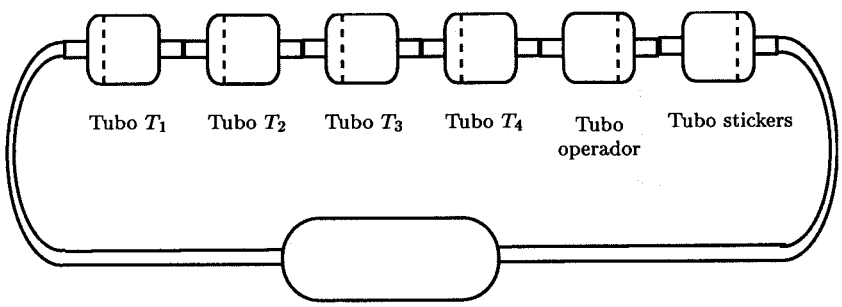


Figura 3.19.

También se necesitarían muchos robots para poder ejecutar varias operaciones (eventualmente distintas) de manera simultánea.

La máquina sticker descrita requiere pocos elementos: bombas de líquidos y calentadores-enfriadores. Se necesitaría un banco de:

- Tubos de datos vacíos.
- Tubos operadores en blanco.
- Tubos operadores stickers.
- Soluciones con varias concentraciones.
- Tubos de datos conteniendo las cadenas de memoria iniciales.
- Tubos de datos conteniendo los stickers asociados a cada región.
- Tubos operadores de extracción asociados a cada región.

Un punto importante a destacar en el diseño presentado es que todos estos tubos son reutilizables de un problema a otro.

3.4. Universalidad de los modelos moleculares

Las soluciones que proporciona la manipulación de moléculas de ADN para problemas **NP**-completos demuestran las interesantes propiedades computacionales que subyacen en las moléculas de ADN. Ahora bien, estos resultados podrían no ser relevantes si no existiera un computador molecular.

El diseño de máquinas de Turing basado en moléculas de ADN es un paso en esta dirección, ya que define un sistema de computación capaz de mantener un estado y una memoria, así como ejecutar un número indefinido de transiciones (pasos de computación).

En [7], D. Beaver diseña una máquina de Turing que consta de una cadena simple de ADN, basándose en la similitud existente entre la cinta de la máquina y la cadena simple de ADN (ambas son lineales y almacenan información por medio de un alfabeto finito). Esto justificará la universalidad de todo modelo de computación molecular que pueda implementar las operaciones utilizadas en esta simulación.

A continuación se describen las especificaciones concretas de la máquina de Turing que D. Beaver simuló a través de procedimientos moleculares.

Una máquina de Turing, M, es una 7-tupla, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F, B)$, en donde:

- Q es un conjunto finito no vacío (cuyos elementos se denominan *estados*).
- Σ es un conjunto finito no vacío, denominado *alfabeto de entrada.*
- Γ es un conjunto finito no vacío tal que $\Sigma \subseteq \Gamma$, denominado *alfabeto de trabajo.*
- δ es una función parcial de $Q \times \Gamma$ en $Q \times \Gamma \times \{0, 1, -1\}$, denominada *función de transición.*
- $q_0 \in Q$, que se denomina *estado inicial.*
- $q_F \in Q - \{q_0\}$, que se denomina *estado final.*
- $B \in \Gamma - \Sigma$, que se denomina *símbolo blanco.*

La máquina consta de una *cinta* que posee una cantidad numerable de *casillas* o *celdas.* La cinta es ilimitada a la derecha y posee primera casilla. Cada celda de la cinta contiene un símbolo del alfabeto de trabajo.

La máquina de Turing dispone de una *cabeza de trabajo* lectora/escritora que, en cada instante, analiza una casilla. Al realizar ese análisis puede escribir un nuevo símbolo en la celda analizada, borrando el símbolo que contenía. Además, puede desplazarse una casilla a la derecha, una casilla a la izquierda (si no está analizando

la primera casilla), o bien permanecer en el mismo sitio analizando otra vez la misma casilla en el instante siguiente.

Si $\delta(q,s) = (q',s',x)$ diremos que la máquina M pasa del estado q al estado q', sustituye el símbolo s (de la casilla que está analizando la cabeza de trabajo) por el símbolo s', y la cabeza se mueve según indica el valor de x ($1 =$ a la derecha; $-1 =$ a la izquierda; $0 =$ permanece en el mismo lugar). Además, si $\triangleright$ es la primera casilla de la cinta, entonces se exige que $\delta(q,\triangleright) = (q',s',x) \rightarrow s' = \triangleright \wedge x \in \{0,1\}$.

Una *descripción o configuración instantánea* de la máquina de Turing M es una cadena wqx, en donde $w \in \Gamma^*$, $q \in Q$, y $x \in (\Gamma - \{B\})^+$. La descripción instantánea wqx se puede interpretar como sigue: wx es la palabra escrita en la cinta (*desde la primera casilla en adelante*, q es el estado en el que se encuentra la máquina en ese instante, y la cabeza de trabajo está analizando el primer símbolo de x. En esta interpretación, admitiremos que toda casilla situada a la derecha del último símbolo de x contiene el símbolo blanco B.

Diremos que una descripción instantánea $wqsy$, con $w \in \Gamma^*$, $q \in Q$, $s \in \Gamma - \{B\}$ e $y \in (\Gamma - \{B\})^*$, es una *descripción de parada* si $\delta(q,s) \uparrow$. Además, diremos que dicha descripción es de *aceptación* si es de parada y $q = q_F$.

Si $w \in \Sigma^*$, entonces la *descripción instantánea inicial* de M sobre w, que notaremos I_w, es $q_0 w$.

Sobre el conjunto de descripciones instantáneas de M se define la relación $\vdash_M$ como sigue:

$$wsqs'y \vdash_M wq'ss''y \Leftrightarrow \delta(q,s') = (q',s'',-1)$$

$$wsqs'y \vdash_M wsq's''y \Leftrightarrow \delta(q,s') = (q',s'',0)$$

$$wsqs'y \vdash_M wss''q'y \Leftrightarrow \delta(q,s') = (q',s'',+1)$$

Dadas dos descripciones instantáneas, I_1 y I_2, de M, diremos que I_2 se obtiene de I_1 tras ejecutar un paso si $I_1 \vdash_M I_2$.

Sobre las descripciones instantáneas de M se define la relación $\vdash_M^*$ como el cierre reflexivo y transitivo de $\vdash_M$. Es decir, diremos que la descripción instantánea I_2 se obtiene de la descripción instantánea I_1 tras ejecutar k pasos en la computación de M si existen descripciones instantáneas $J_1, J_2, \ldots, J_{k+1}$ tales que

$$J_1 = I_1 \wedge J_{k+1} = I_2 \wedge \forall i \; (1 \leq i \leq k \rightarrow J_i \vdash_M J_{i+1})$$

Si $w \in \Sigma^*$, entonces diremos que la máquina M *para* sobre w, y notaremos $M \downarrow w$, si existe una descripción instantánea de parada, I, tal que $I_w \vdash_M^* I$. Caso contrario, diremos que M no para sobre w, y notaremos $M \uparrow w$.

Si $M \downarrow w$, entonces se prueba fácilmente que existe una única descripción instantánea, I_w^f, tal que es de parada y, además, $I_w \vdash_M^* I_w^f$.

El *lenguaje aceptado o reconocido* por la máquina de Turing M se define como sigue:

$$L(M) = \{w \in \Sigma^* : \ M \downarrow w \wedge I_w^f \text{ es una descripción instantánea de aceptación}\}$$

Diremos que un lenguaje L sobre Σ es *recursivamente enumerable* si existe una máquina de Turing, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F, B)$, tal que $L(M) = L$. Diremos que el lenguaje L sobre Σ es *recursivo* si existe una máquina de Turing, M, tal que $L(M) = L$ y, además, para cada $w \in \Sigma^*$ se tiene que $M \downarrow w$.

La simulación de máquinas de Turing arbitrarias a través de la manipulación de moléculas de ADN permite establecer la universalidad de aquellos modelos de computación molecular que son capaces de implementar todas las operaciones necesarias para realizar dicha simulación.

La simulación presentada D. Beaver requiere desarrollar los siguientes pasos:

1. Codificar las descripciones instantáneas (en particular, las iniciales) a través de moléculas de ADN.

2. Determinar la codificación de la descripción instantánea siguiente de una descripción instantánea que no es de parada, especificada a partir de su código.

3. Decidir cuándo es de parada una descripción instantánea codificada por una molécula de ADN y, en su caso, determinar si es o no de aceptación.

En primer lugar, vamos a proceder a codificar las descripciones instantáneas de una máquina de Turing a través de moléculas de ADN.

Para ello, supongamos que f es una función 1-aria total computable tal que para cada dato de entrada de tamaño n, la máquina de Turing M usa a lo sumo $f(n)$ casillas. Entonces se considera una función de codificación:

$$E : \{1, \ldots, f(n)\} \times (Q \cup \Gamma) \rightarrow \Sigma_{ADN}^*$$

que satisfaga las condiciones siguientes:

- $E(i,j)$ y $E(i',j')$ no deben poseer subcadenas *relevantes* iguales, a menos que $i = i'$ y $j = j'$.

- $E(i,j)$ y $\overline{E}(i',j')$ (donde $\overline{E}(i',j')$ es la cadena complementaria a $E(i',j')$) no deben poseer subcadenas *relevantes* iguales, a menos que $i = i'$ y $j = j'$.

- Si $s \in \Gamma$, entonces $E(i, s)$ indica que el símbolo s está en la posición i-ésima de la cinta.
- Si $q \in Q$, entonces $E(i, q)$ indica que la máquina M está en el estado q y la cabeza de trabajo está analizando la casilla i-ésima de la cinta de M.

Una función codificadora, E, verificando las condiciones antes citadas puede ser efectivamente implementada en el laboratorio.

A partir de la función E, se codifican las descripciones instantáneas de M de la siguiente forma: si wqx es una descripción instantánea de M, con $w = w_1 \dots w_r$ y $x = x_1 \dots x_s$, entonces dicha descripción instantánea se codifica por la cadena de ADN siguiente:

$$E(1, w_1) \dots E(r, w_r)E(r+1, q)E(r+1, x_1) \dots E(r+s, x_s)E(r+s+1, B) \dots E(f(n), B)$$

Una vez codificadas las descripciones instantáneas por moléculas de ADN, el siguiente paso para la simulación de la máquina de Turing M consiste en implementar la función de transición, δ.

Para ello, vamos a ver previamente cómo es posible realizar en el laboratorio una operación que permite sustituir un cierto trozo en una cadena de ADN por otro trozo. Es decir, se trata de realizar una lectura parcial de la cadena simple que codifica una cierta descripción instantánea I (en un entorno que correspondería a la casilla analizada por la cabeza de trabajo) y, posteriormente, a través de la operación de sustitución se modifica el trozo correspondiente a fin de obtener la descripción instantánea $\delta(I)$.

Concretamente, la operación de sustitución consiste en lo siguiente:

- Se parte de una cadena simple, $\gamma = LAXBR$, en donde L, A, X, B, R son las correspondientes subcadenas que integran γ: L es la parte izquierda de la cadena, R es la parte derecha, AXB es la parte de la cadena donde se efectúa la lectura y X es el trozo que se va a sustituir.
- Se parte, además, de otra cadena simple Y.
- Se trata de generar la cadena $\gamma' = LAYBR$.

Esta operación se notará por $SUST(A, X, B, Y)$ y se implementa en el laboratorio como sigue:

1. Se sintetizan las cadenas $\overline{AYB}$ y $\overline{R}$, y se añaden a la solución donde se encuentra la cadena simple $\gamma = LAXBR$.

2. Se somete la solución a un proceso de renaturalización.
3. Usando la polimerasa y comenzando por $\overline{R}$ se *escribe* la cadena $\overline{L}$.
4. Usando la ligasa se rellenan los huecos que pudieran quedar.
5. Se somete la solución a un proceso de desnaturalización.
6. Se extrae de la solución la cadena $3' - \overline{LAYBR}$ utilizando el método de las sondas con microesferas metálicas que llevan adheridas $5' - L$.
7. A la solución que contiene $3' - \overline{LAYBR}$ se le añade la cadena $5' - L$ y se somete la solución a un proceso de renaturalización.
8. Usando la polimerasa se extiende la cadena superior, y con la ligasa se sellan los huecos que pudieran aparecer.
9. Se somete la solución a un proceso de desnaturalización.
10. Se extrae de la solución la cadena $5' - LAYBR$ utilizando el método de las sondas con microesferas metálicas que llevan adheridas $3' - \overline{L}$.

A partir de la operación de sustitución antes descrita, veamos cómo se implementa la función de transición, δ, de la máquina de Turing M.

Sea I una descripción instantánea de M que tiene q como estado y con la cabeza en la posición i-ésima apuntando al símbolo $s \in \Gamma$. La codificación de la descripción I será la siguiente:

$$LE(i-2, s_{--}E(i-1, s_-)E(i, q)E(i, s)E(i+1, s_+)E(i+2, s_{++})R$$

En donde s_{--} y s_- son los símbolos que ocupan las posiciones anteriores al símbolo s, y s_+ y s_{++} son los símbolos que ocupan las posiciones posteriores al símbolo s.

- **Caso 1**: Supongamos que $\delta(q, s) = (q', s', -1)$. En este caso, se considera

$$\begin{aligned} A = &\quad E(i-2, s_{--}) \\ X = &\quad E(i-1, s_-)E(i, q)E(i, s) \\ B = &\quad E(i+1, s_+)E(i+2, s_{++}) \\ Y = &\quad E(i-1, q')E(i-1, s_-)E(i, s') \end{aligned}$$

  Entonces, se tiene que $\delta(I) = SUST(A, X, B, Y)$.

- **Caso 2**: Supongamos que $\delta(q, s) = (q', s', +1)$. En este caso, se considera

$$\begin{aligned} A = &\quad E(i-2, s_{--})E(i-1, s_{-}) \\ X = &\quad E(i,q)E(i,s)E(i+1,s_{+}) \\ B = &\quad E(i+2, s_{++}) \\ Y = &\quad E(i,s')E(i+1,q')E(i+1,s_{+}) \end{aligned}$$

Entonces, se tiene que $\delta(I) = SUST(A, X, B, Y)$.

- **Caso 3**: Supongamos que $\delta(q,s) = (q', s', 0)$. En este caso, se considera

$$\begin{aligned} A = &\quad E(i-2, s_{--})E(i-1, s_{-}) \\ X = &\quad E(i,q)E(i,s) \\ B = &\quad E(i+1, s_{+})E(i+2, s_{++}) \\ Y = &\quad E(i,q')E(i,s') \end{aligned}$$

Entonces, se tiene que $\delta(I) = SUST(A, X, B, Y)$.

En resumen, la implementación de la función de transición δ en el laboratorio para simular la máquina de Turing, M, se realizaría como sigue:

- Para la creación del tubo inicial, que debe codificar la descripción instantánea inicial para el dato de entrada $w \in \Sigma^*$, se sintetiza la cadena

$$E(1, q_0)E(1, w_1)\ldots E(|w|, w_{|w|})E(|w|+1, B)\ldots E(f(n), B)$$

en el tubo inicial T_0.

- Para los pasos de transición, se colocan suficientes oligos sintéticos del tipo $E(i,q)$, $E(i,s)$ para cada $i \in \{1,\ldots,f(n)\}$, $q \in Q$ y $s \in \Gamma$. A continuación se procede en dos fases:

1. En una primera fase, se procede a separar moléculas en tubos que clasifiquen las distintas descripciones instantáneas según el uso posterior que se les va a dar:

   para cada $q \in Q \wedge i \in \{1,\ldots,f(n)\}$ hacer
   $T(i,q) \leftarrow$ extraer$(T_0, E(i,q))$
   para cada $q \in Q \wedge i \in \{1,\ldots,f(n)\} \wedge s \in \Gamma$ hacer
   $T(i,q,s) \leftarrow$ extraer$(T(i,q), E(i,s))$
   para cada $q \in Q \wedge i \in \{2,\ldots,f(n)\} \wedge s, s_{-} \in \Gamma$ hacer
   $T(i,q,s,s_{-}) \leftarrow$ extraer$(T(i,q,s), E(i-1, s_{-}))$
   para cada $q \in Q \wedge i \in \{2,\ldots,f(n)\} \wedge s, s_{-}, s_{+} \in \Gamma$ hacer
   $T(i,q,s,s_{-},s_{+}) \leftarrow$ extraer$(T(i,q,s,s_{+}), E(i+1, s_{+}))$

2. En una segunda fase se implementa la función de transición:
   para cada $q, q' \in Q \wedge s, s' \in \Gamma$ tales que $\delta(q, s) = (q', s', -1)$ hacer
     para cada $i \in \{3, \ldots, f(n) - 1\} \wedge s_-, s_+, s_{--} \in \Gamma$ hacer
       $T(i, q, s, s_-, s_+, s_{--}) \leftarrow \texttt{extraer}(T(i, q, s, s_-, s_+), E(i-2, s_{--}))$
       $A \leftarrow E(i-2, s_{--})$
       $X \leftarrow E(i-1, s_-)E(i, q)E(i, s)$
       $B \leftarrow E(i+1, s_+)$
       $Y \leftarrow E(i-1, q')E(i-1, s_-)E(i, s')$
       Ejecutar $SUST(A, X, B, Y)$ en $T(i, q, s, s_-, s_+, s_{--})$
       $T_0 \leftarrow T_0 \cup T(i, q, s, s_-, s_+, s_{--})$
   para cada $q, q' \in Q \wedge s, s' \in \Gamma$ tales que $\delta(q, s) = (q', s', +1)$ hacer
     para cada $i \in \{2, \ldots, f(n) - 2\} \wedge s_-, s_+, s_{++} \in \Gamma$ hacer
       $T(i, q, s, s_-, s_+, s_{++}) \leftarrow \texttt{extraer}(T(i, q, s, s_-, s_+), E(i+2, s_{++}))$
       $A \leftarrow E(i-1, s_-)$
       $X \leftarrow E(i, q)E(i, s)E(i+1, s_+)$
       $B \leftarrow E(i+2, s_{++})$
       $Y \leftarrow E(i, s')E(i+1, q')E(i+1, s_+)$
       Ejecutar $SUST(A, X, B, Y)$ en $T(i, q, s, s_-, s_+, s_{++})$
       $T_0 \leftarrow T_0 \cup T(i, q, s, s_-, s_+, s_{++})$
   para cada $q, q' \in Q \wedge s, s' \in \Gamma$ tales que $\delta(q, s) = (q', s', 0)$ hacer
       $A \leftarrow E(i-1, s_-)$
       $X \leftarrow E(i, q)E(i, s)$
       $B \leftarrow E(i+1, s_+)$
       $Y \leftarrow E(i, q')E(i, s')$
       Ejecutar $SUST(A, X, B, Y)$ en $T(i, q, s, s_-, s_+)$
       $T_0 \leftarrow T_0 \cup T(i, q, s, s_-, s_+)$

- Para la aceptación o rechazo de la descripción instantánea admitiremos que la entrada es aceptada si y sólo si el símbolo 1 está en la casilla situada más a la izquierda, y de rechazo en caso de que sea el símbolo 0 el que esté en dicha casilla. En consecuencia, la implementación de que M acepta w será

$$\texttt{detectar}(\texttt{extraer}(\texttt{T}_{\texttt{out}}, \texttt{E}(1, \texttt{q}_\texttt{F})\texttt{E}(1, 1)))$$

y la implementación de que M rechaza w será

$$\texttt{detectar}(\texttt{extraer}(\texttt{T}_{\texttt{out}}, \texttt{E}(1, \texttt{q}_\texttt{F})\texttt{E}(1, 0)))$$

siendo T_{out} el tubo de salida de la computación de M sobre w.

Capítulo 4

Verificación formal en el modelo Sticker

Cuando se diseña un algoritmo o programa, el objetivo final es la resolución de un problema. Pero, ¿cómo podemos estar seguros de que dicho programa es correcto? Es decir, ¿cómo saber si dicho algoritmo resuelve realmente el problema abstracto para el que ha sido diseñado?.

Hasta mediados de la década de los sesenta, la única técnica usada para responder a la cuestión planteada consistía en ejecutar dicho algoritmo sobre una muestra finita de datos de entrada e ir depurando su diseño de acuerdo con los resultados obtenidos (lo que se denomina *test* y *debugging*). Con la aparición de algoritmos cada vez más complejos queda patente que este método de *corrección* es del todo insuficiente. Como hace notar E.W.Dijkstra [20], dicha técnica, que se podría denominar **validación dinámica**, *sólo pondrá de manifiesto la presencia de errores, pero nunca la ausencia de ellos.*

Como consecuencia del teorema de Rice, se deduce que el problema de la corrección no es resoluble algorítmicamente; es decir, no existe un procedimiento mecánico general para determinar si, dado un algoritmo diseñado para resolver un problema abstracto, devuelva **SI** en caso de que el algoritmo resuelva el problema, y devuelva **NO** en caso contrario.

Es por ello que surge la necesidad de crear una **validación estática**, frente a la anterior, que consista en *obtener información a priori sobre el comportamiento del programa mediante el análisis de su propio texto* [5] para, posteriormente, demostrar la corrección del mismo en un contexto puramente matemático. En otras palabras, utilizar una metodología que haga uso de métodos analíticos, cuya finalidad sea la de probar un teorema que se pueda *interpretar* como: *el algoritmo es correcto.* Para

ello, dicha metodología, que podríamos denominar **verificación formal**, requeriría:

- Un *método de especificación formal*, que proporcione los criterios para considerar correcto un algoritmo.
- Una *semántica formal del lenguaje de programación*, que permita considerar al algoritmo como objeto matemático.
- Unas *reglas formales de demostración*, que proporcionen los medios para establecer la veracidad del teorema de corrección buscado.

En este capítulo vamos a estudiar procesos de verificación de programas en el modelo sticker. Para ello, comenzamos en la primera sección analizando la posibilidad de describir los programas moleculares diseñados para resolver problemas de decisión como sistemas formales. La segunda sección está dedicada a la representación de conjuntos finitos en el modelo sticker. Para ello diseñaremos y verificaremos programas en el modelo citado que resuelven problemas que nos permitirán atacar la resolución de otros problemas numéricos **NP**-completos clásicos.

4.1. Programas moleculares como sistemas formales

Para verificar formalmente un programa molecular diseñado para resolver un problema, hay que demostrar dos resultados básicos:

(a) Toda molécula del tubo de salida representa una solución correcta del problema (*corrección del programa molecular*); es decir, si el tubo de salida no está vacío, entonces toda molécula de dicho tubo codifica una solución correcta del problema. Con otras palabras, para un problema de decisión, si el programa devuelve SI, entonces la respuesta del problema es SI.

(b) Toda molécula del tubo inicial que codifica una solución correcta del problema debe estar en el tubo de salida (*completitud del programa molecular*); es decir, si el tubo de salida está vacío, entonces no existe ninguna solución correcta del problema. Con otras palabras, para un problema de decisión, si el programa devuelve NO, entonces la respuesta del problema es NO.

A continuación, vamos a describir como sistemas formales, los programas diseñados para resolver problemas de decisión; de tal manera que demostrar la verificación de un programa molecular sea equivalente a establecer la adecuación y completitud del sistema formal asociado.

Un *sistema formal* consta, en esencia, de

(a) una *sintaxis*, obtenida a partir de un alfabeto prefijado y de una regla de formación de fórmulas del sistema;

(b) una *semántica* que proporciona el concepto de *validez* ; y

(c) una *deducción* que proporciona el concepto de demostrabilidad de una fórmula en el sistema.

Se dice que un sistema formal es *adecuado* si y sólo si toda fórmula demostrable es válida, y se dice que es *completo* si y sólo si toda fórmula válida es demostrable.

Un *problema de decisión*, X, es una aplicación de un cierto conjunto $E_X \subseteq \Sigma_X$ en $\{0,1\}$. Con otras palabras, informalmente, se dice que un problema es de decisión si únicamente admite como respuesta SI o NO. Los elementos de E_X se denominan ejemplares o datos de entrada del problema, y Σ_X es el alfabeto de entrada del mismo.

Sea P un programa molecular diseñado para resolver un problema de decisión, X. Sea Σ_P el alfabeto sobre el que se describe P. Para cada ejemplar del problema, $a \in E_X$, notaremos $I(P,a)$ al conjunto de tubos iniciales del programa P relativos al dato de entrada a. Para cada ejemplar del problema, $a \in E_X$, y cada tubo inicial, $T \in I(P,a)$, notaremos $P(a,T)$ el resultado de ejecutar el programa molecular P con dato de entrada a y tubo de ensayo inicial T.

Definición 4.1 *Sea P un programa molecular diseñado para resolver un problema de decisión, X.* Verificar *(X,P) consiste en demostrar que el programa molecular P resuelve el problema de decisión X.*

Es decir, verificar (X,P) consiste en probar que para cada ejemplar del problema, $a \in E_X$, y cada tubo inicial, $T \in I(P,a)$, se tiene que $P(a,T) = 1$ si y sólo si $X(a) = 1$. En esta equivalencia, la implicación directa recibe el nombre de *corrección* del programa P para el problema X, y la implicación recíproca el de *completitud* del programa P para el problema X.

Definición 4.2 *Sea P un programa molecular diseñado para resolver un problema de decisión, X. Diremos que (X,P) está verificado si y sólo si para cada $a \in E_X$ y cada $T \in I(P,a)$ se tiene que $P(a,T) = 1$ si y sólo si $X(a) = 1$.*

Definición 4.3 *Sea P un programa molecular diseñado para resolver un problema de decisión, X. El sistema formal, $\mathcal{S}(X,P)$, asociado a (X,P) consta de:*

(a) Un alfabeto, $\Sigma = \Sigma_X \cup \Sigma_P$.

(b) *Un conjunto de fórmulas: una fórmula es un par ordenado* (a, T), *en donde* $a \in E_X$ *y* $T \in I(P, a)$.

(c) *Un concepto de validez: una fórmula* (a, T) *es válida si y sólo si* $X(a) = 1$.

(d) *Un concepto de deducción: una fórmula* (a, T) *es demostrable si y sólo si* $P(a, T) = 1$.

Proposición 4.4 *Sea* P *un programa molecular diseñado para resolver un problema de decisión,* X. *Se verifica:*

1. *El sistema* $\mathcal{S}(X, P)$ *es adecuado si y sólo si*

$$\forall\, a \in E_X\ \forall\, T \in I(P, a)\ (P(a, T) = 1 \Longrightarrow X(a) = 1)$$

2. *El sistema* $\mathcal{S}(X, P)$ *es completo si y sólo si*

$$\forall\, a \in E_X\ \forall\, T \in I(P, a)\ (X(a) = 1 \Longrightarrow P(a, T) = 1)$$

Corolario 4.5 *Sea* P *un programa molecular diseñado para resolver un problema de decisión,* X. *Entonces* (X, P) *está verificado si y sólo si el sistema formal asociado,* $\mathcal{S}(X, P)$, *es adecuado y completo.*

Pese a que la formalización anterior está encaminada a verificar programas moleculares que intentan resolver problemas de decisión, el mismo método puede ser usado (con pequeñas variaciones) para verificar programas que resuelvan problemas de tipo más general.

4.2. Representación de conjuntos finitos en el modelo sticker

En el capítulo siguiente vamos a diseñar y verificar programas moleculares en el modelo sticker que resuelven problemas **NP**-completos, algunos de los cuales no son de decisión. Antes de acometer esta tarea estudiaremos algunos problemas de menor envergadura pero que resultan de gran utilidad a la hora de resolver los problemas **NP**-completos que se estudian en el capítulo siguiente.

Los problemas **NP**-completos que se van a resolver en el capítulo siguiente, se encuadran dentro de lo que se denominan problemas de conjuntos numéricos (véase [26]). La característica común a todos ellos es el hecho de que plantean problemas sobre conjuntos numéricos, unas veces de recubrimiento, otras de particiones y otras asociados a funciones de pesos.

Por ello, como paso previo a la resolución de dichos problemas, vamos a describir un método para representar conjuntos numéricos en el modelo sticker. Ya vimos, en la introducción de este modelo, que todo complejo de memoria del tipo (n, k, m) puede ser identificado con un número binario de k dígitos, de modo que el i-ésimo dígito de dicho número es 1 si y sólo si la región i-ésima del complejo está activada. Una vez realizada esta identificación, se puede asociar de manera natural un subconjunto, A, del conjunto $\{1, \ldots, k\}$, a cada (n, k, m)-complejo de memoria, σ, de acuerdo con el siguiente criterio:

$$\forall i \ (1 \leq i \leq k \rightarrow (i \in A \Leftrightarrow \sigma(i) = 1))$$

Por abuso del lenguaje, y apoyándonos en esta identificación, escribiremos $i \in \sigma$ para el caso en que $1 \leq i \leq k$ y $\sigma(i) = 1$, e $i \notin \sigma$ en caso contrario.

Para facilitar la notación que se usará en los diseños y verificación de los programas que se van a dar a continuación, consideramos las siguientes operaciones entre conjuntos numéricos:

Definición 4.6 *Dados $A, B \subseteq \mathbf{N}$ y $p \in \mathbf{N}$, se definen:*

$$A \cup B = \{x \in \mathbf{N} : \ x \in A \vee x \in B\}$$
$$A \cap B = \{x \in \mathbf{N} : \ x \in A \wedge x \in B\}$$
$$p + A = \{p + x : \ x \in A\}$$
$$A - p = \{x - p : \ x \in A\}, \text{ supuesto que } \forall x \in A \ (p \leq x)$$

En lo que sigue, si σ es un complejo de memoria, entonces notaremos con el mismo símbolo tanto el número binario como el conjunto asociado a σ.

Definición 4.7 *Sea σ un (n, k, m)-complejo de memoria y $p, q \in \mathbf{N}$ tales que $1 \leq p \leq q \leq k$. Se define $\sigma_{[p,q]} = \sigma \cap [p, q]$.*

4.2.1. El problema del rellenado

Problema: *Sea $A = \{1, \ldots, p\}$. Sea $\mathcal{F} = \{B_1, \ldots, B_q\}$ una familia finita de subconjuntos de A. Determinar todos los pares ordenados $(\mathcal{F}', B)$, en donde $\mathcal{F}'$ es una subfamilia de $\mathcal{F}$ y $B = \bigcup \mathcal{F}'$.*

Para cada i tal que $1 \leq i \leq q$ notamos $r_i = |B_i|$ y $B_i = \{x_i^1, \ldots, x_i^{r_i}\}$.

Para resolver este problema en el modelo sticker consideramos como tubo de ensayo inicial, T_0, una $(p + q, q)$-librería.

Sea σ un complejo de memoria con $p + q$ regiones (en adelante diremos simplemente que σ es una molécula). Podemos interpretar que σ codifica un par ordenado,

$(\mathcal{F}_\sigma, A_\sigma)$, formado por la subfamilia $\mathcal{F}_\sigma = \{B_k : k \in \sigma_{[1,q]}\}$ de $\mathcal{F}$ y el subconjunto $A_\sigma = \sigma_{[q+1,q+p]} - q$ de A.

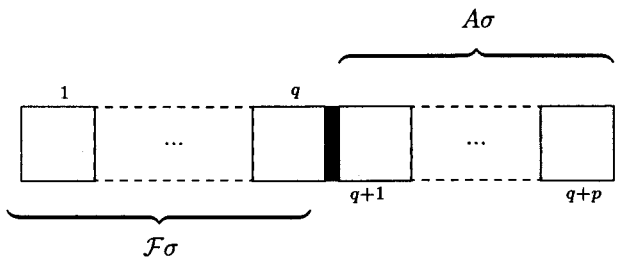


Figura 4.1. Complejo de memoria de una $(p+q, q)$-librería

Definición 4.8 *Diremos que la molécula σ es consistente si verifica que $A_\sigma = \bigcup \mathcal{F}_\sigma$.*

Un programa molecular que resuelve el problema del rellenado en el modelo sticker es el siguiente:

```
Procedimiento Rellenado
Entrada: T_0 (una (p+q,q)-librería)
    para i = 1 hasta q hacer
        T_i^+ ← +(T_0, i);  T_i^- ← −(T_0, i)
        para j = 1 hasta r_i hacer
            activar(T_i^+, q + x_i^j)
        T_0 ← T_i^+ ∪ T_i^−
```

El número de operaciones moleculares que realiza este programa es del orden $O(q \cdot p)$. El volumen molecular del tubo inicial es 2^q, y el número de tubos usados es del orden de $O(q)$.

Verificación del programa molecular diseñado

A efectos de establecer la verificación formal de este programa procedemos a un re-etiquetado de tubos.

```
Procedimiento Rellenado
Entrada: T_0 (una (p+q,q)-librería)
    para i = 1 hasta q hacer
        T_i^+ ← +(T_{i−1}, i);  T_i^- ← −(T_{i−1}, i)
        T*_{i,0} ← T_i^+
        para j = 1 hasta r_i hacer
            T*_{i,j} ← activar(T*_{i,j−1}, q + x_i^j)
        T_i ← T*_{i,r_i} ∪ T_i^−
```

Conviene resaltar el hecho de que la semántica de este programa molecular está muy próxima a la propia semántica del problema que se trata de resolver. Las dificultades que aparecen a la hora de establecer la verificación del mismo se derivan principalmente del hecho de usar operaciones moleculares (`activar`) que alteran la estructura interna de las moléculas, lo que complica el análisis del rastro de las moléculas de los tubos *relevantes*, $\{T_i : 1 \leq i \leq q\}$, del programa a lo largo de la ejecución. Para poder seguir esa huella definiremos una función que ponga de manifiesto la transformación que sufren las moléculas a lo largo de la ejecución del programa.

La función STEP

A continuación se define una función, que denominaremos $STEP$, y que trata de capturar la siguiente idea: $STEP(\sigma, i)$ es la molécula obtenida a partir de $\sigma \in T_{i-1}$ tras la ejecución del paso i-ésimo del bucle principal.

Definición 4.9 *Sean i (con $1 \leq i \leq q$) y $\sigma \in T_{i-1}$.*

- *Si $i \notin \sigma$, entonces se define $STEP(\sigma, i) = \sigma$.*
- *Si $i \in \sigma$, entonces se define $STEP(\sigma, i) = \sigma \cup (q + B_i)$. Es decir, $STEP(\sigma, i) = \tau$, donde $\tau = \sigma$ excepto, quizás, en los elementos $q + x_i^j$ (con $1 \leq j \leq r_i$), en donde $\tau(q + x_i^j) = 1$.*

Obsérvese que la función $STEP$ es total; es decir, siempre está definida sobre los datos de entrada. De la definición anterior resulta inmediatamente que si $1 \leq i \leq q$, $\sigma \in T_{i-1}$ y $\tau = STEP(\sigma, i)$, entonces $\sigma_{[1,q]} = \tau_{[1,q]}$ y, además, $\sigma \subseteq \tau$.

El siguiente lema nos indica que, en efecto, la función $STEP$ define una molécula que está en el tubo relevante del siguiente paso de la ejecución del bucle.

Lema 4.10 $\forall i \; (1 \leq i \leq q \rightarrow \forall \sigma \in T_{i-1} \; (STEP(\sigma, i) \in T_i))$.

Demostración:
Sea i tal que $1 \leq i \leq q$ y $\sigma \in T_{i-1}$. Distingamos dos casos:

- **Caso 1**. Supongamos que $i \notin \sigma$.

  En este caso $\sigma \in -(T_{i-1}, i) = T_i^- \subseteq T_i$. Como $STEP(\sigma, i) = \sigma$, se deduce que $STEP(\sigma, i) \in T_i$.

- **Caso 2**. Supongamos que $i \in \sigma$.

Definimos recursivamente $\sigma_{(j)}$, para $0 \leq j \leq r_i$, como sigue:

$$\begin{cases} \sigma_{(0)} = \sigma \\ \sigma_{(j+1)} = \sigma_{(j)} \cup \{q + x_i^{j+1}\} \end{cases}$$

Es decir $\sigma_{(j+1)} = \rho_{(j)}$ excepto, quizás, en $q + x_i^{j+1}$, donde $\sigma_{(j+1)}(q + x_i^{j+1}) = 1$.

Veamos por inducción sobre j que $\forall j\ (0 \leq j \leq r_i \rightarrow \sigma_{(j)} \in T^*_{i,j})$.

El caso base, $j = 0$, es trivial, ya que $\sigma_{(0)} = \sigma \in +(T_{i-1}, i) = T_i^+ = T^*_{i,0}$.

Sea $j < r_i$ tal que $\sigma_{(j)} \in T^*_{i,j}$. De la definición de $\sigma_{(j+1)}$ se deduce que $\sigma_{(j+1)} \in \texttt{activar}(T^*_{i,j}, q + x_i^{j+1}) = T^*_{i,j+1}$.

De la definición de los $\sigma_{(j)}$, con $0 \leq j \leq r_i$, resulta que la molécula $\sigma_{(r_i)}$ coincide con $STEP(\sigma, i)$. En consecuencia: $STEP(\sigma, i) = \sigma_{(r_i)} \in T^*_{i,r_i} \subseteq T_i$.

□

Historia de una molécula a lo largo de la ejecución del programa

Definición 4.11 *Sea $\sigma \in T_0$. Definimos recursivamente σ^i, con $0 \leq i \leq q$, como sigue:*

$$\begin{cases} \sigma^0 = \sigma \\ \sigma^i = STEP(\sigma^{i-1}, i), \text{ para } 1 \leq i \leq q \end{cases}$$

El lema 4.10 justifica que esta definición es correcta. Además, se verifica que:

$$\forall i\ (0 \leq i \leq q \rightarrow \sigma^i \in T_i)$$

Definición 4.12 *Si $\sigma \in T_0$, entonces la historia de σ es $\hat{\sigma} = (\sigma^0, \sigma^1, \ldots, \sigma^q)$.*

El siguiente lema pone de manifiesto la transformación que sufren las moléculas en la ejecución del bucle secundario del programa.

Lema 4.13 *Para cada i tal que $1 \leq i \leq q$ y cada j tal que $1 \leq j \leq r_i$, se verifica que $\forall \tau \in T^*_{i,j}\ \exists \rho \in T_i^+\ (\tau = \rho \cup (q + B_i^j))$, en donde $B_i^j = \{x_i^1, \ldots, x_i^j\}$.*

Demostración:

Sea i tal que $1 \leq i \leq q$. Probemos por inducción débil sobre j que:

$$\forall j\ (1 \leq j \leq r_i \rightarrow \forall \tau \in T^*_{i,j}\ \exists \rho \in T_i^+\ (\tau = \rho \cup (q + B_i^j)))$$

Para probar el caso base, $j = 1$, sea $\tau \in T^*_{i,1} = \texttt{activar}(T^*_{i,0}, q + x^1_i) = \texttt{activar}(T^+_i, q + x^1_i)$. Entonces existe $\rho \in T^+_i$ tal que $\tau = \rho \cup \{q + x^1_i\}$. Es decir: $\tau = \rho \cup (q + B^1_i)$.

Sea $j < r_i$ $(j \geq 1)$ y supongamos cierto el resultado para j. Sea $\sigma \in T^*_{i,j+1} = \texttt{activar}(T^*_{i,j}, q + x^{j+1}_i)$. Existe $\rho' \in T^*_{i,j}$ tal que $\tau = \rho' \cup \{q + x^{j+1}_i\}$. Como $\rho' \in T^*_{i,j}$, por hipótesis de inducción existe $\rho \in T^+_i$ tal que $\rho' = \rho \cup (q + B^j_i)$. Luego $\tau = \rho' \cup \{q + x^{j+1}_i\} = \rho \cup (q + B^j_i) \cup \{q + x^{j+1}_i\} = \rho \cup (q + B^{j+1}_i)$, con $\rho \in T^+_i$.

□

Del lema anterior se deduce que la función $STEP$ antes definida refleja, en cierto sentido, la transformación que sufren las moléculas tras la ejecución completa del bucle secundario.

Corolario 4.14 *Para cada i tal que $1 \leq i \leq q$, se tiene que:*

$$\forall \tau \in T^*_{i,r_i}\ \exists \rho \in T_{i-1}\ (i \in \rho \wedge STEP(\rho, i) = \tau))$$

Demostración:
Sea i tal que $1 \leq i \leq q$. Sea $\tau \in T^*_{i,r_i}$. Aplicando el lema 4.13 al caso $j = r_i$ se deduce que existe $\rho \in T^+_i$ tal que $\tau = \rho \cup (q + B^{r_i}_i) = \rho \cup (q + B_i)$.

Como $\rho \in T^+_i = +(T_{i-1}, i)$, resulta que $\rho \in T_{i-1} \wedge i \in \rho$. De la definición de la función $STEP$, se deduce que $STEP(\rho, i) = \tau$.

□

Corrección del programa

Hemos de probar que toda molécula del tubo de salida, T_q, codifica una solución correcta del problema. Es decir, que si $\tau \in T_q$, entonces la subfamilia $\mathcal{F}_\tau$, de $\mathcal{F}$, y el subconjunto A_τ, de A, codificados por τ verifican que $A_\tau = \bigcup \mathcal{F}_\tau$. Dicho con otras palabras, hemos de probar que toda molécula, τ, del tubo de salida, T_q, es consistente.

Para ello, consideremos la fórmula

$$\theta(i) \equiv \forall \tau \in T_i\ \ [\forall k\ (1 \leq k \leq i \wedge k \in \tau \rightarrow (q + B_k) \subseteq \tau)\ \wedge$$
$$\wedge \forall s\ (1 \leq s \leq p \wedge (q + s) \in \tau \rightarrow \exists k \in \tau\ (1 \leq k \leq i \wedge s \in B_k))]$$

Es decir, la fórmula $\theta(i)$ expresa que tras la ejecución del paso i-ésimo del bucle principal, en el tubo T_i están exactamente las moléculas, τ, tales que

$$A_\tau = \bigcup\{B_k \in \mathcal{F}_\tau : \ 1 \leq k \leq i\}$$

Teorema 4.15 *La fórmula $\theta(i)$ es un invariante del bucle principal. Es decir,*

$$\forall i \ (1 \leq i \leq q \rightarrow \theta(i))$$

Demostración:
Por inducción débil sobre i:

Para probar el caso base, $i = 1$, sea $\tau \in T_1 = T^*_{1,r_1} \cup T_1^-$.

- **Caso 1**: Supongamos que $\tau \in T_1^-$. En este caso: $\tau \in T_0 \wedge 1 \notin \tau$. Como $\tau \in T_0$ se tiene que

  $$\forall s \ (1 \leq s \leq p \rightarrow (q+s) \notin \tau) \ [e1]$$

  Luego se verifica

  (a) $\forall k \ (1 \leq k \leq 1 \wedge k \in \tau \rightarrow (q + B_1) \subseteq \tau))$, ya que $1 \notin \tau$ y, por tanto, el antecedente es falso.

  (b) $\forall s \ (1 \leq s \leq p \wedge (q+s) \in \tau \rightarrow \exists k \in \tau \ (1 \leq k \leq 1 \wedge s \in B_1))$, ya que por $[e1]$ el antecedente también es falso.

  En consecuencia, en este caso sería verdadera la fórmula $\theta(1)$.

- **Caso 2**: Supongamos que $\tau \in T^*_{1,r_1}$. Del corolario 4.14 se deduce que existe $\rho \in T_0$ tal que $1 \in \rho \wedge STEP(\rho, 1) = \tau$. Por tanto, $\tau = \rho \cup (q + B_1)$ $[e2]$.

  (a) Sea k tal que $1 \leq k \leq 1 \wedge k \in \tau$. Esta situación se da realmente ya que $1 \in \rho \rightarrow 1 \in \tau$. Además por $[e2]$ se tiene que $(q + B_1) \subseteq \tau$.

  (b) Sea s $(1 \leq s \leq p)$ tal que $(q+s) \in \tau$. Como $\rho \in T_0$, se tiene que $(q+s) \notin \rho$ (pues T_0 es una $(p+q, q)$-librería). Por tanto, de $[e2]$ se deduce que $(q+s) \in (q + B_1)$, o lo que es lo mismo, $s \in B_1$.

  En consecuencia, también en este caso se verificaría $\theta(1)$.

Sea $i < q$ tal que $i \geq 1$, y supongamos cierto el resultado para i. Sea $\tau \in T_{i+1} = T^*_{i+1,r_{i+1}} \cup T^-_{i+1}$.

- **Caso 1**: Supongamos que $\tau \in T^-_{i+1}$.

  En este caso se tiene que $\tau \in T_i \wedge i+1 \notin \tau$. Como $\tau \in T_i$, por hipótesis de inducción se verifica que:

(a) $\forall k\ (1 \leq k \leq i \wedge k \in \tau \rightarrow (q + B_k) \subseteq \tau)$

(b) $\forall s\ (1 \leq s \leq p \wedge (q+s) \in \tau \rightarrow \exists k \in \tau\ (1 \leq k \leq i \wedge s \in B_k))$

Teniendo presente que $i+1 \notin \tau$ resulta que:

(a') $\forall k\ (1 \leq k \leq i+1 \wedge k \in \tau \rightarrow (q + B_k) \subseteq \tau)$.

Por otra parte, de (b) resulta directamente que:

(b') $\forall s\ (1 \leq s \leq p \wedge (q+s) \in \tau \rightarrow \exists k \in \tau\ (1 \leq k \leq i+1 \wedge s \in B_k))$

En consecuencia, la fórmula $\theta(i+1)$ sería verdadera en este caso.

- **Caso 2**: Supongamos que $\tau \in T^*_{i+1,r_{i+1}}$.

  En este caso, del corolario 4.14 se deduce que existe $\rho \in T_i$ tal que $i+1 \in \rho \wedge STEP(\rho, i+1) = \tau$. Por tanto, $\tau = \rho \cup (q + B_{i+1})$ [e3].

  1. Sea k tal que $1 \leq k \leq i+1 \wedge k \in \tau$.
     - Si $1 \leq k \leq i$, entonces teniendo presente que $k \in \rho$, de la hipótesis de inducción resulta que $(q + B_k) \subseteq \rho$. Luego de [e3] se deduce que $(q + B_k) \subseteq \tau$.
     - Si $k = i+1$, de [e3] se deduce directamente que $(q + B_k) \subseteq \tau$.
  2. Sea s tal que $1 \leq s \leq p \wedge (q+s) \in \tau$.
     - O bien $(q+s) \in \rho$, en cuyo caso por hipótesis de inducción $\exists k \in \rho\ (1 \leq k \leq i \wedge s \in B_k)$. Ahora bien, $k \in \rho \rightarrow k \in \tau$. Luego, $\exists k \in \tau\ (1 \leq k \leq i+1 \wedge s \in B_k)$.
     - O bien $(q+s) \notin \rho$, en cuyo caso de [e3] se deduce que $s \in B_{i+1}$.

  En consecuencia, la fórmula $\theta(i+1)$ es verdadera.

□

El siguiente corolario nos proporciona la corrección del programa diseñado.

Corolario 4.16 *(Corrección) Toda molécula, τ, del tubo de salida, T_q, es una molécula consistente. Es decir,* $\forall \tau \in T_q\ (A_\tau = \bigcup \mathcal{F}_\tau)$.

Demostración:

Teniendo presente que la fórmula $\theta(q)$ es verdadera, para cada $\tau \in T_q$ se verifica:

(1) $\forall k\ (1 \leq k \leq q \wedge k \in \tau \rightarrow (q + B_k) \subseteq \tau)$.

(2) $\forall s\ (1 \leq s \leq p \wedge (q+s) \in \tau \rightarrow \exists k \in \tau\ (1 \leq k \leq q \wedge s \in B_k))$.

Veamos que $\bigcup \mathcal{F}_\tau \subseteq A_\tau$.

En efecto: sea $s \in \bigcup \mathcal{F}_\tau$. Entonces $1 \leq s \leq p$ y existe $k \in \tau$ tal que $1 \leq k \leq q \wedge s \in B_k$. De (1) resulta que $(q + B_k) \subseteq \tau$. Luego $(q + s) \in \tau$. Por tanto: $s \in A_\tau$.

Veamos que $A_\tau \subseteq \bigcup \mathcal{F}_\tau$.

En efecto: sea $s \in A_\tau$. Entonces $1 \leq s \leq p$ y $(q + s) \in \tau$. De (2) resulta que $\exists k \in \tau\ (1 \leq k \leq q \wedge s \in B_k)$. Luego $s \in \bigcup \mathcal{F}_\tau$.

De donde podemos concluir que $\bigcup \mathcal{F}_\tau = A_\tau$. Es decir, la molécula τ es consistente.

□

Completitud del programa

La completitud del programa molecular diseñado para resolver este problema se expresa en los siguientes términos: para cada subfamilia, $\mathcal{F}'$, de $\mathcal{F}$ existe una molécula en el tubo de salida que codifica el par ordenado $(\mathcal{F}', \bigcup \mathcal{F}')$. Es decir, para establecer la completitud del programa hay que demostrar que toda molécula, σ, del tubo inicial, T_0, evoluciona a lo largo de la ejecución hasta generar una molécula, σ^q, del tubo de salida, T_q, que es consistente.

Para probar la completitud del programa, consideremos la siguiente fórmula:

$$\delta(i) \equiv \forall \sigma \in T_0\ \ [\forall k\ (1 \leq k \leq i \wedge k \in \sigma^i \rightarrow (q + B_k) \subseteq \sigma^i) \wedge \\ \wedge \forall s\ (1 \leq s \leq p \wedge (q + s) \in \sigma^i \rightarrow \exists k \in \sigma^i\ (1 \leq k \leq i \wedge s \in B_k))]$$

Teorema 4.17 *La fórmula $\delta(i)$ es un invariante del bucle principal. Es decir,*

$$\forall i\ (1 \leq i \leq q \rightarrow \delta(i))$$

Demostración:
Sea i tal que $1 \leq i \leq q$. Sea $\sigma \in T_0$. Como consecuencia del lema 4.10 resulta que $\sigma^i \in T_i$. Como $\theta(i)$ es verdadera, se tiene que:

1. $\forall k\ (1 \leq k \leq i \wedge k \in \sigma^i \rightarrow (q + B_k) \subseteq \sigma^i)$.
2. $\forall s\ (1 \leq s \leq p \wedge (q + s) \in \sigma^i \rightarrow \exists k \in \sigma^i\ (1 \leq k \leq i \wedge s \in B_k))$.

Luego se verifica la fórmula $\delta(i)$.

□

Como consecuencia de este resultado, se obtiene la completitud del programa diseñado para resolver el problema del rellenado.

Corolario 4.18 *(Completitud) Toda molécula, σ, del tubo de ensayo inicial, proporciona al final de la ejecución, una molécula $\sigma^q \in T_q$ que es consistente.*

Demostración:
Sea $\sigma \in T_0$. Como la fórmula $\delta(q)$ es verdadera, se verifica:

1. $\forall k\ (1 \leq k \leq q \wedge k \in \sigma^q \rightarrow (q + B_k) \subseteq \sigma^q)$.

2. $\forall s\ (1 \leq s \leq p \wedge (q+s) \in \sigma^q \rightarrow \exists k \in \sigma^q\ (1 \leq k \leq q \wedge s \in B_k))$.

Razonando como hemos hecho en el corolario 4.16, usando la molécula σ^q en lugar de τ, se obtiene que $\bigcup \mathcal{F}_{\sigma^q} = A_{\sigma^q}$. Es decir, la molécula σ^q es consistente.

□

4.2.2. El problema del recubrimiento

Problema: *Sea $A = \{1, \ldots, p\}$. Sea $\mathcal{F} = \{B_1, \ldots, B_q\}$ una familia finita de subconjuntos de A. Dada la colección $\{(\mathcal{F}', \bigcup \mathcal{F}') : \mathcal{F}'$ subfamilia de $\mathcal{F}\}$, determinar todos los elementos de esta colección que constituyen un recubrimiento de A.*

Vamos a diseñar un programa molecular en el modelo sticker que resuelve este problema. Para ello, supondremos que el tubo de ensayo inicial del programa contendrá moléculas consistentes, con respecto a las subfamilias de $\mathcal{F}$, de acuerdo con el tubo de salida que se obtenía tras la ejecución del programa que resuelve el problema del rellenado estudiado anteriormente.

Un programa molecular que resuelve este problema en el modelo sticker es el siguiente:

```
Procedimiento Selección_Recubrimientos
Entrada: T_0 (una (p+q, q)-librería)
        Rellenado(T_0)
        para s = 1 hasta p hacer
            T_0 ← +(T_0, q+s)
```

El número de operaciones moleculares que realiza este programa es del orden de $O(qp)$. El número de tubos usados es del orden $O(q)$ y el volumen molecular del tubo inicial es 2^q.

A efectos de establecer la verificación formal del programa procedemos a un re-etiquetado simple de tubos.

```
Procedimiento Selección_Recubrimientos
Entrada: T_0 (una (p+q,q)-librería)
    Rellenado(T_0)
    para s = 1 hasta p hacer
        T_s ← +(T_{s-1}, q+s)
```

Corrección del programa

Hemos de probar que toda molécula del tubo de salida codifica un recubrimiento del conjunto A.

Para ello, consideremos la siguiente fórmula:

$$\theta(s) \equiv \forall \tau \in T_s \ ((q + A^s) \subseteq \tau)$$

En donde $A^s = \{1, \ldots, s\}$.

La fórmula $\theta(s)$ expresa que toda molécula, τ, del tubo T_s verifica que $A^s \subseteq A_\tau$ (en donde A_τ es el subconjunto de A asociado a τ, tal como se definió en el problema del rellenado).

Teorema 4.19 *La fórmula $\theta(s)$ es un invariante del bucle principal. Es decir,*

$$\forall s \ (1 \leq s \leq p \rightarrow \theta(s))$$

Demostración:
Por inducción sobre s:

Para probar el caso base , $s = 1$, sea $\tau \in T_1 = +(T_0, q+1)$. Entonces $(q+1) \in \tau$, o lo que es lo mismo, $(q + A^1) \subseteq \tau$.

Sea $s < p$ tal que $s \geq 1$ y supongamos cierto el resultado para s. Sea $\tau \in T_{s+1} = +(T_s, q + s + 1)$. Entonces $\tau \in T_s$ y $(q + s + 1) \in \tau$. Como $\tau \in T_s$, por hipótesis de inducción resulta que $(q + A^s) \subseteq \tau$. Luego $(q + A^{s+1}) \subseteq \tau$.

□

Corolario 4.20 *(Corrección) Toda molécula del tubo de salida codifica un recubrimiento del conjunto A. Es decir, $\forall \tau \in T_p \ ((q + A) \subseteq \tau)$.*

Demostración:
Basta tener presente que la fórmula $\theta(p)$ es verdadera y que $A^p = A$.

□

Completitud del programa

Hemos de probar que toda molécula del tubo inicial que codifique un recubrimiento de A, debe pertenecer al tubo de salida. Para ello, consideremos la fórmula:

$$\delta(s) \equiv \forall \sigma \in T_0 \ ((q + A) \subseteq \sigma \rightarrow \sigma \in T_s)$$

La fórmula $\delta(s)$ expresa que toda molécula del tubo inicial que codifica un recubrimiento de A, pertenece al tubo T_s.

Teorema 4.21 *La fórmula $\delta(s)$ es un invariante del bucle principal. Es decir,*

$$\forall s \ (1 \leq s \leq p \rightarrow \delta(s))$$

Demostración:
Por inducción sobre s:

Para probar el caso base, $s = 1$, sea $\sigma \in T_0$ tal que $(q + A) \subseteq \sigma$. Entonces $\sigma \in T_0 \wedge (q + 1) \in \sigma$. Luego, $\sigma \in +(T_0, q + 1) = T_1$.

Sea $s < p$ tal que $s \geq 1$, y supongamos cierto el resultado para s. Sea $\sigma \in T_0$ tal que $(q + A) \subseteq \sigma$. Por h.i. se tiene que $\sigma \in T_s$. Como $1 \leq s + 1 \leq p$, resulta que $(q + s + 1) \in \sigma$. Luego, $\sigma \in +(T_s, q + s + 1)$. Es decir, $\sigma \in T_{s+1}$.

□

Corolario 4.22 *Toda molécula del tubo inicial que codifique un recubrimiento de A, pertenece al tubo de salida. Es decir,*

$$\forall \sigma \in T_0 \ ((q + A) \subseteq \sigma \rightarrow \sigma \in T_p)$$

Demostración:
Basta tener presente que la fórmula $\delta(p)$ es verdadera.

□

4.2.3. El problema de la ordenación según la cardinalidad

Problema: *Sean* $A = \{1, ..., p\}$, $B = \{b_1, ..., b_s\} \subseteq A$, *y* $\mathcal{F} = \{D_1, ..., D_t\} \subseteq \mathcal{P}(A)$. *Ordenar los conjuntos de* $\mathcal{F}$ *de acuerdo con su cardinal relativo a* B *(esto es, de acuerdo con el número de elementos de* $B \cap D_i$*).*

A continuación vamos a diseñar un programa molecular en el modelo sticker que resuelve el problema propuesto. La idea del programa es la siguiente:

- El tubo de entrada, T_0, contendrá complejos de memoria, σ, codificando cada conjunto de la familia $\mathcal{F}$.

- El programa consistirá en un bucle principal **para** con s pasos. En el paso i-ésimo, se generan $i+1$ tubos, $T_0, T_1, \ldots, T_i$, verificando la siguiente condición: $\forall\sigma\ (\sigma \in T_j \longrightarrow |\sigma \cap \{b_1, \ldots, b_i\}| = j)$. Con el fin de conseguir que se verifique esta propiedad, diseñamos el cuerpo del bucle por inducción. Una vez generados los tubos correspondientes el paso i-ésimo, $T_0, T_1, \ldots, T_i$, los tubos del próximo paso $T_0, T_1, \ldots, T_i, T_{i+1}$ se generan de la siguiente forma:

$$\begin{cases} T_0 = -(T_0, b_{i+1}) \\ T_j = +(T_{j-1}, b_{i+1}) \cup -(T_j, b_{i+1}) & (1 \leqslant j \leqslant i) \\ T_{i+1} = +(T_i, b_{i+1}) \end{cases}$$

La figura 4.2 ilustra gráficamente la idea antes descrita para construir los tubos en el caso $s = 3$.

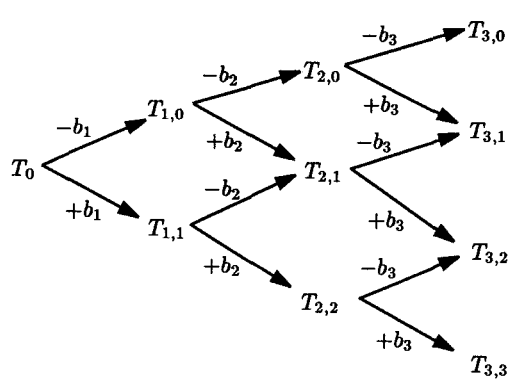


Figura 4.2. Diagrama de construcción para $s = 3$

Estas ideas sugieren el diseño del siguiente programa molecular en el modelo sticker:

```
Procedimiento Orden_Cardinal
Entrada: (T_0, B)
      para i = 1 hasta s hacer
           (T_0, T'_1) ← separación (T_0, b_i)
           para j = 0 hasta i − 1 hacer
                (T''_j, T'_{j+1}) ← separación (T_j, b_i)
                T_j ← T'_j ∪ T''_j
           T_i ← T'_i
Salida: T_0, ..., T_s
```

El procedimiento descrito devuelve $s+1$ tubos que notaremos por:

$$\texttt{Orden\_Cardinal}(T_0, B)[j],\ (0 \leq j \leq s)$$

Se verifica que $|\texttt{Orden\_Cardinal}(T_0, B)[j]| = j$, para cada j tal que $0 \leq j \leq s$.

Este programa molecular usa $2s$ tubos y el número de operaciones moleculares que realiza es $s \cdot (s+3)/2$. El volumen molecular del tubo inicial es t.

El programa presentado para resolver el problema de ordenación según la cardinalidad puede ser adaptado de manera natural a un modelo molecular sin memoria (restringido, débil, etc.), debido a que no utiliza operaciones que alteren la estructura interna de las moléculas.

Verificación del programa molecular diseñado

Para establecer la verificación formal del programa diseñado procedemos a reetiquetar los tubos que se obtienen a lo largo de la ejecución del programa, con el fin de poder realizar un seguimiento individualizado de los mismos.

```
Procedimiento Orden_Cardinal
Entrada: (T_0, B)
      T_{0,0} ← T_0;  T_{0,−1} ← ∅;  T_{0,1} ← ∅
      para i = 1 hasta s hacer
           T_{i,−1} ← ∅;  T_{i,i+1} ← ∅
           para j = 0 hasta i hacer
                T_{i,j} ← +(T_{i−1,j−1}, b_i) ∪ −(T_{i−1,j}, b_i)
Salida: T_{s,0}, ..., T_{s,s}
```

Por cuestiones técnicas, se supondrá que $T_{0,-1} = T_{0,1} = \emptyset$, se notará el conjunto $B_j = \{b_1, ..., b_j\}$ y, por definición, convendremos que $B_0 = \emptyset$.

Proposición 4.23 $\forall i\ (1 \leq i \leq s \rightarrow \forall j \leq i\ \forall \sigma\ (\sigma \in T_{i,j} \rightarrow |\sigma \cap B_i| = j))$.

Demostración:
Por inducción en i.

Para probar el caso base, $i = 1$, hemos de ver que

$$\forall j \leq 1\ \forall \sigma(\sigma \in T_{1,j} \rightarrow |\sigma \cap B_1| = j)$$

- Sea $\sigma \in T_{1,0} = +(T_{0,-1}, b_1) \cup -(T_{0,0}, b_1)$. Como $T_{0,-1} = \emptyset$ y $T_{0,0} = T_0$, resulta que $\sigma \in -(T_0, b_1)$; es decir, $b_1 \notin \sigma$. Por tanto, $|\sigma \cap B_1| = 0$.
- Sea $\sigma \in T_{1,1} = +(T_{0,0}, b_1) \cup -(T_{0,1}, b_1)$. Como $T_{0,1} = \emptyset$, resulta que $\sigma \in T_{0,0} = T_0$ y $b_1 \in \sigma$. Por tanto, $|\sigma \cap B_1| = 1$.

Sea i tal que $1 \leq i < s$ y supongamos que $\forall j \leq i\ \forall \sigma \in T_{i,j}\ (|\sigma \cap B_i| = j)$. Veamos que el resultado se verifica para $i + 1$. Para ello, se procederá por inducción en j, probándose que

$$\forall j \leq i + 1\ \forall \sigma \in T_{i+1,j}\ (|\sigma \cap B_{i+1}| = j)$$

- Para probar el caso base, $j = 0$, sea $\sigma \in T_{i+1,0} = +(T_{i,-1}, b_{i+1}) \cup -(T_{i,0}, b_{i+1})$. Como $T_{i,-1} = \emptyset$, resulta que $\sigma \in T_{i,0}$ y $b_{i+1} \notin \sigma$. De la hipótesis de inducción se deduce que $|\sigma \cap B_i| = 0$. Por tanto, $|\sigma \cap B_{i+1}| = 0$, ya que $b_{i+1} \notin \sigma$.
- Sea $j > 0$ y $\sigma \in T_{i+1,j} = +(T_{i,j-1}, b_{i+1}) \cup -(T_{i,j}, b_{i+1})$. Se verifica:
  - Si $\sigma \in T_{i,j-1}$ y $b_{i+1} \in \sigma$, entonces de la hipótesis de inducción se deduce que $|\sigma \cap B_i| = j - 1$. Como $b_{i+1} \in \sigma$, se puede concluir que $|\sigma \cap B_{i+1}| = j - 1 + 1 = j$.
  - Si $\sigma \in T_{i,j}$ y $b_{i+1} \notin \sigma$, entonces de la hipótesis de inducción se deduce que $|\sigma \cap B_i| = j$. Como $b_{i+1} \notin \sigma$, se concluye que $|\sigma \cap B_{i+1}| = j$.

□

Proposición 4.24 $\forall \sigma \in T_0\ \forall i\ (0 \leq i \leq s \rightarrow \sigma \in T_{i,|\sigma \cap B_i|})$.

Demostración:
Por inducción débil sobre i.

Para el caso base, $i = 0$, el resultado es trivial.

Sea i tal que $0 \leq i < s$ y supongamos cierto el resultado para i $(0 \leq i < s)$. Probemos que se verifica para $i + 1$.

- Si $b_{i+1} \in \sigma$, entonces se tiene que $|\sigma \cap B_{i+1}| = 1 + |\sigma \cap B_i|$. De la hipótesis de inducción se deduce que $\sigma \in T_{i,|\sigma \cap B_i|}$. Por tanto,
$$\sigma \in +(T_{i,|\sigma \cap B_i|}, b_{i+1}) \subseteq T_{i+1,|\sigma \cap B_i|+1}$$
- Si $b_{i+1} \notin \sigma$, entonces $|\sigma \cap B_{i+1}| = |\sigma \cap B_i|$. De la hipótesis de inducción se deduce que $\sigma \in T_{i,|\sigma \cap B_i|}$. Por tanto,
$$\sigma \in -(T_{i,|\sigma \cap B_i|}, b_{i+1}) \subseteq T_{i+1,|\sigma \cap B_i|} = T_{i+1,|\sigma \cap B_{i+1}|}$$

□

De las proposiciones 4.23 y 4.24 se concluye, respectivamente, la corrección (toda molécula de un tubo de salida proporciona una solución correcta asociada a dicho tubo) y la completitud (toda molécula del tubo de entrada aparece en el correspondiente tubo de salida, de acuerdo con su cardinalidad) del programa diseñado.

Corolario 4.25 *(Corrección)* $\forall j \, \forall \sigma \, (0 \leq j \leq s \wedge \sigma \in T_{s,j} \rightarrow |\sigma \cap B| = j)$.

Corolario 4.26 *(Completitud) Si* $\sigma \in T_0$ *y* $|\sigma \cap B| = j$, *entonces* $\sigma \in T_{s,j}$.

Como casos particulares que pueden resultar de interés, y que usaremos en el diseño de otros programas moleculares, se consideran los procedimientos siguientes:

- `Orden_Cardinal`(T_0), en el caso $B = A$.
- `Orden_Cardinal`(T_0, l, k), en el caso $B = \{l, l+1, ..., k\}$.

4.2.4. El problema de las familias disjuntas

Problema: *Sea* $A = \{1, \ldots, p\}$. *Sea* $\mathcal{F} = \{B_1, \ldots, B_q\}$ *una familia finita de subconjuntos de* A. *Determinar todas las subfamilias de* $\mathcal{F}$ *cuyos elementos son disjuntos dos a dos.*

Diseño de un programa molecular

Al igual que en el problema del rellenado, para cada j tal que $1 \leq j \leq q$ notaremos $r_j = |B_j|$ y $B_j = \{x_j^1, \ldots, x_j^{r_j}\}$.

Vamos a diseñar un programa molecular que resuelve este problema en el modelo sticker. La idea es la siguiente: a partir de un tubo de ensayo inicial, T_0, cuyas moléculas codifican todas las posibles subfamilias de $\mathcal{F}$ (para ello, basta que T_0 sea una $(p+q, q)$-librería), se procede como sigue:

- Se clasifican las moléculas de T_0 en dos tubos: T^+, formado por las moléculas que contienen al 1, y T^-, formado por las moléculas que no contienen al 1.
  - De entre las moléculas de T^+
    - Se desechan aquellas que para algún $x_1^j \in B_1$ tiene activada la posición $q + x_1^j$.
    - Si tienen desactivadas todas las posiciones $q + x_1^j$ $(1 \leq j \leq r_1)$, entonces se activan todas ellas (formando un tubo $T_{r_1}^*$).
  - Se considera $T_0 = T_{r_1}^* \cup T^-$.
- Se reitera el proceso para 2 (con relación al conjunto B_2), 3 (con relación al conjunto B_3),..., hasta q (con relación al conjunto B_q).

Estas ideas sugieren el diseño del siguiente programa molecular en el modelo sticker:

```
Procedimiento Disjuntos
Entrada: T_0 (una (p+q,q)-librería)
    para i = 1 hasta q hacer
        (T^+, T^-) ← separación(T_0, i)
        T_0^* ← T^+
        para j = 1 hasta r_i hacer
            (T^basura, T) ← separación(T_{j-1}^*, q + x_i^j)
            T_j^* ← activar(T, q + x_i^j)
        T_0 ← mezcla(T_{r_i}^*, T^-)
```

El número de operaciones moleculares que realiza este programa es del orden $O(q \cdot p)$. El número de tubos usado es del orden de $O(p)$ y el volumen molecular del tubo inicial es 2^q.

Verificación del programa molecular diseñado

A efectos de establecer la verificación formal de este programa procedemos a un re-etiquetado de tubos.

```
Procedimiento Disjuntos
Entrada: T_0
    para i = 1 hasta q hacer
        (T_i^+, T_i^-) ← separación(T_{i-1}, i)
        T*_{i,0} ← T_i^+
        para j = 1 hasta r_i hacer
            (T_{i,j}^{basura}, T•_{i,j}) ← separación(T*_{i,j-1}, q + x_i^j)
            T*_{i,j} ← activar(T•_{i,j}, q + x_i^j)
        T_i ← mezcla(T*_{i,r_i}, T_i^-)
```

Seguidamente tratamos de adaptar a este programa la función $STEP$ que se ha estudiado en el problema del rellenado, de tal manera que capture la evolución de una molécula tras ejecutar un paso del bucle principal.

Téngase presente que en este programa hay moléculas (las del tubo $T_{i,j}^{basura}$) que se "pierden" tras la ejecución del paso correspondiente del bucle principal. Por ello, a diferencia del programa diseñado para resolver el problema del rellenado, la función $STEP$ asociada a este programa no será necesariamente total.

Definición 4.27 *Sean i tal que $1 \leq i \leq q$ y $\sigma \in T_{i-1}$. Se define:*

$$STEP(\sigma, i) = \begin{cases} \sigma & , si\ i \notin \sigma \\ \sigma \cup (q + B_i) & , si\ i \in \sigma \wedge \sigma \cap (q + B_i) = \emptyset \\ \uparrow & , e.c.o.c. \end{cases}$$

En cualquiera de los dos primeros casos, escribiremos $STEP(\sigma, i)\downarrow$.

Es decir, si $\sigma \in T_{i-1} \wedge i \in \sigma \wedge \forall j\ (1 \leq j \leq r_i \rightarrow q + x_i^j \notin \sigma)$, entonces la molécula $STEP(\sigma, i) = \tau$ coincide con σ excepto en los siguientes valores

$$\forall j\ (1 \leq j \leq r_i \rightarrow \sigma(q + x_i^j) = 0 \wedge \tau(q + x_i^j) = 1)$$

Téngase presente que si $STEP(\sigma, i)\downarrow$, entonces $\sigma \subseteq STEP(\sigma, i)$.

Definición 4.28 *Para cada i tal que $1 \leq i \leq p$, se define la función $STEP_i$ como sigue: $STEP_i(\sigma) = STEP(\sigma, i)$, para cada $\sigma \in T_{i-1}$.*

De la definición anterior se tiene que $dom(STEP_i) \subseteq T_{i-1}$. Veamos que $STEP_i$ es una función parcial de T_{i-1} en T_i; es decir, que $rang(STEP_i) \subseteq T_i$.

Lema 4.29 $\forall i\ (1 \leq i \leq q \rightarrow \forall \sigma \in T_{i-1}\ (STEP(\sigma, i)\downarrow \rightarrow STEP(\sigma, i) \in T_i))$.

Demostración:
Sea i tal que $1 \leq i \leq q$ y $\sigma \in T_{i-1}$ tal que $STEP(\sigma, i)\downarrow$.

- **Caso 1**: Supongamos que $i \notin \sigma$.

  Como $\sigma \in T_{i-1}$ e $i \notin \sigma$, se obtiene por una parte que $\sigma \in -(T_{i-1}, i) = T_i^- \subseteq T_i$ y, por otra, de la definición de $STEP$, se sigue que $STEP(\sigma, i) = \sigma$. Por tanto $STEP(\sigma, i) \in T_i$.

- **Caso 2**: Supongamos que $i \in \sigma \wedge \sigma \cap (q + B_i) = \emptyset$. En este caso definimos recursivamente $\sigma_{(j)}$, para $0 \leq j \leq r_i$, como sigue:

$$\begin{cases} \sigma_{(0)} = \sigma \\ \sigma_{(j+1)} = \sigma_{(j)} \cup \{q + x_i^{j+1}\} \end{cases}$$

  Veamos que $\forall j\ (0 \leq j \leq r_i \rightarrow \sigma_{(j)} \in T_{i,j}^*)$.

  Para probar el caso base, $j = 0$, basta tener presente que $\sigma_{(0)} = \sigma \in +(T_{i-1}, i) = T_i^+ = T_{i,0}^*$.

  Sea $j < r_i$ tal que $\sigma_{(j)} \in T_{i,j}^*$. De la definición de $\sigma_{(j)}$ resulta que $\forall t\ (j < t \leq r_i \rightarrow (t \in \sigma_{(j)} \leftrightarrow t \in \sigma))$. Como $q + x_i^{j+1} \notin \sigma$ se deduce que $q + x_i^{j+1} \notin \sigma_{(j)}$. Luego, $\sigma_{(j)} \in -(T_{i,j}^*, q + x_i^{j+1}) = T_{i,j+1}^\bullet$. De la definición de $\sigma_{(j+1)}$ resulta que $\sigma_{(j+1)} \in \texttt{activar}(T_{i,j+1}^\bullet, q + x_i^{j+1}) = T_{i,j+1}^*$.

  Teniendo presente que de las definiciones anteriores $\sigma_{(r_i)} = STEP(\sigma, i)$, se concluye que $STEP(\sigma, i) \in T_{i,r_i}^* \subseteq T_i$.

□

El lema anterior nos permite definir la historia de una molécula, σ, del tubo de ensayo inicial, T_0.

Definición 4.30 *Para cada $\sigma \in T_0$ se define:*

$$\begin{cases} \sigma^0 = \sigma \\ \sigma^{i+1} = STEP(\sigma^i, i+1) \quad , \text{para } 0 \leq i < q \end{cases}$$

Si $\sigma \in T_0$ es tal que $\sigma^i \downarrow$ para un cierto i $(1 \leq i \leq q)$, entonces diremos que la molécula σ ha *sobrevivido* tras la ejecución del paso i-ésimo del bucle principal.

Lema 4.31 $\forall i\ (0 \leq i \leq q \rightarrow \forall \sigma \in T_0\ (\sigma^i \downarrow \rightarrow \sigma^i \in T_i))$.

Demostración:
Por inducción débil sobre i.

El caso base, $i = 0$, es trivial, ya que para cada $\sigma \in T_0$ se tiene que $\sigma^0 = \sigma \in T_0$.

Sea $i < q$ tal que el resultado es válido para i. Sea $\sigma \in T_0$ tal que $\sigma^{i+1} \downarrow$. Entonces $STEP(\sigma^i, i+1) \downarrow$. Luego $\sigma^i \downarrow$. Por tanto, de la hipótesis de inducción resulta que $\sigma^i \in T_i$. Como $\sigma^i \in T_i \wedge STEP(\sigma^i, i+1) \downarrow$, del lema 4.29 resulta que $STEP(\sigma^i, i+1) \in T_{i+1}$. Luego, $\sigma^{i+1} \in T_{i+1}$.

□

Seguidamente vamos a describir las moléculas del tubo $T^*_{i,j}$ a través de moléculas del tubo T^+_i.

Lema 4.32 *Para cada i tal que $1 \leq i \leq q$ y cada j tal que $1 \leq j \leq r_i$, se tiene que*

$$\forall \tau \in T^*_{i,j}\ \exists \sigma \in T^+_i\ (\tau = \sigma \cup (q + B^j_i) \wedge \sigma \cap (q + B^j_i) = \emptyset))$$

En donde $B^j_i = \{x^1_i, \ldots, x^j_i\}$.

Demostración:
Sea i tal que $1 \leq i \leq q$. Probemos por inducción débil sobre j que

$$\forall j\ (1 \leq j \leq r_i \rightarrow \forall \tau \in T^*_{i,j}\ \exists \sigma \in T^+_i\ (\tau = \sigma \cup (q + B^j_i) \wedge \sigma \cap (q + B^j_i) = \emptyset))$$

Para probar el caso base, $j = 1$, sea $\tau \in T^*_{i,1} = \texttt{activar}(T^\bullet_{i,1}, q + x^1_i)$. Entonces existe $\sigma \in T^\bullet_{i,1}$ tal que $(\tau = \sigma \cup \{q + x^1_i\} = \sigma \cup B^1_i)$. Ahora bien, $\sigma \in T^\bullet_{i,1} = -(T^*_{i,0}, q + x^1_i)$. Luego, $\sigma \in T^*_{i,0} = T^+_i$ y, además, $(q + x^1_i) \notin \sigma$. Es decir, $\sigma \cap (q + B^1_i) = \emptyset$.

Sea $j < r_i$ tal que $j \geq 1$ y supongamos cierto el resultado para j. Veamos que el resultado es cierto para $j+1$. Para ello, sea $\tau \in T^*_{i,j+1} = \texttt{activar}(T^\bullet_{i,j+1}, q + x^{j+1}_i)$. Existe $\sigma' \in T^\bullet_{i,j+1}$ tal que $\tau = \sigma' \cup \{q + x^{j+1}_i\}$. Ahora bien, $\sigma' \in T^\bullet_{i,j+1} = -(T^*_{i,j}, q + x^{j+1}_i)$. Luego, $\sigma' \in T^*_{i,j}$ y, además, $(q + x^{j+1}_i) \notin \sigma'$. Como $\sigma' \in T^*_{i,j}$, por hipótesis de inducción existe $\sigma \in T^+_i$ tal que $\sigma' = \sigma \cup (q + B^j_i) \wedge \sigma \cap (q + B^j_i) = \emptyset$. Por tanto, $\tau = \sigma \cup (q + B^{j+1}_i) \wedge \sigma \cap (q + B^{j+1}_i) = \emptyset$.

□

A continuación veamos que las moléculas, τ, de T^*_{i,r_i} están en el rango de la función $STEP_i$. Más aún, son imágenes por $STEP_i$ de alguna molécula $\sigma \in T_{i-1}$ tal que $\sigma \neq \tau$.

Corolario 4.33 *Para cada i tal que $1 \leq i \leq q$, se tiene que*

$$\forall \tau \in T^*_{i,r_i}\ \exists \sigma \in T_{i-1}\ (i \in \sigma \wedge \rho \neq \tau \wedge STEP(\sigma, i) = \tau))$$

Demostración:
Sea i tal que $1 \leq i \leq q$ y $\tau \in T^*_{i,r_i}$.

Del lema 4.32 aplicado a $j = r_i$ se deduce que existe $\sigma \in T_i^+$ tal que $\tau = \sigma \cup (q + B_i) \wedge \sigma \cap (q + B_i) = \emptyset$. Luego $\sigma \neq \tau$.

Además, $\sigma \in T_i^+ = +(T_{i-1}, i)$. Por tanto, $\sigma \in T_{i-1}$ e $i \in \sigma$. Teniendo presente que $\sigma \cap (q + B_i) = \emptyset$, se deduce que $STEP(\sigma, i) = \sigma \cup (q + B_i) = \tau$.

□

Veamos que los tubos de ensayo $T^*_{i,j}$ y T_i^- no tienen moléculas en común.

Corolario 4.34 *Para cada i tal que $1 \leq i \leq q$ y cada j tal que $1 \leq j \leq r_i$, se tiene que $T^*_{i,j} \cap T_i^- = \emptyset$.*

Demostración:
Sean i, j tales que $1 \leq i \leq q \wedge 1 \leq j \leq r_i$. Sea $\tau \in T^*_{i,j}$. Del lema 4.32 se deduce que existe $\sigma \in T_i^+$ tal que $\tau = \sigma \cup (q + B_i^j) \wedge \sigma \cap (q + B_i^j) = \emptyset$. Como $\sigma \in T_i^+ = +(T_{i-1}, i)$, resulta que $\sigma \in T_{i-1}$ e $i \in \sigma$. Como $\sigma_{[1,q]} = \tau_{[1,q]} \wedge i \in \sigma \wedge 1 \leq i \leq q$, se deduce que $i \in \tau$. Por tanto, $\tau \notin -(T_{i,1}, i)$. Es decir, $\tau \notin T_i^-$.

□

Veamos que la función $STEP_i$ restringida a T_i^- es la aplicación identidad.

Lema 4.35 *Para cada i tal que $1 \leq i \leq q$, se tiene que*

$$\forall \tau \in T_i^-\ (\tau \in T_{i-1} \wedge STEP(\tau, i) = \tau))$$

Demostración:
Sea i tal que $1 \leq i \leq q$, y $\tau \in T_i^-$. Como $T_i^- = -(T_{i-1}, i)$, resulta que $\tau \in T_{i-1} \wedge i \notin \tau$. De la definición de $STEP$ se deduce que $STEP(\tau, i) = \tau$.

□

A continuación vamos a probar que la función $STEP_i$ es inyectiva; es decir, que dos moléculas distintas siguen evoluciones distintas por la ejecución de un paso del bucle principal.

Lema 4.36 *Para cada i tal que $1 \leq i \leq q$, se verifica que*

$$\forall \sigma \in T_{i-1} \forall \tau \in T_{i-1}\ (STEP(\sigma, i) \downarrow = STEP(\tau, i) \rightarrow \sigma = \tau))$$

Demostración:
Sea i tal que $1 \leq i \leq q$. Sean $\sigma \in T_{i-1}$, y $\tau \in T_{i-1}$ tales que $STEP(\sigma, i) \downarrow = STEP(\tau, i)$.

- **Caso 1**: Supongamos que $i \notin \sigma$.

  Como $\sigma_{[1,q]} = \tau_{[1,q]}$, resulta que $i \notin \tau$. Luego $STEP(\sigma, i) = \sigma$ y $STEP(\tau, i) = \tau$. De donde se deduce que $\sigma = \tau$.

- **Caso 2**: Supongamos que $i \in \sigma$ y $\sigma \cap (q + B_i) = \emptyset$.

  Como $\sigma_{[1,q]} = \tau_{[1,q]} \wedge i \in \sigma$, resulta que $i \in \tau$. Como $STEP(\sigma, i) \downarrow \Rightarrow STEP(\tau, i) \downarrow$, teniendo presente que $i \in \tau$ resulta que $\tau \cap (q + B_i) = \emptyset$. Sean $\sigma' = STEP(\sigma, i) \wedge \tau' = STEP(\tau, i)$. Entonces:

  $$\begin{cases} \sigma' = \sigma \cup (q + B_i) \wedge \sigma \cap (q + B_i) = \emptyset \\ \tau' = \tau \cup (q + B_i) \wedge \tau \cap (q + B_i) = \emptyset \end{cases}$$

  Como $\sigma' = \tau'$, de la relación anterior se concluye que $\sigma = \tau$.

□

Veamos que la evolución de una molécula $\sigma \in T_0$, tras ejecutar i pasos del bucle principal, identifica a dicha molécula.

Corolario 4.37 *Sean $\sigma, \rho \in T_0$. Sea i tal que $0 \leq i \leq q$. Si $\sigma^i \downarrow = \rho^i$, entonces $\sigma = \rho$.*

Demostración:
Por inducción débil sobre i.

El caso base, $i = 0$, es trivial, ya que $\sigma^0 = \sigma \wedge \rho^0 = \rho$.

Sea $i < q$ tal que el resultado es válido para i. Sean $\sigma \in T_0$, $\rho \in T_0$ tales que $\sigma^{i+1} \downarrow = \rho^{i+1}$. Entonces $STEP(\sigma^i, i+1) \downarrow = STEP(\rho^i, i+1)$. De lema 4.36 se deduce que $\sigma^i \downarrow = \rho^i$, y de la hipótesis de inducción se concluye que $\sigma = \rho$.

□

Seguidamente vamos a justificar la relevancia de los tubos T_i a lo largo de la ejecución, probando que cada molécula de dichos tubos proviene de la evolución de una única molécula del tubo inicial.

Lema 4.38 *Para cada i tal que $0 \leq i \leq q$ y cada molécula $\tau \in T_i$, existe una única molécula del tubo de entrada, $\sigma \in T_0$, verificando que $\sigma^i = \tau$.*

Demostración:
Vamos a probar la existencia por inducción débil sobre i.

El caso base, $i = 0$, es trivial.

Sea $i < q$ tal que se verifica el resultado para i. Sea $\tau \in T_{i+1} = T^*_{i+1,r_{i+1}} \cup T^-_{i+1}$.

- **Caso 1**: Supongamos que $\tau \in T^*_{i+1,r_{i+1}}$.
  Del corolario 4.33 se deduce que existe $\rho \in T_i$ tal que $i+1 \in \wedge\rho \neq \tau \wedge STEP(\rho, i+1) = \tau$. Como $\rho \in T_i$, por hipótesis de inducción existe $\sigma \in T_0$ tal que $\rho = \sigma^i$. Luego, $\tau = STEP(\rho, i+1) = STEP(\sigma^i, i+1) = \sigma^{i+1}$.
- **Caso 2**: Supongamos que $\tau \in T^-_{i+1}$.
  Del lema 4.35 resulta que $\tau \in T_i \wedge STEP(\tau, i+1) = \tau$. Como $\tau \in T_i$, de la hipótesis de inducción se deduce que existe $\sigma \in T_0$ tal que $\tau = \sigma^i$. Luego, $\tau = STEP(\tau, i+1) = STEP(\sigma^i, i+1) = \sigma^{i+1}$.

La unicidad de la molécula $\sigma \in T_0$ verificando que $\sigma = \tau$, se sigue directamente del corolario 4.37.

□

Lema 4.39 *Para cada i tal que $1 \leq i \leq q$ y cada j tal que $1 \leq j \leq r_i$, se verifica que*

$$\forall \tau \in T^*_{i,j}\ (i \in \tau)$$

Demostración:
Sean i, j tales que $1 \leq i \leq q$ y $1 \leq j \leq r_i$. Sea $\tau \in T^*_{i,j}$. Del lema 4.32 se deduce la existencia de una molécula $\rho \in T_i^+$ verificando que $\tau = \rho \cup (q + B_i^j)$ y $\rho \cap (q + B_i^j) = \emptyset$. Como $\rho \in T_i^+ = +(T_{i-1}, i)$, se tiene que $i \in \rho$. Teniendo presente que $\rho_{[1,q]} = \tau_{[1,q]} \wedge 1 \leq i \leq q$, se concluye que $i \in \tau$. □

Proposición 4.40 *Sea $s \in T_0$. Sea i tal que $1 \leq i \leq q$ y $\sigma^i \downarrow$. Para cada k tal que $0 \leq k < i$, se tiene que $\sigma^k_{[1,q]} = \sigma^i_{[1,q]} \wedge \sigma^k_{[q+1,q+p]} \subseteq \sigma^i_{[q+1,q+p]}$.*

Demostración:
Por inducción débil sobre i.

El caso base, $i = 0$, es inmediato, ya que $\sigma^0_{[1,q]} = \sigma^1_{[1,q]}$ y como $\sigma^0_{[q+1,q+p]} = \emptyset$, se deduce que $\sigma^0_{[q+1,q+p]} \subseteq \sigma^1_{[q+1,q+p]}$.

Sea $i < q$ tal que $i \geq 1$ y supongamos cierto el resultado para i. Veamos que el resultado también es cierto para $i+1$. Para ello, sea k tal que $0 \leq k < i+1$ y sea $\sigma \in T_0$ tal que $\sigma^{i+1}\downarrow$.

- **Caso 1**: Supongamos que $0 \leq k < i$.
  Como $\sigma^{i+1}\downarrow$ y $\sigma^{i+1} = STEP(\sigma^i, i+1)$, resulta que $\sigma^i\downarrow$. Como $0 \leq k < i \wedge \sigma \in T_0 \wedge \sigma^i\downarrow$, de la hipótesis de inducción se deduce que $\sigma^k_{[1,q]} = \sigma^i_{[1,q]} \wedge \sigma^k_{[q+1,q+p]} \subseteq \sigma^i_{[q+1,q+p]}$. Teniendo presente que $\sigma^i_{[1,q]} = \sigma^{i+1}_{[1,q]} \wedge \sigma^i_{[q+1,q+p]} \subseteq \sigma^{i+1}_{[q+1,q+p]}$ (ya que $\sigma^i \subseteq STEP(\sigma^i, i+1) = \sigma^{i+1}$), se concluye que $\sigma^k_{[1,q]} = \sigma^{i+1}_{[1,q]} \wedge \sigma^k_{[q+1,q+p]} \subseteq \sigma^{i+1}_{[q+1,q+p]}$.
- **Caso 2**: Supongamos que $k = i$.
  Entonces se tiene que $\sigma^i_{[1,q]} = \sigma^{i+1}_{[1,q]}$ y $\sigma^i_{[q+1,q+p]} \subseteq \sigma^{i+1}_{[q+1,q+p]}$.

□

Para establecer la corrección del programa anterior, consideremos la fórmula

$$\theta(i) \equiv \forall \tau \in T_i \; \forall k, k' \in \tau \; (1 \leq k < k' \leq i \rightarrow B_k \cap B_{k'} = \emptyset)$$

Es decir, la fórmula $\theta(i)$ expresa que toda molécula del tubo T_i codifica una subfamilia de $\mathcal{F}$ tal que los conjuntos de esa familia de índice menor o igual que i son disjuntos dos a dos.

Teorema 4.41 *La fórmula $\theta(i)$ es un invariante del bucle principal. Es decir,*

$$\forall i \; (1 \leq i \leq q \rightarrow \theta(i))$$

Demostración:
Por inducción débil sobre i.

El caso base, $i = 1$, es trivial.

Sea $i < q$ tal que $i \geq 1$ y supongamos que la fórmula $\theta(i)$ es verdadera. Sean $\tau \in T_{i+1}$ y $k, k' \in \tau$ tales que $1 \leq k < k' \leq i+1$. Como $\tau \in T_{i+1} \wedge i+1 \leq q$, por el lema 4.38 existe $\sigma \in T_0$ tal que $\sigma^{i+1} = \tau$.

- **Caso 1**: Supongamos que $k' < i+1$.
  Como $k, k' \in \tau = \sigma^{i+1}$, $1 \leq k < k' \leq q$ y $\sigma^{i+1}_{[1,q]} = \sigma^i_{[1,q]}$, resulta que $k, k' \in \sigma^i$. Como $\sigma^{i+1}\downarrow$ y $\sigma^{i+1} = STEP(\sigma^i, i+1)$, resulta que $\sigma^i\downarrow$. Luego, del lema 4.31, se deduce que $\sigma^i \in T_i$. Así pues $k, k' \in \sigma^i \wedge 1 \leq k < k' \leq i \wedge \sigma^i \in T_i$. Como por hipótesis de inducción la fórmula $\theta(i)$ es verdadera, se concluye que $B_k \cap B_{k'} = \emptyset$.

- **Caso 2**: Supongamos que $k' = i + 1$.

  Hemos que probar que $B_k \cap B_{i+1} = \emptyset$. En primer lugar observemos que $k, k' \in \sigma^i$, ya que $1 \leq k < k' = i+1 \leq q \wedge k, k' \in \tau \wedge \tau = \sigma^{i+1} \wedge \sigma^{i+1}_{[1,q]} = \sigma^i_{[1,q]}$. Del lema 4.31 resulta que $\sigma^i \in T_i$ y, por tanto, $\sigma^i \in +(T_i, k') = +(T_i, i+1) = T^+_{i+1}$.

  Veamos que $B_k \cap B_{i+1} = \emptyset$; es decir que no existe u tal que $1 \leq u \leq r_{i+1}$ y $x^u_{i+1} \in B_k$.

  Supongamos lo contrario, es decir, que existiera u tal que $1 \leq u \leq r_{i+1}$ y $x^u_{i+1} \in B_k$. Entonces, existiría v tal que $1 \leq v \leq r_k \wedge x^v_k = x^u_{i+1}$. En tal situación, tendríamos que $k \in \sigma^i \wedge 1 \leq k \leq q \wedge \sigma^i_{[1,q]} = \sigma^{k-1}_{[1,q]}$ y, por tanto, $k \in \sigma^{k-1}$. Como $\sigma^k = STEP(\sigma^{k-1}, k)$ y $k \in \sigma^{k-1}$, de la definición de $STEP$ se deduce que $\sigma^k = \sigma^{k-1} \cup (q + B_k)$ y $\sigma^{k-1} \cap (q + B_k) = \emptyset$. Como $1 \leq v \leq r_k$, resulta que $(q + x^v_k) \in \sigma^k$. Ahora bien, $k < k' = i+1$, por tanto $k \leq i$. Aplicando la proposición 4.40 tenemos que $\sigma^k_{[q+1,q+p]} \subseteq \sigma^i_{[q+1,q+p]}$ y, por tanto, $(q + x^v_k) \in \sigma^i$, o lo que es lo mismo, $(q + x^u_{i+1}) \in \sigma^i$. Ahora bien, teniendo presente que $\sigma^{i+1} \downarrow$ y que $i + 1 \in \sigma^i$, de la definición de $STEP$ se deduce que $\sigma^i \cap (q + B_{i+1}) = \emptyset$. En particular, $(q + x^u_{i+1}) \notin \sigma^i$. Lo que es una contradicción.

□

Corolario 4.42 *(Corrección) Toda molécula del tubo de salida codifica una subfamilia de $\mathcal{F}$ que está formada por conjuntos disjunto dos a dos.*

Demostración:
Basta tener presente que la fórmula $\theta(q)$ es verdadera. Y, por tanto, se verifica que

$$\forall \tau \in T_q \; \forall k, k' \in \tau \; (1 \leq k < k' \leq q \rightarrow B_k \cap B_{k'} = \emptyset)$$

□

Para establecer la completitud del programa diseñado consideramos la siguiente fórmula:

$$\delta(i) \equiv \forall \sigma \in T_0 \; ((\forall k, k' \in \sigma \; (1 \leq k < k' \leq q \rightarrow B_k \cap B_{k'} = \emptyset)) \rightarrow \sigma^i \in T_i)$$

Es decir, la fórmula $\delta(i)$ expresa que toda molécula del tubo de ensayo inicial que codifique una subfamilia de $\mathcal{F}$ formada por conjuntos disjuntos dos a dos, pertenece al tubo T_i .

Teorema 4.43 *La fórmula $\delta(i)$ es un invariante del bucle principal. Es decir,*

$$\forall i\ (1 \leq i \leq q \rightarrow \delta(i))$$

Demostración:
Por inducción débil sobre i.

Para probar el caso base, $i = 1$, sea $\sigma \in T_0$ tal que $\forall k, k' \in \sigma\ (1 \leq k < k' \leq q \rightarrow B_k \cap B_{k'} = \emptyset)$.

- **Caso 1**: Supongamos que $1 \notin \sigma$.
  En este caso, $\sigma^1 = STEP(\sigma, 1) \downarrow = \sigma$. Luego, del lema 4.31 resulta que $\sigma^1 \in T_1$.
- **Caso 2**: Supongamos que $1 \in \sigma$.
  Como $\sigma \in T_0$ y T_0 es una $(p+q, q)$-librería, resulta que $\sigma_{[q+1,q+p]} = \emptyset$. En particular, $\sigma \cap (q + B_1) = \emptyset$, luego $\sigma^1 \downarrow$. Del lema 4.31 se concluye que $\sigma^1 \in T_1$.

Sea $i < q$ tal que $i \geq 1$ y supongamos cierto el resultado para i. Sea $\sigma \in T_0$ tal que $\forall k, k' \in \sigma\ (1 \leq k < k' \leq q \rightarrow B_k \cap B_{k'} = \emptyset)$. De la hipótesis de inducción se deduce que $\sigma^i \in T_i$.

- **Caso 1**: Supongamos que $i + 1 \notin \sigma^i$.
  En este caso, $STEP(\sigma^i, i+1) = \sigma^i$. Luego $\sigma^{i+1} \downarrow$. Del lema 4.31 se deduce que $\sigma^{i+1} \in T_{i+1}$.
- **Caso 2**: Supongamos que $i + 1 \in \sigma^i$.
  Basta probar que $\sigma^{i+1} \downarrow$. Para ello, por la definición de $STEP$ y teniendo presente que $\sigma^i \downarrow$, bastará ver que $\sigma^i \cap (q + B_{i+1}) = \emptyset$.
  Supongamos lo contrario. Entonces existiría

  $$j_0 = \text{mín}\{j :\ 1 \leq j \leq r_{i+1} \wedge (q + x^j_{i+1}) \in \sigma^i\}$$

  Veamos que entonces debería verificarse que $(q + x^{j_0}_{i+1}) \in \sigma^{i-1}$.

  En efecto: si $(q + x^{j_0}_{i+1}) \notin \sigma^{i-1}$, resultaría que $\sigma^{i-1} \neq \sigma^i$. Teniendo presente que $\sigma^i = STEP(\sigma^{i-1}, i)$, deduciríamos que

  $$i \in \sigma^{i-1} \wedge \sigma^i = \sigma^{i-1} \cup (q + B_i) \wedge \sigma^i \cap (q + B_i) = \emptyset$$

  Luego existiría u tal que $1 \leq u \leq r_i \wedge x^u_i = x^{j_0}_{i+1}$. De donde obtendríamos que $B_i \cap B_{i+1} \neq \emptyset$.

Teniendo presente que $(q + x_{i+1}^{j_0}) \in \sigma^{i-1}$, existiría

$$k_0 = \min\{k : (q + x_{i+1}^{j_0}) \in \sigma^k\}$$

Como T_0 es una $(p+q, q)$-librería, resultaría que $k_0 > 0$ (ya que $\sigma_{[q+1,q+p]} = \emptyset$). Teniendo presente que $\sigma^{k_0} = STEP(\sigma^{k_0-1}, k_0)$, y que $(q + x_{i+1}^{j_0}) \notin \sigma^{k_0-1}$, deduciríamos que $\sigma^{k_0} \neq \sigma^{k_0-1}$. Y, por tanto, $k_0 \in \sigma^{k_0-1} \wedge \sigma^{k_0} = \sigma^{k_0-1} \cup (q + B_{k_0}) \wedge \sigma^{k_0-1} \cap (q + B_{k_0}) = \emptyset$. Luego existiría v tal que $1 \leq v \leq r_{k_0} \wedge x_{i+1}^{j_0} = x_{k_0}^{v}$. De donde resultaría que $B_{i+1} \cap B_{k_0} \neq \emptyset$ (con $k_0 \neq i + 1$). Lo que es una contradicción.

□

Corolario 4.44 *(Completitud) Toda molécula del tubo de ensayo inicial que codifique una subfamilia de $\mathcal{F}$ formada por conjuntos disjuntos dos a dos, sobrevive todos los pasos del bucle principal y pertenece al tubo de salida T_q.*

Demostración:
Teniendo presente que la fórmula $\delta(q)$ es verdadera, resulta que

$$\forall \sigma \in T_0 \ ((\forall k, k' \in \sigma \ (1 \leq k < k' \leq q \rightarrow B_k \cap B_{k'} = \emptyset)) \rightarrow \sigma^q \in T_q)$$

□

4.2.5. Funciones Peso sobre conjuntos

Para terminar esta sección, presentamos un programa molecular que se va a usar como subrutina auxiliar para la resolución de problemas del tipo de la Mochila o el Subset–Sum, relativos al almacenamiento del peso de un conjunto con respecto a una función de valores dada.

Planteamiento del problema y diseño

Problema: *Dada una familia de subconjuntos de un conjunto finito A y una función $f : A \longrightarrow \mathbf{N}$, hallar los pesos, respecto de la función f, de los elementos de la familia.*

Sean $A = \{1, ..., p\}$, $r \in \mathbf{N}$ y $f : A \longrightarrow \mathbf{N}$ una función total. Si $B \subseteq A$, entonces notaremos $f(B) = \sum_{i \in B} f(i)$. Por conveniencia, definimos $f(0) = 0$. Sean $q_f = f(A)$, $A_i = \{0, \ldots, i\}$ (con $0 \leq i \leq p$) y T_0 un tubo de entrada que consta de (n, k, m)-complejos de memoria, con $k \geq p + r + q_f$.

Cada molécula $\sigma \in T_0$ se puede considerar que codifica un subconjunto, $B_\sigma \subseteq A$, caracterizado por la condición $B_\sigma = \sigma_{[1,p]}$. Y recíprocamente, cada subconjunto $B \subseteq A$ puede ser codificado por una molécula $\sigma_B \in T_0$, caracterizada por la siguiente condición: $\sigma_B(i) = 1$ si y sólo si $i \in B$.

En cada molécula del tubo de entrada, $\sigma \in T_0$, consideramos las siguientes *zonas* diferenciadas:

$$(A\sigma) = \sigma_{[1,p]}, \qquad (F\sigma) = \sigma_{[p+r+1,p+r+q_f]}$$
$$(L\sigma) = \sigma_{[p+1,p+r]}, \quad (R\sigma) = \sigma_{[p+r+q_f+1,k]}$$

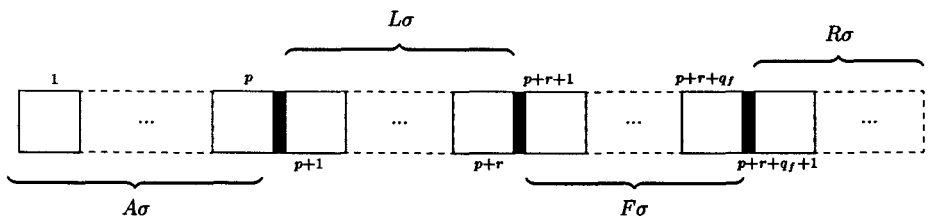


Figura 4.3. Zonas consideradas en los complejos de memoria

La subrutina que presentamos trabaja sobre el tubo inicial T_0, y modifica sus elementos de forma que las moléculas del tubo de salida almacenan en $(F\sigma)$ el peso, con respecto a la función f, del subconjunto de A codificado en $(A\sigma)$ (las zonas $(R\sigma)$ y $(L\sigma)$ no intervienen en este proceso, pero serán útiles para usos más generales). Es decir,

$$\sum_{i=1}^{p} \sigma(i) f(i) = \sum_{j=p+r+1}^{p+r+q_f} \sigma(j)$$

El programa molecular que diseñamos en el modelo sticker para implementar dicho proceso es el siguiente:

```
Procedimiento Peso
Entrada: (T_0, f, p, r)
    para i = 1 hasta p hacer
        (T^+, T^-) ← separación(T_0, i)
        para j = 1 hasta f(i) hacer
            T^+ ← activar(T^+, p + r + f(A_{i-1}) + j)
        T_0 ← mezclar(T^+, T^-)
Salida: T_0
```

El número de operaciones moleculares que realiza este programa es del orden de $O(q_f)$. El número de tubos usados es del orden de $O(q_f)$. El volumen molecular coincide con el tamaño de la subfamilia de subconjuntos de A sobre la que se aplica la subrutina.

Verificación del programa diseñado

Para establecer la verificación formal del programa diseñado procedemos a re-etiquetar los tubos que se obtienen a lo largo de la ejecución.

```
Procedimiento Peso
Entrada: (T_0, f, p, r)
    para i = 1 hasta p hacer
        (T^+_{i,0}, T^-_i) ← separación(T_{i-1}, i)
        para j = 1 hasta f(i) hacer
            T^+_{i,j} ← activar(T^+_{i,j-1}, p + r + f(A_{i-1}) + j)
        T_i ← mezclar (T^+_{i,f(i)}, T^-_i)
Salida: T_p
```

Con el fin de que las pruebas sean más fáciles de seguir, subdividiremos la zona $(F\sigma)$ de la siguiente forma: para cada i tal que $1 \leq i \leq p$ consideramos la *región*:

$$R_i = \{p + r + f(A_{i-1}) + 1, ..., p + r + f(A_i)\}$$

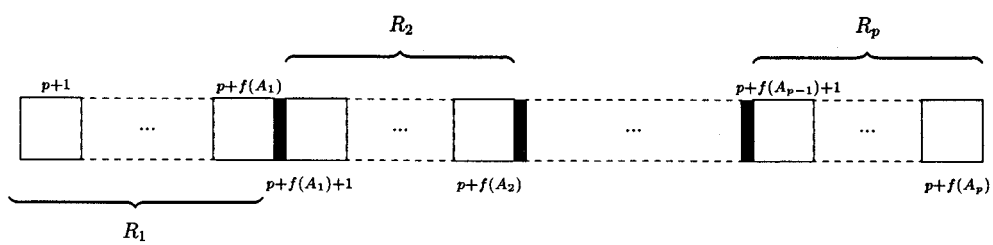


Figura 4.4. Regiones de la zona $(F\sigma)$

Definición 4.45 *Para cada $\sigma \in T_0$ y cada k tal que $1 \leq k \leq p$, notaremos por σ^k la molécula que se obtiene de σ tras la ejecución del paso k–ésimo del bucle principal del programa.*

Es decir, las moléculas σ^k proporcionan la *historia* de la molécula σ del tubo de entrada a lo largo de la ejecución del programa, del mismo modo en que se definió anteriormente. Teniendo presente la estructura del programa, es inmediato probar los siguientes resultados:

Lema 4.46

1. *La zona inicial de la molécula (que codifica el subconjunto de A) permanece invariable a lo largo de la ejecución del programa; es decir,*

$$\forall \sigma \in T_0 \; \forall k \; (1 \leq k \leq p \rightarrow (A\sigma) = (A\sigma^k)) \tag{4.1}$$

2. *Las moléculas que se obtienen en el paso k-ésimo del bucle principal son almacenadas en el tubo T_k; es decir,*

$$\forall \sigma \in T_0 \, \forall k \, (1 \leq k \leq p \rightarrow \sigma^k \in T_k) \tag{4.2}$$

3. *Todas las moléculas del tubo k-ésimo provienen de alguna molécula del tubo inicial; es decir,*

$$\forall k \, (1 \leq k \leq p \rightarrow \forall \tau \in T_k \, \exists \sigma \in T_0 \, (\sigma^k = \tau)) \tag{4.3}$$

4. *La ejecución de un paso del bucle principal no modifica las regiones correspondientes a pasos anteriores; es decir,*

$$\forall \sigma \in T_0 \, \forall i \, \forall k \, (1 \leq i \leq k \leq p \rightarrow \sigma^i_{|R_i} = \sigma^k_{|R_i}) \tag{4.4}$$

5. *Tras la ejecución del paso i-ésimo del bucle principal, la región R_i de σ ha sido modificada de acuerdo con el valor de $\sigma(i)$; es decir ,*

$$\forall \sigma \in T_0 \, \forall i \, \forall k \, (1 \leq i \leq k \leq p \rightarrow \sigma^k_{|R_i} \equiv \sigma(i)) \tag{4.5}$$

6. *La ejecución de un paso del bucle principal no modifica las zonas $(L\sigma)$ y $(R\sigma)$; es decir,*

$$\forall \sigma \in T_0 \, \forall k \, (1 \leq k \leq p \rightarrow (L\sigma) = (L\sigma^k) \wedge (R\sigma) = (R\sigma^k)) \tag{4.6}$$

El siguiente resultado asegura que el bucle principal modifica las regiones R_i de las moléculas, codificando el peso parcial del conjunto representado por cada una de ellas.

Proposición 4.47 *Sea $B \subseteq A$ tal que $\sigma_B \in T_0$. Entonces para cada k tal que $1 \leq k \leq p$, se tiene que:*

$$f(B \cap \{1, ..., k\}) = \sum_{j=p+r+1}^{p+r+f(A_k)} \sigma^k_B(j)$$

Demostración:
Del apartado 1 del lema 4.46 se sigue que

$$f(B \cap \{1, ..., k\}) = \sum_{i=1}^{k} f(i) \cdot \sigma_B(i) = \sum_{i=1}^{k} f(i) \cdot \sigma^k_B(i)$$

Por otro lado, de los apartados 1 y 5 del lema 4.46 se deduce que

$$f(i) \cdot \sigma_B^k(i) = \sum_{j \in R_i} \sigma_B^k(j)) \ (1 \leq i \leq k)$$

□

Corolario 4.48 *Para cada $B \subseteq A$ tal que $\sigma_B \in T_0$, existe $\tau \in T_p$ que verifica*

$$f(B) = \sum_{i=p+r+1}^{p+r+q_f} \tau(i)$$

Demostración:
Dado $B \subseteq A$, consideremos la molécula asociada $\sigma_B \in T_0$. Basta tomar $\tau = \sigma_B^p$, ya que

$$f(B) = f(B \cap \{1, ..., p\}) = \sum_{j=p+r+1}^{p+r+q_f} \sigma_B^p(j)$$

□

4.3. Análisis de los costes

Terminamos este capítulo resumiendo los costes en tiempo (número de operaciones moleculares) y en espacio (volumen molecular del tubo inicial), así como el número total de tubos usado en cada uno de los procedimientos diseñados en la sección anterior relativos a la representación de conjuntos finitos en el modelo sticker.

Procedimiento	**Op. mol.**	**Tubos usados**	**Vol. mol. inicial**
Rellenado	$O(qp)$	$O(q)$	2^q
Recubrimiento	$O(qp)$	$O(q)$	2^q
Ordenación Cardinal	$\frac{s(s+3)}{2}$	$2s$	t
Familias Disjuntas	$O(qp)$	$O(p)$	2^q
Funciones Peso	$O(q_f)$	$O(q_f)$	–

Capítulo 5

Resolución de problemas numéricos NP-completos en el modelo sticker

En este capítulo se estudian soluciones moleculares en el modelo sticker de algunos problemas **NP**-completos clásicos (para una descripción más detallada de los problemas presentados véase [26]). Las soluciones presentadas a continuación hacen uso de la batería de programas moleculares estudiados en el capítulo anterior usados como subrutinas, siendo la verificación formal de las mismas una consecuencia inmediata de las verificaciones vistas anteriormente.

5.1. El problema Subset-Sum

Problema: *Sea* $A = \{1, ..., p\}$ *y* $w : A \longrightarrow \mathbf{N}$ *una función peso. Sea* $k \in \mathbf{N}$ *tal que* $k \leq w(A) = q_w$. *Determinar si existe un subconjunto,* $B \subseteq A$, *cuyo peso sea exactamente* k.

A continuación diseñamos un programa molecular en el modelo sticker que resuelve el problema Subset–Sum.

La idea del programa es la siguiente:

- El tubo de entrada, T_0, será una $(p + q_w, p)$-librería.
- En una primera etapa (cálculo del peso), cada molécula, σ, del tubo de entrada se *rellena* con el fin de obtener en sus últimas q componentes el peso del subconjunto que codifica.
- Las moléculas del tubo resultante en la etapa anterior son ordenadas de acuerdo a su cardinalidad con respecto al peso.

- Finalmente, se lee el tubo k-ésimo de dicha ordenación, que contiene las moléculas del tubo de entrada que codifican subconjuntos de A de peso k, si las hay.

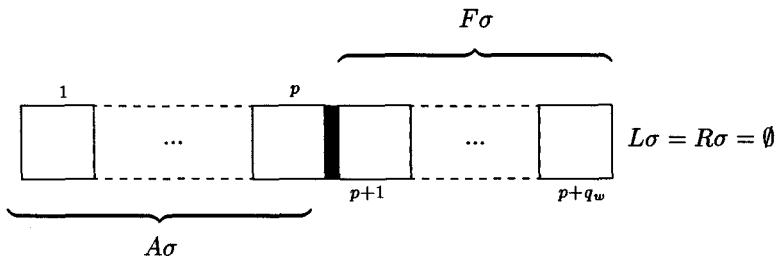


Figura 5.1. Zonas consideradas en los complejos

Estas ideas sugieren el diseño del siguiente programa molecular en el modelo sticker:

```
Procedimiento Subset_Sum
Entrada: (p, w, k)
```
$q_w \leftarrow \sum_{i=1}^{p} w(i)$
$T_0 \leftarrow (p + q_w, p)$-`librería`
$T_1 \leftarrow$ `Peso`$(T_0, w, p, 0)$
$T_{out} \leftarrow$ `Orden_Cardinal`$(T_1, p+1, p+q_w)[k]$
`leer`(T_{out})

El número de tubos usados (incluyendo las subrutinas) es $4 + 2q$ y el número de operaciones moleculares realizadas es $2p + q + 1 + \frac{q \cdot (q+3)}{2}$. El volumen molecular del tubo inicial es 2^p.

El siguiente resultado muestra la corrección del programa molecular. Es decir, toda molécula del tubo de salida codifica una solución correcta para el problema Subset–Sum.

Teorema 5.1 *(Corrección) Si $T_{out} \neq \emptyset$, existe $B \subseteq A$ tal que $w(B) = k$.*

Demostración:
Basta tomar $\tau \in T_{out}$ y aplicar el apartado 4.3 del lema 4.46 y la proposición 4.47 para obtener una molécula $\sigma \in T_0$ que verifica el resultado para B_σ.

□

El siguiente teorema prueba la completitud del programa molecular diseñado. Es decir, toda molécula del tubo de entrada que codifica una solución correcta del problema Subset–Sum está en el tubo de salida.

Teorema 5.2 *(Completitud) Sea $\sigma \in T_0$ tal que $w(B_\sigma) = k$. Entonces $T_{out} \neq \emptyset$.*

Demostración:
Sea $\sigma \in T_0$ tal que $w(B_\sigma) = k$. Del corolario 4.48, tras la ejecución del procedimiento de rellenado por peso, se obtiene una molécula $\tau = \sigma^p \in T_1$ verificando que $w(B_\sigma) = \sum_{i=p+1}^{p+q_w} \tau(i)$. Por tanto, $\tau \in T_{out}$.

□

Nota 5.3 El programa anterior no sólo resuelve el problema de decisión, sino que devuelve todas las soluciones del mismo. Es decir, en el tubo de salida T_{out} están todas las moléculas que codifican subconjuntos $B \subseteq A$ tales que $w(B) = k$ (y sólo esas).

5.2. El problema de la Mochila 0/1 acotado

Problema: *Sean $A = \{1, ..., p\}$ un conjunto finito no vacío, $w : A \longrightarrow \mathbf{N}$ una función peso, y $\rho : A \longrightarrow \mathbf{N}$ una función aditiva de valores. Sean $k, k' \in \mathbf{N}$ tales que $k \leq w(A) = q_w$ y $k' \leq \rho(A) = q_\rho$. Determinar si existe un subconjunto $B \subseteq A$ tal que $w(B) \leq k$ y $\rho(B) \geq k'$.*

La ide de un programa molecular que resuelve este problema en el modelo sticker es la siguiente:

- El tubo de entrada será una $(p + q_w + q_\rho, p)$-librería.
- En una primera etapa (cálculo del peso) se procede como en el problema anterior: cada molécula del tubo de entrada se rellena adecuadamente con el fin de codificar el peso de su subconjunto asociado.
- Las moléculas, σ, del tubo resultante se ordenan de acuerdo al cardinal de $w(A\sigma)$, obteniéndose los tubos $T_0, ..., T_{q_w}$, verificando que $\forall\sigma$ $(\sigma \in T_j \Rightarrow |w(A\sigma)| = j)$.
- Con el tubo $T_0 \cup ... \cup T_k$ se procede a una segunda etapa de cálculo del peso, de acuerdo con la función de valores, ρ, proporcionada por el problema.
- Las moléculas, σ, del tubo resultante se ordenan ahora de acuerdo con el cardinal de $\rho(A\sigma)$, obteniéndose los tubos $T_0, ..., T_{q_\rho}$ verificando que $\forall\sigma$ $(\sigma \in T_j \Rightarrow |\rho(A\sigma)| = j)$.

- Finalmente, se lee el contenido del tubo $T_{k'} \cup ... \cup T_{q_\rho}$, que contendrá las soluciones codificadas del problema.

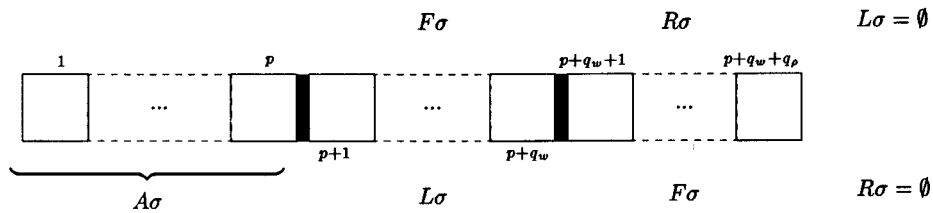


Figura 5.2. Zonas consideradas en los complejos de memoria

Estas ideas sugieren el diseño del siguiente programa molecular en el modelo sticker:

```
Procedimiento Mochila
Entrada: (p, w, ρ, k, k')
    q_w ← Σ_{i=1}^{p} w(i);  q_ρ ← Σ_{i=1}^{p} ρ(i);  T_0 ← (p + q_w + q_ρ, p)-librería
    T_0 ← Peso(T_0, w, p, 0)
    Orden_Cardinal(T_0, p + 1, p + q_w)
    T_1 ← ∅
    para i = 1 hasta k hacer
       T_1 ← mezclar(T_1, Orden_Cardinal(T_0, p + 1, p + q_w)[i])
    T_0 ← Peso(T_1, ρ, p, q_w)
    Orden_Cardinal(T_0, p + q_w + 1, p + q_w + q_ρ)
    T_1 ← ∅
    para i = k' hasta q_ρ hacer
       T_1 ← mezclar(T_1, Orden_Cardinal(T_0, p + q_w + 1, p + q_w + q_ρ)[i])
    leer(T_1)
```

El número de tubos usados por el programa (incluyendo, de nuevo, los usados por las subrutinas) es $5 + 2 \cdot \max\{q_w, q_\rho\}$, y el número de operaciones moleculares que se llevan a cabo en la ejecución es $4p + k - k' + \frac{q_w \cdot (q_w+5) + q_\rho \cdot (q_\rho+7)}{2} + 1$. El volumen molecular del tubo inicial es 2^p.

La verificación formal de este programa es similar a la realizada para el problema del Subset–Sum, por lo que omitimos su prueba, ya que no añade ninguna idea nueva a los métodos de verificación en el modelo sticker.

5.3. El problema de la Mochila 0/1 no acotado

Problema: *Bajo las mismas condiciones que en el problema de la mochila 0/1 acotado, determinar un subconjunto $B \subseteq A$ que verifique la siguiente condición:*

$$\rho(B) = \max\{\rho(C) : \ C \subseteq A \wedge w(C) \leq k\}$$

Una solución molecular a este problema se obtiene de la solución dada para el problema acotado, cambiando la etapa final de dicho programa: tras la ordenación de los tubos de acuerdo con la función de valores, se escogerá el tubo no vacío de mayor índice.

Las zonas que se consideran en las moléculas para resolver este problema son las mismas que se consideraban para el problema anterior.

```
Procedimiento Mochila_no_acotado
Entrada: (p, w, ρ, k)
    q_w ← Σ_{i=1}^{p} w(i);  q_ρ ← Σ_{i=1}^{p} ρ(i);  T_0 ← (p + q_w + q_ρ, p)-librería
    T_0 ← Peso(T_0, w, p, 0)
    Orden_Cardinal(T_0, p + 1, p + q_w)
    T_1 ← ∅
    para i = 0 hasta k hacer
      T_1 ← mezclar(T_1, Orden_Cardinal(T_0, p + 1, p + q_w)[i])
    T_0 ← Peso(T_1, ρ, p, q_w)
    i ← q_ρ;  t ← 0
    Orden_Cardinal(T_0, p + q_w + 1, p + q_w + q_ρ)
    mientras i ≥ 1  ∧  t = 0 hacer
      T' ← Orden_Cardinal(T_0, p + q_w + 1, p + q_w + q_ρ)[i]
      si T' ≠ ∅ entonces:
        leer(T');  t ← 1
      si no:
        i ← i − 1
```

El número de tubos usados por el programa es $5 + 2 \cdot \max\{q_w, q_\rho\}$ y el número de operaciones moleculares que se realizan en la ejecución del mismo, $4p + k - k' + \frac{q_w \cdot (q_w+5) + q_\rho \cdot (q_\rho+9)}{2}$, se obtiene directamente del caso acotado. El volumen molecular del tubo inicial es 2^p.

5.4. El problema SET PACKING

Problema: *Sean $A = \{1, \ldots, p\}$ un conjunto finito, $\mathcal{F} = \{B_1, \ldots, B_q\}$ una familia de subconjuntos de A, y $k \in \mathbf{N}$. Determinar si existen k subconjuntos de $\mathcal{F}$ disjuntos dos a dos.*

La idea de un programa molecular que resuelve este problema en el modelo sticker es la siguiente:

- El tubo de entrada, T_0, será una $(p+q, q)$-librería.
- Usando la subrutina `Disjuntos` se obtiene un tubo T_1 que contiene todas las subfamilias de $\mathcal{F}$ cuyos elementos son disjuntos dos a dos.
- Se ordenan los elementos de T_1 según su cardinalidad.
- Finalmente, se lee el tubo $T(k)$ que contiene las moléculas del tubo de entrada que codifican subfamilias de $\mathcal{F}$ con k elementos formadas por conjuntos disjuntos dos a dos.

Estas ideas sugieren el siguiente programa molecular:

```
Procedimiento Set_Packing
Entrada: (T0, k) (T0 es una (p+q,q)-librería)
        T1 ← Disjuntos(T0)
        T[.] ← Orden_Cardinal(T1)
        Leer(T(k))
```

El número de operaciones moleculares es del orden $O(q \cdot p + q^2)$. El número total de tubos usados (incluyendo los utilizados en las subrutinas) es del orden de $O(p)$. El volumen molecular del tubo inicial es 2^q.

5.5. El problema EXACT-COVER

Problema: *Sean $A = \{1, \ldots, p\}$ un conjunto finito y $\mathcal{F} = \{B_1, \ldots, B_q\}$ una familia de subconjuntos de A. Determinar si existen $i_1, \ldots, i_k$ tales que $\{B_{i_1}, \ldots, B_{i_k}\}$ es una partición de A.*

La idea de un programa molecular que resuelve este problema en el modelo sticker es la siguiente:

- Usando la subrutina `Disjuntos` se obtiene un tubo que contiene todas las subfamilias de $\mathcal{F}$ cuyos elementos son disjuntos dos a dos.

- Finalmente, haciendo uso de la subrutina `Selección_Recubrimientos` se extraen todas aquellas moléculas que codifican subfamilias de $\mathcal{F}$ que determinen un recubrimiento en A.

Estas ideas sugieren el siguiente programa molecular:

```
Procedimiento Exact_Cover
Entrada: T0 (una (p+q,q)-librería)
        T0 ← Disjuntos(T0)
        T0 ← Selección_Recubrimientos(T0)
```

El número de operaciones moleculares de este programa es del orden $O(q{\cdot}p{+}p) = O(q{\cdot}p)$. El número total de tubos usados (incluyendo los utilizados en las subrutinas) es . El volumen molecular del tubo inicial es 2^q.

5.6. El Problema del recubrimiento Minimal

Problema: *Sea $A = \{1 \ldots, p\}$. Sea $\mathcal{F} = \{B_1, \ldots, B_q\}$ una familia finita de subconjuntos de A. Encontrar el menor recubrimiento de A por elementos de $\mathcal{F}$.*

La primera solución molecular a este problema fue dada como ejemplo de la potencia del modelo sticker por S. Roweis y otros, en [77]. A continuación presentamos otra solución ligeramente distinta utilizando algunas subrutinas estudiadas en el capítulo anterior.

La idea de un programa molecular que resuelve este problema en el modelo sticker es la siguiente:

El tubo de entrada es una $(p + q, q)$-librería, de tal forma que codifica todas las posibles subfamilias de $\mathcal{F}$, tal y como se vio en el problema del rellenado, y el programa lo estructuramos en tres fases:

- Fase 1: *Fase de rellenado*: en la que se rellena la parte de las moléculas del tubo de entrada correspondiente al subconjunto A, en función de la subfamilia codificada en las mismas.

- Fase 2: *Fase de selección de recubrimientos*: en la que se extraen aquellas moléculas en las que la subfamilia codificada es, en efecto, un recubrimiento del conjunto A.

- Fase 3: *Fase de selección de un recubrimiento minimal*: en la que se extraen de entre todos los recubrimientos de A aquellos de menor cardinalidad.

Teniendo presente que en la sección anterior hemos diseñado procedimientos que nos permiten realizar cada una de las fases descritas, un programa molecular que resuelve el problema del recubrimiento minimal en el modelo sticker es el siguiente:

```
Procedimiento Recubrimiento_Minimal
Entrada: T_0 (una (p+q,q)-librería)
        T_1 ← Rellenado(T_0)
        T_2 ← Selección_Recubrimientos(T_1)
        T[.] ← Orden_Cardinal(T_2, 1, q)
        t ← 0; i ← 1
        mientras i ≤ q ∧ t = 0 hacer
            si T[i] ≠ ∅ entonces:
                leer(T[i]); t ← 1
            si no:
                i ← i + 1
```

El número de operaciones moleculares que realiza este programa es de orden $O(q \cdot p + q^2)$. El número total de tubos usados (incluyendo los utilizados en las subrutinas) es del orden de $O(q)$. El volumen molecular del tubo inicial es 2^q.

5.7. Análisis de los costes

Finalizamos el capítulo resumiendo los costes en tiempo (número de operaciones moleculares) y en espacio (volumen molecular del tubo inicial), así como el número total de tubos usados en cada uno de los programas diseñados para resolver los problemas **NP**-completos que se citan, en el modelo sticker.

Procedimiento	**Op. mol.**	**Tubos usados**	**Vol. mol. in.**
Subset-Sum	$2p + q + 2 + \frac{q(q+3)}{2}$	$4 + 2q$	2^p
Mochila 0/1 Acotado	$4p + k - k' + \frac{q_w(q_w+5)+q_\rho(q_\rho+7)}{2} + 1$	$5 + 2\,\text{máx}\{q_w, q_\rho\}$	2^p
Mochila 0/1 no Acotado	$4p + k - k' + \frac{q_w(q_w+5)+q_\rho(q_\rho+9)}{2} + 1$	$5 + 2\,\text{máx}\{q_w, q_\rho\}$	2^p
Set-Packing	$O(qp + q^2)$	$O(p)$	2^q
Exact-Cover	$O(qp)$	$O(p)$	2^q
Recubrimiento Minimal	$O(qp + q^2)$	$O(q)$	2^q

Capítulo 6

Computación celular con membranas

En Octubre de 1998 Gh. Păun introduce un nuevo modelo de computación de tipo paralelo y distribuido, denominado *P sistemas de transición* [56]. Este modelo también se conoce con el nombre de *computación celular con membranas*, por las razones que veremos más adelante.

Este capítulo está estructurado como sigue. En la primera sección se presenta una descripción informal, intuitiva, de los P sistemas básicos de transición, de su funcionamiento y de las características especiales de los mismos. En la sección 2 se presenta una formalización de este nuevo modelo de Computación Natural que permite atacar con garantías la verificación de sistemas desarrollados en este modelo. En la tercera sección se aborda el estudio de los P sistemas de transición como dispositivos generadores de multiconjuntos (de números o de relaciones), como dispositivos de reconocimiento y como máquinas de cálculo. En la sección 4 se estudia la potencia computacional de los P sistemas de transición y en la sección 5 se justifica que este modelo de computación proporciona herramientas para atacar la conjetura **P=NP**. En la última sección de este capítulo se estudian algunas variantes de los P sistemas que permitan resolver, en tiempo polinomial y en modo determinista, problemas que son computacionalmente intratables, si bien el parámetro exponencial suele enmascararse en las descripciones de las variantes.

Como primera nota diferenciadora con respecto a los modelos de computación molecular basados en ADN vistos en los capítulos anteriores, conviene hacer notar que este modelo no está descrito, propiamente, a través de un lenguaje de programación; es decir, no proporciona una serie de operaciones básicas susceptibles de ser secuenciadas sobre un dato de entrada para obtener un resultado final (solución del problema que se pretende resolver). En este modelo se generan *máquinas* (al modo de las máquinas de Turing) cuya ejecución modifica el contenido de las distintas

componentes que la integran hasta llegar, en su caso, a un estado de parada (en el que la máquina deja de funcionar). En este sentido, la ejecución de los P sistemas se podría decir que es independiente del usuario, puesto que una vez construido el sistema no es necesario, en principio, intervenir para *dirigir* la ejecución.

6.1. Los P sistemas de transición

Hasta ahora hemos introducido modelos de computación natural en los que se simula el modo en que la naturaleza *calcula* a un nivel genético (algoritmos genéticos y computación molecular basada en ADN), o a un nivel neuronal (redes neuronales). Un estudio más detallado del funcionamiento de los organismos vivos nos sugiere un nuevo nivel de computación no analizado hasta ahora: *el nivel celular.*

El comportamiento de una célula puede ser considerado como el de una máquina que realiza un cierto proceso de cálculo: una máquina no trivial, desde el punto de vista biológico, en la que por medio de una distribución jerárquica de membranas interiores se produce el flujo y alteración de las componentes químicas que la propia célula procesa.

Por supuesto, los procesos que se producen en una célula son lo suficientemente complejos como para no intentar modelizarlos completamente en este trabajo y, posiblemente, un modelo de computación que intente simular *literalmente* esos procesos dejaría de ser práctico, salvo desde un punto de vista biológico. Se pretende crear un modelo de computación abstracto que simule, de la forma más aproximada posible, el comportamiento de las células y que nos permita (al menos en principio) obtener soluciones alternativas de problemas computacionalmente intratables desde un punto de vista convencional. Para ello, hay que hacer emerger todas aquellas características del comportamiento y constitución de la célula que puedan ser de utilidad para la elaboración de un modelo de computación que sea a la vez potente (en cuanto a los problemas que pueda resolver) y sencillo (en cuanto a su definición, implementación y ejecución).

La primera característica que llama la atención en cuanto a la estructura interna de la célula, y que será útil para la definición de los P sistemas de transición, es el hecho de que las distintas partes del sistema biológico que la componen se encuentran delimitadas por varios tipos de *membranas* (en su sentido más amplio), desde la propia membrana que separa el exterior de la célula del interior de la misma, hasta las distintas membranas que delimitan las vesículas interiores. Además, con respecto a la funcionalidad de estas membranas en la naturaleza, interesa el hecho de que

no generan compartimentos estancos sino que permiten el paso (flujo) de ciertos compuestos químicos, algunas veces de forma selectiva e incluso en una sola de las direcciones. Estas ideas han sido consideradas con anterioridad por G. Berry y V. Manca en [10] y [45], respectivamente.

A pesar de que el modelo que se va a definir tiene inspiración biológica, puede resaltarse el hecho de que constituye igualmente un buen modelo teórico de computación distribuida, en donde distintas unidades de cálculo trabajan independientemente pero estructuradas en una cierta jerarquía vertical; por ejemplo, la jerarquización usada en las conexiones establecidas en redes como internet puede ser representada como una estructura de membranas.

En líneas generales, el sistema que se va a definir a continuación está basado en una *estructura de membranas*, como modelo matemático de ordenación física de la célula. En esta agrupación (jerarquizada) de membranas se introducirán unos elementos denominados *objetos*, a modo de las distintas componentes con las que se van a realizar las operaciones internas y el traspaso de información entre las membranas, y una serie de *reglas de evolución* que nos permitirán, eventualmente, modificar los objetos presentes en cada membrana y establecer una cierta comunicación entre las mismas. Los objetos introducidos en las membranas no forman propiamente un conjunto, sino que pueden aparecer repetidos (existir varias copias idénticas de un mismo elemento) y, además, será irrelevante el orden de ubicación de los mismos en las membranas. Por ello, para implementar la manipulación de objetos en el modelo, se necesita el concepto de *multiconjunto*. Como característica adicional, las reglas de evolución pueden hacer desaparecer la membrana sobre la que se están ejecutando, simulando el hecho biológico de la *disolución* de la membrana, de tal modo que a partir del instante en que esta disolución ocurre, los objetos de la membrana disuelta pasan a formar parte de la membrana que la contuviese directamente (como parece natural, se exigirá que la membrana más exterior en la estructura no puede ser disuelta).

En los P sistemas de transición se puede considerar un cierto orden en la aplicación de las reglas a través de una *relación de prioridad*, de modo que si dos reglas pueden ser aplicadas simultáneamente, y hay una relación de prioridad entre ellas, entonces sólo se podrá ejecutar aquella de prioridad más alta. En caso de existencia de varias reglas en una misma membrana que puedan ser ejecutadas en un cierto instante, el sistema actuará de manera *no determinista*; es decir, el sistema tendrá varias posibilidades de evolución, pudiéndose ejecutar cualquiera de las reglas de igual prioridad y que sean aplicables. Además, las reglas deben aplicarse de forma *maximal*, en el sentido de que en cada paso de computación deben agotarse todos

los objetos afectados por reglas que se puedan aplicar.

Para la evolución global del sistema, se supone que hay una especie de *reloj universal* que marca las actuaciones de todos los elementos que lo integran; es decir, simultáneamente actúan todas las reglas dentro de cada membrana, en todas las membranas presentes, sin interferencias directas entre ellas: en un paso de reloj se considera la ejecución de todas las reglas (que pueden ser aplicadas) de cada membrana del P sistema.

Tal y como se ha descrito informalmente los P sistemas de transición, se observa que una de las características que puede resultar de mayor interés es el *paralelismo* que implementa. Conviene hacer notar que el paralelismo presente en los P sistemas se produce a dos niveles bien diferenciados: en un primer nivel, dentro de cada membrana, todas las reglas que puedan aplicarse lo harán de manera simultánea; en un segundo nivel, todas las membranas presentes en el sistema están trabajando a la vez y de forma sincronizada.

A continuación se van a describir los *P sistemas de transición con membrana de salida* siguiendo las ideas originales de su creador, Gh. Păun [56].

6.1.1. Alfabetos y multiconjuntos

Un *alfabeto*, Σ, es un conjunto no vacío (finito o infinito) cuyos elementos se denominan *símbolos*. Utilizando los símbolos del alfabeto podemos construir cadenas finitas y ordenadas de símbolos, a modo de *palabras*. El número de elementos que tiene una palabra se denomina *longitud* de la misma. Habitualmente, el conjunto de las cadenas que constan de n símbolos del alfabeto Σ se denotan por Σ^n. Por extensión, la palabra que usa 0 símbolos (es decir, de longitud 0) también se considera una palabra sobre el alfabeto, que se denomina *palabra vacía* y se denota por λ. Como caso particular de palabra, si a es un símbolo del alfabeto, notaremos por a^n la palabra de longitud n que consta de n repeticiones del símbolo a. Al conjunto de todas las palabras que se pueden formar sobre un alfabeto, Σ, se denota por Σ^*. Entre las palabras se pueden definir de manera natural operaciones tales como la *concatenación*, con el significado intuitivo habitual, y relaciones como *ser prefijo, subpalabra* o *sufijo*, etc. Un *lenguaje* sobre un alfabeto es un conjunto de palabras sobre ese alfabeto; es decir, L es un lenguaje sobre Σ si $L \subseteq \Sigma^*$.

Como ya se ha comentado anteriormente en la disciplina de Computación Natural suele ser habitual tratar con una extensión del concepto de conjunto que permite la aparición de elementos repetidos. Es lo que formalmente se denomina *multiconjunto*. Dentro de un multiconjunto, al número de veces que aparece repetido un elemento

se le denomina *multiplicidad* de dicho elemento, entendiendo por multiplicidad 0 si dicho elemento no aparece (de esta forma, un conjunto es un multiconjunto en el que cada elemento que aparece en él tiene multiplicidad 1). Al igual que el contenido de los conjuntos suele notarse enmarcado por los símbolos auxiliares $\{,\}$, cuando hablamos de multiconjuntos usaremos como símbolos auxiliares $\{\{,\}\}$.

Muchas veces, y por razones de brevedad en la escritura, es habitual relacionar los multiconjuntos que se pueden formar con elementos de un determinado conjunto no vacío, con las palabras que se pueden escribir considerando dicho conjunto como alfabeto. Así, dada una palabra sobre un conjunto no vacío, basta considerar como multiconjunto asociado el formado por los elementos que integran la palabra, donde la multiplicidad de cada uno de ellos será el número de apariciones que tenga en la palabra. Recíprocamente, si tenemos un multiconjunto, para formar una palabra asociada, basta tomar todos los elementos del multiconjunto y formar una cadena con ellos. Es obvio que esta relación no es biyectiva ya que un mismo multiconjunto puede generar muchas palabras haciendo variar el orden en que se toman los símbolos al escribirla (por ejemplo, el multiconjunto $\{\{a, a, b\}\}$ tiene como palabras asociadas a^2b, ba^2 y aba).

6.1.2. Estructuras de membranas

Para introducir las estructuras de membranas, vamos a considerar, en primer lugar, el lenguaje MS sobre el alfabeto $\{[,]\}$. Por definición, MS es el menor lenguaje sobre $\{[,]\}$ que verifica simultáneamente las condiciones siguientes:

1. $[\,] \in MS$.
2. Si $\mu_1, \ldots, \mu_n \in MS$, entonces $[\mu_1 \ldots \mu_n] \in MS$.

Intuitivamente se quieren describir las estructuras de membranas a través de pares de corchetes *coincidentes*, imponiendo que dentro de las membranas, el orden de aparición no sea relevante. Para ello, se define una relación, $\sim$, sobre los elementos del lenguaje MS de forma que estén relacionadas todas aquellas cadenas que reflejen la misma estructura (salvo el orden):

$$x \sim y \Leftrightarrow x = \mu_1\mu_2\mu_3\mu_4 \wedge y = \mu_1\mu_3\mu_2\mu_4 \wedge \mu_1\mu_4, \mu_2, \mu_3 \in MS$$

Si notamos por $\sim^*$ la clausura transitiva y reflexiva de la relación anterior, entonces se tiene que la relación obtenida es de equivalencia. Notaremos por $\overline{MS}$ el conjunto de clases de equivalencia de MS con respecto a dicha relación, $\sim^*$, y a sus elementos

se les llama *estructuras de membranas.* Dentro de una estructura de membranas, a cada par de corchetes *coincidentes* se le denomina *membrana.* El número de membranas de una estructura de membranas, μ, se denomina *grado* de la estructura, y se notará por $deg(\mu)$. A la estructura *más externa* se le llama *piel* (*skin*), y aquellas membranas que no contienen otras membranas en su interior se denominan *elementales.*

La *profundidad* de una estructura, μ, se denota por $dep(\mu)$, y se define recursivamente como sigue:

1. $dep([\,]) = 1$.

2. $dep([\mu_1 \ldots \mu_n]) = 1 + max\{dep(\mu_i) : \ 1 \leq i \leq n\}$.

Se puede representar gráficamente las estructuras de membranas como un conjunto de curvas simples cerradas (a modo de *diagrama de Venn*) manteniendo la jerarquía impuesta por la cadena que la representa en $\overline{MS}$:

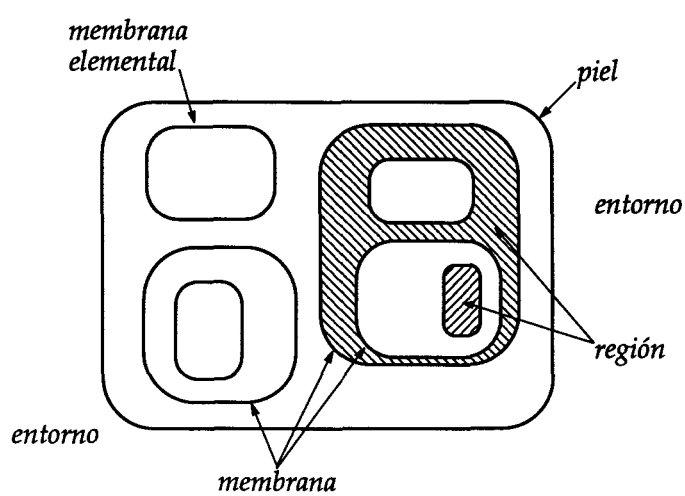


Figura 6.1. Representación gráfica de una estructura de membranas.

A partir de esta representación se puede considerar el concepto intuitivo de *región* delimitada por una membrana μ, como el espacio comprendido entre dicha membrana y las inmediatamente interiores. Es evidente que existe una biyección natural entre las membranas de una estructura y las regiones que delimitan, siendo habitual identificar unas con otras.

6.1.3. Células

Una vez que se han introducido las estructuras de membranas, el siguiente paso para poder modelizar la célula consiste en dotar de contenido a dichas membranas; es decir, asociar a cada región de una estructura de membranas un multiconjunto de objetos de un determinado alfabeto prefijado. El resultado obtenido por medio de esta asociación es lo que se conoce como *célula* (en el trabajo original, Gh. Păun le dio el nombre de *super-célula*).

Las nociones de grado, profundidad y región, definidas para las estructuras de membranas se extienden de forma natural a las células. El multiconjunto asociado a una región (o membrana) de una célula se denomina *contenido* de la región, y el número total de objetos de una región se denomina *tamaño* de la región.

Si una membrana, m, está situada dentro de otra membrana, m', de tal manera que ambas contribuyen a delimitar la misma región (cuyo contenido son los objetos dentro de m' que están situados fuera de m), entonces se dice que los objetos de m' son *adyacentes* a la membrana m. En este caso, también diremos que las membranas m y m' son *adyacentes*.

6.1.4. Evolución de los P sistemas de transición

Hasta ahora se han introducido los elementos necesarios para poder definir los P sistemas de transición, pero todavía no se ha explicado cómo se les puede dotar de procedimientos que permitan hacerlos evolucionar, transformando los objetos que contienen y, en su caso, propiciando la comunicación entre las membranas. La herramienta que permitirá implementar estas acciones son los *conjuntos de reglas*, asociados a las membranas.

Un *P sistema de transición* de grado n, con $n \geq 1$, es una tupla

$$\Pi = (V, \mu, m_1, \ldots, m_n, (R_1, \rho_1), \ldots, (R_n, \rho_n), i_0)$$

en donde:

- V es un alfabeto, cuyos elementos se denominan *objetos*.
- μ es una estructura de membranas de grado n, con las membranas (regiones) etiquetadas biyectivamente con elementos de un conjunto de etiquetas Λ. Será habitual usar como conjunto de etiquetas $\Lambda = \{1, \ldots, n\}$.
- Para cada i tal que $1 \leq i \leq n$, m_i es un multiconjunto finito sobre V. El multiconjunto m_i se considera asociado a la región de μ etiquetada por i.

- Para cada i tal que $1 \leq i \leq n$, R_i es un conjunto de *reglas de evolución* sobre V. El conjunto R_i se considera asociado a la región de μ etiquetada por i. Para cada i tal que $1 \leq i \leq n$, ρ_i es un orden parcial estricto sobre R_i, que especifica la prioridad entre las reglas del conjunto R_i.

  Una *regla de evolución* es un par ordenado (u, v) (habitualmente escrito en la forma $u \rightarrow v$), en donde u es un multiconjunto sobre V y $v = v'$ o $v = v'\delta$, siendo v' un multiconjunto sobre el alfabeto $(V \times \{here, out\}) \cup (V \times \{in_j : 1 \leq j \leq n\})$, y $\delta \notin V$ es un símbolo especial.

  La longitud de u se denomina *radio* de la regla (u, v).

- El número i_0 verifica que $1 \leq i_0 \leq n$ y representa la *membrana de salida* del P sistema.

Cualquiera de los multiconjuntos asociados a las membranas pueden ser vacíos, así como los conjuntos de reglas o las relaciones de prioridad.

A la tupla $(\mu, m_1, \ldots, m_n, (R_1, \rho_1), \ldots, (R_n, \rho_n))$ se le denomina *configuración inicial* del P sistema Π. En general, una configuración de Π es una tupla

$$(\mu', m'_{i_1}, \ldots, m'_{i_k}, (R_{i_1}, \rho_{i_1}), \ldots, (R_{i_k}, \rho_{i_k}))$$

en donde

- μ' es una estructura de membranas que se obtiene de μ eliminando las membranas distintas de $i_1, \ldots, i_k$.

- $m'_{i_1}, \ldots, m'_{i_k}$ son multiconjuntos sobre V.

- el conjunto $\{i_1, \ldots, i_k\}$ está contenido en $\{1, \ldots, n\}$, y, además, debe contener la etiqueta asociada a la membrana piel.

Sean C_1 y C_2 dos configuraciones de un P sistema de transición Π, con

$$C_1 = (\mu', m'_{i_1}, \ldots, m'_{i_k}, (R_{i_1}, \rho_{i_1}), \ldots, (R_{i_k}, \rho_{i_k}))$$

$$C_2 = (\mu', m'_{j_1}, \ldots, m'_{j_l}, (R_{j_1}, \rho_{j_1}), \ldots, (R_{j_l}, \rho_{j_l}))$$

Diremos que C_2 se obtiene de C_1 en *un paso* de Π (*una transición*), y lo notaremos $C_1 \Rightarrow_\Pi C_2$, si se puede pasar de C_1 a C_2 usando las reglas de evolución que aparecen en $R_{i_1}, \ldots, R_{i_k}$ de la siguiente forma:

- Consideremos una regla $u \rightarrow v$ de un conjunto R_{i_t}. Se analiza la región de μ' asociada a la membrana i_t. Si los objetos mencionados por u, con las multiplicidades especificadas por u, aparecen en m'_{i_t}, entonces estos objetos pueden evolucionar de acuerdo con la regla $u \rightarrow v$. La regla puede ser aplicada si no hay ninguna otra regla de prioridad superior que se pueda aplicar a la vez (es decir, se examinan las reglas a aplicar en orden decreciente según su prioridad). Todos los objetos de u son *agotados* en el uso de la regla $u \rightarrow v$; es decir, el multiconjunto definido por u es *eliminado* del multiconjunto m'_{i_t}.

- El resultado de usar la regla $u \rightarrow v$ viene determinado por v:

  - Si un objeto aparece en v en un par de la forma $(a, here)$, entonces permanecerá en la misma región, i_t (es habitual que al especificar las reglas, los pares de la forma $(a, here)$ se escriban simplemente a, omitiendo *here*).
  - Si un objeto aparece en v en un par de la forma (a, out), entonces a *saldrá* de la membrana i_t, pasando a formar parte de la membrana inmediatamente superior (si estuviera en la más exterior, en la piel, entonces simplemente abandona la célula y pasa al entorno).
  - Si un objeto aparece en v en un par de la forma (a, in_q), entonces a será añadido al multiconjunto m'_q, en el caso en que la membrana q sea inmediatamente interior a la membrana i_t; es decir, si los objetos de a son adyacentes a la membrana q. En caso contrario, la regla no será aplicable.
  - Si el símbolo δ aparece en v, entonces la membrana i_t es eliminada (*disuelta*) al igual que el par (R_{i_t}, ρ_{i_t}). Además, por la acción de la disolución, su contenido, m'_{i_t}, es añadido (como multiconjunto) a la región asociada a la membrana inmediatamente superior a i_t. No se permite la disolución de la membrana piel, luego no será aplicable una regla del tipo $u \rightarrow a\delta$ que aparezca en la piel.

Todas las operaciones antes descritas se realizan en paralelo, para todas las posibles reglas $u \rightarrow v$ aplicables (en forma no determinista), para todas las ocurrencias de multiconjuntos u en la región asociada a cada regla y para todas las regiones simultáneamente. Conviene hacer notar que no aparecen contradicciones a causa de la disolución simultánea de varias membranas, ni a la aparición de símbolos de la forma (a, out) y δ. Si al mismo tiempo tenemos (a, in_q) en el exterior de la membrana q y δ en el interior de dicha membrana, el efecto es el mismo que $(a, here)$.

Hay dos posibilidades a la hora de considerar la relación de prioridad dada sobre las reglas:

- **Versión Fuerte**: Si la regla r_1 tiene mayor prioridad que la regla r_2 y se puede aplicar r_1, entonces, aunque sobraran elementos para que se pudiera aplicar r_2, no se hará. Una posible interpretación de esta versión se encuentra en el consumo de energía: en cada paso de transición tenemos una cantidad fija de energía para poder aplicar las reglas, de tal manera que las reglas de prioridad superior consumen la suficiente energía como para que no quede energía para reglas de prioridad inferior.

- **Versión débil**: Si la regla r_1 tiene mayor prioridad que la regla r_2 y, además, la regla r_2 puede aplicarse (sin perjuicio para r_1), entonces se aplicará. La interpretación de esta versión radica en que la energía de la que dispone el sistema es ilimitada.

En nuestra memoria consideraremos la versión fuerte en los P sistemas de transición.

Si en un P sistema de transición Π hay reglas de radio mayor o igual 2, el sistema se dice *cooperativo*; en caso contrario, se dice *no cooperativo*. Un P sistema se dice *catalítico* si existen objetos $c_1, \ldots, c_k$, que llamaremos *catalizadores*, de tal manera que las reglas del sistema son de la forma $a \to v$ o $c_i a \to c_i v$, donde a no es un catalizador y v es una cadena que no contiene catalizadores. Es decir, un catalizador es un objeto que no es transformado por la aplicación de las reglas, pero que es necesario para que se produzca la ejecución de las mismas.

Una *computación*, C, de un P sistema de transición Π, es una sucesión (finita o infinita) de configuraciones $C_0 \Rightarrow_\Pi C_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_r$, con $r \geq 0$, de tal manera que C_0 es la configuración inicial de Π y para cada $i \geq 0$ se verifica que C_{i+1} se obtiene de C_i por un paso de Π. Diremos que una computación, $C_0 \Rightarrow_\Pi C_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_r$ es de *parada* si $r \in \mathbf{N}$ y en la configuración C_r (denominada *final*) no hay reglas que se puedan aplicar; es decir, si no existe ninguna configuración que se pueda obtener por transición a partir de C_r. Diremos que una computación, $C_0 \Rightarrow_\Pi C_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_r$ es *exitosa* si es de parada y, además, la membrana i_0 aparece en C_r como membrana elemental; es decir, i_0 no debe contener otras membranas en su interior, en la configuración C_r. En este caso, a la configuración C_r se le denomina una *configuración exitosa* de dicha computación.

Así pues, un P sistema de transición se puede considerar como una máquina generadora de multiconjuntos, en el siguiente sentido: un *multiconjunto*, m, se dice que es *generado* por el P sistema Π si existe alguna computación exitosa, de tal manera que m es el contenido de la membrana de salida de Π.

También se pueden considerar los P sistemas de transición como dispositivos

generadores de números naturales. Para ello, en vez de considerar como salida el contenido de la membrana de salida en la configuración exitosa, se considera el tamaño de dicho contenido. Al conjunto de números naturales generados por un P sistema de transición, Π, se denota por $N(\Pi)$.

Una generalización de este uso de los P sistemas de transición consiste en considerarlo como *generador de relaciones*. Para ello, si queremos generar una relación en $\mathbf{N}^k$, basta considerar k elementos distinguidos, $a_1, \ldots, a_k$ del alfabeto base V. En estas condiciones, $(n_1, \ldots, n_k)$ pertenece a la relación generada por el P sistema si existe una computación exitosa tal que en la membrana de salida de la correspondiente configuración exitosa, la multiplicidad del elemento a_i es n_i, para cada i tal que $1 \leq i \leq n$.

6.2. Formalización de los P sistemas de transición

El problema de describir una formalización completa de la definición de un P sistema de un tipo dado y, principalmente, de sus computaciones y los resultados de las mismas fue formulado explícitamente por Gh. Păun en [58]. La técnica que se utiliza en la formalización presentada en esta memoria es completamente distinta de las propuestas por A. V. Baranda y otros en [6] y por A. Obtulowicz en [53]. Asimismo es ligeramente diferente de la presentada por M. J. Pérez Jiménez y F. Sancho en [66].

6.2.1. Una sintaxis para los P sistemas de transición

Hemos visto en la sección anterior que las estructuras de membranas de un P sistema de transición constituyen una ordenación jerárquica de membranas (entendidas como vesículas) incluidas en una membrana piel que sirve a modo de separación entre el sistema y el entorno.

En primer lugar, vamos a introducir una serie de conceptos relativos a multiconjuntos y grafos que serán de utilidad para el desarrollo formal que pretendemos.

Multiconjuntos

Definición 6.1 *Un* multiconjunto *sobre un conjunto, A, es una aplicación $m : A \longrightarrow \mathbf{N}$. El* soporte *del multiconjunto m es el subconjunto de A: $supp(m) = \{a \in A : m(a) > 0\}$. Un multiconjunto se dice* vacío *(respectivamente finito) si su soporte es vacío (respectivamente finito), y se denotará por $m = \emptyset$.*

Denotaremos por $\mathbf{M}(A)$ el conjunto de todos los multiconjuntos sobre A (notaremos simplemente $\mathbf{M}$, cuando no haya confusión sobre el conjunto base).

Definición 6.2 *Dados $m_1, m_2 \in \mathbf{M}(A)$, definimos los siguientes conceptos:*

- *Inclusión:* $m_1 \leq m_2 \Leftrightarrow \forall j\ (j \in A \rightarrow m_1(j) \leq m_2(j))$.
- *Inclusión estricta:* $m_1 < m_2 \Leftrightarrow m_1 \leq m_2 \wedge m_1 \neq m_2$.
- *Unión: La aplicación* $+ : \mathbf{M}(A) \times \mathbf{M}(A) \rightarrow \mathbf{M}(A)$, *se define como sigue:* $(m_1 + m_2)(j) = m_1(j) + m_2(j)$, *para cada* $j \in A$.
- *Diferencia: La aplicación* $- : \mathbf{M}(A) \times \mathbf{M}(A) \rightarrow \mathbf{M}(A)$, *se define como sigue:* $(m_1 - m_2)(j) = \text{máx}\{m_1(j) - m_2(j), 0\}$, *para cada* $j \in A$.
- *Amplificación: La aplicación* $\otimes : \mathbf{N} \times \mathbf{M}(A) \rightarrow \mathbf{M}(A)$, *se define como sigue:* $(n \otimes m_1)(j) = n \cdot m_1(j)$, *para cada* $j \in A$.

El *tamaño* de un multiconjunto finito, m, que se notará por $|m|$, se define como sigue: $|m| = \sum_{a \in A} m(a)$. Esto es, el tamaño de un multiconjunto finito es la suma de las multiplicidades de los elementos que aparecen en él (obsérvese que esta suma es finita ya que sólo hay un número finito de términos no nulos).

Grafos

Un *grafo* es una estructura que expresa relaciones binarias entre los elementos de un cierto conjunto. Concretamente, un grafo es un par ordenado (V, E), en donde V es un conjunto finito cuyos elementos se denominan *vértices* o *nodos* y $E \subseteq \{\{u, v\} : u \in V \wedge v \in V \wedge u \neq v\}$ (en cuyo caso el grafo se dice que es *no dirigido*), o bien $E \subseteq V \times V$ (en cuyo caso el grafo se dice *dirigido*).

Sea $G = (V, E)$ un grafo y $u, v \in V$. Se dice que el vértice u es adyacente al vértice v si se verifica que $\{u, v\} \in E$ (en el caso de un grafo dirigido) o bien $(u, v) \in E$ (en el caso de un grafo dirigido). En este caso, notaremos genéricamente $[u, v] \in E$.

Un *camino de longitud* k desde un vértice u de $G = (V, E)$ hasta un vértice v de G es una sucesión finita $(x_0, x_1, \ldots, x_k)$ de vértices de G tal que $u = x_0$, $v = x_k$ y $\forall j\ (0 \leq j < k \rightarrow [x_j, x_{j+1}] \in E)$. La longitud del camino es el número de aristas que lo forman.

Si existe un camino desde un vértice u hasta un vértice v, diremos que v es *alcanzable* desde u en G, y se denotará por $u \rightsquigarrow_G v$. En este caso suele ser habitual decir que u y v están *conectados* en G.

Se dice que un grafo no dirigido es *conexo* si todo par de vértices distintos están conectados por algún camino en el grafo.

Un camino se dice *simple* si todas las aristas y todos los vértices que lo forman, excepto posiblemente el primero y el último, son diferentes entre sí. Un *ciclo* es un camino simple de longitud al menos 1 que comienza y acaba en el mismo vértice. Nótese que en un grafo no dirigido, un ciclo debe tener al menos longitud 3. Un grafo no dirigido sin ciclos se denomina *acíclico*.

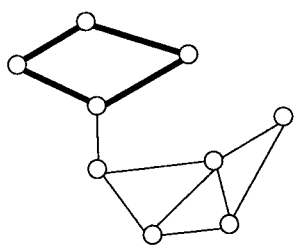


Figura 6.2. Grafo conexo en el que se ha destacado un ciclo

Un *árbol* (*libre*) es un grafo no dirigido, conexo y acíclico. Un *árbol enraizado* es un árbol con uno de sus vértices distinguido. A dicho vértice distinguido se le llama *raíz* del árbol. Normalmente, representaremos un árbol enraizado por un par ordenado en el que la primera componente es la raíz del árbol y la segunda componente es la *lista de adyacencias* del árbol (una lista formada por tantas listas como vértices tenga el árbol, y cada una de ellas representando el conjunto de vértices adyacentes al vértice correspondiente).

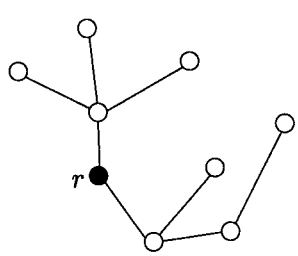


Figura 6.3. Árbol enraizado en el que se ha destacado la raíz

Sea G un árbol enraizado de raíz r. Dado un vértice u, cualquier otro vértice, v ($v \neq u$), que esté en el único camino que une la raíz r con u se dice que es un *antecesor propio* de u. Si v es un antecesor propio de u, también diremos que u es un *descendiente propio* de v. A los vértices del árbol que no tengan descendientes propios se les llama *hojas* del árbol enraizado.

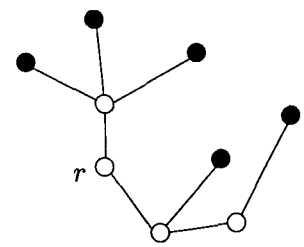


Figura 6.4. Árbol enraizado en el que se han destacado sus hojas

Para cada vértice, u, del árbol que no sea la raíz, notaremos por $f(u)$ (el *padre* de u en G) el único antecesor propio de u que verifica $\{f(u), u\} \in E$. El padre de la raíz no está definido. Para cualquier vértice, u, del árbol, notaremos por $Ch(u)$ (los *hijos* de u en G) el conjunto de descendientes propios de u tales que u es su padre.

En un árbol enraizado, $G = (V, E)$, de raíz r podemos definir una relación binaria $E^*(G)$ en V, de forma natural, como sigue: $(x, y) \in E^* \Leftrightarrow y \in Ch(x)$. Esta relación establece, de alguna forma, un direccionamiento sobre las aristas del árbol enraizado.

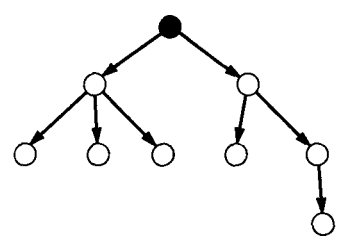


Figura 6.5. El mismo árbol enraizado con el direccionamiento inducido

Estructura de Membranas

La formalización del concepto de *estructura de membranas* hará uso de los conceptos sobre multiconjunto y sobre grafos que se acaban de introducir.

Definición 6.3 *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.

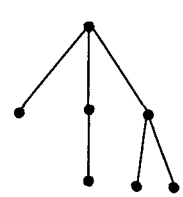


Figura 6.6. Árbol asociado a la estructura de la figura 6.1

En esta formalización las etiquetas de las membranas (tal y como aparecen en la visión informal anterior [56]) serán los propios vértices del grafo. Si esta formalización se extiende a otras versiones de P sistemas (por ejemplo, donde se permita la *creación* de membranas) puede ser útil un etiquetado independiente de los nodos; no obstante, para la versión que estamos formalizando esto no es necesario.

Para introducir el concepto de transición en los P sistemas se necesita formalizar previamente el concepto de *célula.*

Definición 6.4 *Una* célula *sobre un alfabeto, A, es un par ordenado (μ, M), en donde μ es una estructura de membranas y M es una aplicación de $V(\mu)$ en $\mathbf{M}(A)$, por la que a cada nodo del árbol se le asocia un multiconjunto sobre el alfabeto base.*

Si $C = (\mu, M)$ es una célula sobre A, denotaremos $\mu = (V(\mu), E(\mu))$; es decir, $V(\mu)$ y $E(\mu)$ son los conjuntos de vértices y aristas, respectivamente, del árbol enraizado etiquetado, μ, que determina la célula. Definimos el *grado* de una célula $C = (\mu, M)$, y lo denotaremos por $|C|$, como $|V(\mu)|$; esto es, el grado de C es el número total de membranas (o nodos) de la célula.

Definición 6.5 *Sea (μ, M) una célula sobre un alfabeto A. Una* regla *(*de evolución*) asociada a dicha célula es una 4-tupla $r = (d_r, v_r, \delta_r, x_r)$, en donde*

- *d_r es un multiconjunto sobre A.*
- *v_r es una función de dominio $V(\mu) \cup \{here, out\}$ y rango contenido en $\mathbf{M}(A)$, en donde $here, out \notin V(\mu)$ y $here \neq out$.*
- *$\delta_r \in \{\neg\delta, \delta\}$, con $\neg\delta, \delta \notin A$ y $\neg\delta \neq \delta$.*
- *x_r es un nodo de μ.*

Definición 6.6 *Sea $C = (\mu, M)$ una célula sobre un alfabeto, A, y $x \in V(\mu)$. Sea $r = (d_r, v_r, \delta_r, x_r)$ una regla de evolución asociada a C. Diremos que r está asociada a x si se verifica que $x_r = x$.*

Informalmente, una regla de evolución, r, de una membrana $x \in V(\mu)$, tiene una expresión del siguiente tipo:

$$a_1 \dots a_m \rightarrow (b_1, in_{j_1}) \dots (b_n, in_{j_n})(c_1, out) \dots (c_k, out)(h_1, here) \dots (h_l, here)s$$

con $\{\{j_1, \dots, j_n\}\}$ un multiconjunto sobre $V(\mu)$, los elementos a_i, b_i, c_i, h_i son símbolos de A y $s = \delta$ ó $s = \lambda$.

En esta formalización $d_r = \{\{a_1, \ldots, a_m\}\}$; para cada $y \in V(\mu)$, $v_r(y)$ es el multiconjunto formado por los elementos que aparecen en la regla bajo la forma (b, in_y); $v_r(out) = \{\{c_1, \ldots, c_k\}\}$, y $v_r(here) = \{\{h_1, \ldots, h_l\}\}$. La componente δ_r (δ o $\neg\delta$) representa si $s = \delta$ o $s = \lambda$. Por último, la componente x_r representa la membrana a la que está asociada la regla.

Finalmente, para dar la sintaxis completa del P sistema se necesita asignar a cada membrana de la estructura un conjunto de reglas que serán las que se puedan aplicar a los elementos presentes en esa membrana. Por ello, se define el concepto de colección de reglas asociada a una célula.

Definición 6.7 *Sea $C = (\mu, M)$ una célula sobre un alfabeto A. Una* colección R *de* reglas *(de evolución) asociada a C es una función con dominio $V(\mu)$ tal que para cada membrana $x \in V(\mu)$, $R(x)$ (que notaremos R_x) es un conjunto finito (posiblemente vacío) de reglas (de evolución) asociadas a x.*

Definición 6.8 *Sea $C = (\mu, M)$ una célula sobre un alfabeto, A, y sea R una colección de reglas (de evolución) asociada a C. Una* relación de prioridad sobre R *es una función, ρ, con dominio en $V(\mu)$ tal que para cada membrana $x \in V(\mu)$, $\rho(x)$ (que notaremos ρ_x) es un orden parcial estricto (posiblemente vacío) sobre R_x.*

El orden parcial estricto, ρ_x, sobre R_x, se interpretará como sigue: $(r', r) \in \rho_x$ significa que la regla r' tiene una prioridad estrictamente mayor que la regla r.

Definición 6.9 *Un P-sistema de transición es una 4-tupla, $\Pi = (A, C_0, \mathcal{R}, i_0)$, en donde:*

- *A es un conjunto finito no vacío (usualmente llamado alfabeto base).*
- *$C_0 = (\mu_0, M_0)$ es una célula sobre A.*
- *$\mathcal{R}$ es un par ordenado (R, ρ) donde R es una colección de reglas (de evolución) asociadas a C_0, y ρ es una relación de prioridad sobre R.*
- *i_0 es un nodo del árbol enraizado μ_0, que especifica la membrana de salida del P sistema de transición Π.*

6.2.2. Una semántica para los P sistemas de transición

La semántica de un P sistema de transición trata de describir la forma en que funciona como dispositivo computacional. Para definir formalmente la semántica se

necesita el concepto de *configuración*, que intenta capturar cada uno de los estados puntuales, instantáneos, que puede alcanzar un P sistema a lo largo de su ejecución/evolución. Básicamente, hemos visto que un P sistema es una estructura de membranas con objetos en su interior, con reglas de evolución propias para cada membrana y con especificaciones de entrada–salida. Un P sistema tiene como únicos elementos variables, la propia estructura de membranas (debido a la operación de disolución) y los objetos contenidos en cada una de las membranas. Por tanto, los elementos básicos para describir una configuración de un P sistema en un instante de su evolución serán las estructuras de membranas y los objetos. Intuitivamente, las configuraciones contienen una descripción completa del estado actual de la ejecución; por ello, van a jugar un papel fundamental en el análisis y discusión de las computaciones de los P sistemas.

Definición 6.10 *Una* configuración *de un P sistema de transición,* $\Pi = (A, C_0, \mathcal{R}, i_0)$, *con* $C_0 = (\mu_0, M_0)$, *es una célula* $C = (\mu, M)$ *sobre* A, *donde* $V(\mu) \subseteq V(\mu_0)$, *y de tal manera que los árboles* μ *y* μ_0 *tienen la misma raíz. Se tiene que* C_0 *es, en particular, una configuración de* Π, *denominada* configuración inicial.

A continuación estudiamos cómo decidir si una regla puede ser aplicada o no en un instante determinado. Para ello, recordemos la interpretación informal de una regla (de evolución), $r \in R_x$, asociada a una membrana x. Supongamos que la regla r tiene la siguiente estructura sintáctica

$$a_1 \dots a_m \rightarrow (b_1, in_{j_1}) \dots (b_n, in_{j_n})(c_1, out) \dots (c_k, out)(h_1, here) \dots (h_l, here)s$$

con $\{\{j_1, \dots, j_n\}\}$ multiconjunto sobre $V(\mu)$, los elementos a_i, b_i, c_i, h_i son símbolos de A y $s = \delta$ ó $s = \lambda$.

Entonces

- Esta regla puede ser aplicada sólo cuando hay suficientes copias de los elementos $a_1, \dots, a_m$ en la membrana x y, además, ninguna regla de R_x con prioridad superior es aplicable.
- El resultado de usar está regla viene determinado por su lado derecho como sigue:
  - Los objetos $h_1, \dots, h_l$ van a parar a la misma región x.
  - Los objetos $c_1, \dots, c_k$ van a parar a la región inmediatamente exterior a x (es decir, al padre de x, en el caso en que x no sea la piel, y si x es la piel del P sistema, entonces los elementos se pierden en el entorno).

- Cada objeto b_s pasará a formar parte del multiconjunto asociado a la membrana j_s, en el caso en que j_s fuese un hijo de la membrana x. Si la membrana j_s no fuese hijo de x, entonces la regla no sería aplicable.
- Si el símbolo δ aparece en la regla (es decir, si $s = \delta$), entonces la membrana x se elimina (se *disuelve*) y, en consecuencia, el conjunto de reglas R_x (junto con su relación de prioridad asociada) no volverá a ser usado. El multiconjunto asociado a la membrana x se añade a la región inmediatamente exterior a ella (su padre). Por supuesto, una regla que contenga el símbolo δ no puede ser aplicada en la membrana piel.

Nota 6.11 Notaremos por $\mathbf{R} = \cup_{x \in V(\mu_0)} R_x$, al conjunto de todas las reglas de evolución asociadas a la configuración inicial (es decir, al P sistema de transición).

Definición 6.12 *Sea $C = (\mu, M)$ una configuración de un P sistema de transición $\Pi = (A, C_0, \mathcal{R}, i_0)$, con $C_0 = (\mu_0, M_0)$. Diremos que la regla (de evolución) $r = (d_r, v_r, \delta_r, x_r) \in \mathbf{R}$ es* aplicable *a C si:*

- *La membrana, x_r, a la que está asociada existe en la célula C; es decir, si $x_r \in V(\mu)$.*
- *No intenta disolver el nodo raíz; es decir, si x_r es el nodo raíz de μ, entonces $\delta_r = \neg\delta$.*
- *La membrana, x_r, a la que está asociada tiene los objetos necesarios para aplicar la regla; es decir, si $d_r \leq M(x_r)$.*
- *Los nodos a los que la regla intenta enviar objetos (por medio de in_j) son hijos de la membrana a la que pertenece la regla; es decir, si $\forall\, j \in V(\mu)(v_r(j) \neq \emptyset \rightarrow j \in Ch(x_r))$.*
- *No hay otras reglas aplicables a C con prioridad mayor; es decir, si*

$$\neg\exists\, r' \ (\rho_{x_r}(r', r) \wedge r' \text{ aplicable a } C)$$

Nota 6.13 Obsérvese que las reglas que intentan enviar objetos a la membrana j por medio de in_j nunca son aplicables. Es decir, si una regla r es aplicable a la configuración $C = (\mu, M)$, entonces $\forall\, j \in V(\mu)$ $(v_r(j) \neq 0 \rightarrow j$ no es raíz de $V(\mu))$.

A continuación, se define el *índice de aplicabilidad de una regla r en una configuración C*, que notaremos $N_{Ap}(r, C)$, y que nos va a indicar el máximo número de veces que la regla r puede ser aplicada a C. Es decir,

$$N_{Ap}(r,C) = \begin{cases} 0 & \text{, si } r \text{ no es aplicable a } C, \\ \text{máx}\{n \ : n \otimes d_r \leq M(x_r)\} & \text{, en otro caso.} \end{cases}$$

Téngase presente que una regla aplicable a una cierta configuración puede no ser aplicada en un instante concreto, debido al no determinismo del P sistema de transición.

A continuación, se introduce el concepto de *multiconjunto de aplicabilidad*, que va a representar un multiconjunto sobre el conjunto $\mathbf{R}$ de reglas que indica, con la multiplicidad asociada a cada regla, cuántas veces ha de aplicarse dicha regla a fin de realizar un paso de transición correcto en el P sistema.

Definición 6.14 *Sea $C = (\mu, M)$ una configuración de un P sistema de transición $\Pi = (A, C_0, \mathcal{R}, i_0)$. Diremos que un multiconjunto m sobre $\mathbf{R}$ es de aplicabilidad sobre C, y lo notaremos por $m \in \mathbf{M_{Ap}}(C)$, si :*

- $\forall r \in \mathbf{R}\ (m(r) \leq N_{Ap}(r,C))$.
- $\forall x \in V(\mu_0)\ (\sum_{r \in R_x} m(r) \otimes d_r \leq M(x))$.
- *Es maximal, en el sentido siguiente,* $\neg \exists\, m' \in \mathbf{M(R)}\ (m < m' \wedge m' \in \mathbf{M_{Ap}}(C))$.

Para determinar una configuración siguiente de una configuración dada, $C = (\mu, M)$, por un multiconjunto, m, de aplicabilidad sobre C, procederemos en dos etapas. En la primera, las reglas que se aplican en ese paso son ejecutadas sin atender a la disolución. En una segunda etapa se disuelve cada membrana/nodo que se tenga que disolver, se distribuye su contenido a su primer antecesor que no se haya disuelto, y se reestructuran las conexiones en el árbol enraizado que resulta.

La primera etapa (aplicación de las reglas sin atender a la disolución) produce la configuración $C'' = (\mu, M'')$, en donde:

$$\begin{aligned} M''(x) = \ & M(x) - \sum_{r \in R_x} m(r) \otimes d_r + \sum_{r \in R_x} m(r) \otimes v_r(here) + \\ & \sum_{r \in R_{f(x)}} m(r) \otimes v_r(x) + \sum_{y \in Ch(x)} \sum_{r \in R_x} m(r) \otimes v_r(out) \end{aligned}$$

Para llevar a cabo la segunda etapa (disolución y distribución de los contenidos) necesitamos caracterizar, previamente, los nodos que deben disolverse por la ejecución de un multiconjunto de aplicabilidad.

Definición 6.15 *Sea $C = (\mu, M)$ una configuración de un P sistema de transición Π, y $m \in \mathbf{M_{Ap}}(C)$ un multiconjunto de aplicabilidad sobre C. Se define el conjunto:*

$$\Delta(m, C) = \{x_r : r \in \mathbf{R} \wedge m(r) > 0 \wedge \delta_r = \delta\}$$

Es decir, $\Delta(m, C)$ representa el conjunto de nodos del árbol enraizado, μ, determinado por la configuración C que deben ser disueltos tras la aplicación/ejecución del multiconjunto m.

Hay que tener presente que en un paso del P sistema de transición, debido al paralelismo, se puede disolver más de una membrana de la estructura. Por tanto, será conveniente determinar para cada membrana, x, que permanezca en la próxima configuración, cuál es el conjunto de membranas que se disuelven por la aplicación de un cierto multiconjunto de aplicabilidad y añaden su contenido a x (tales membranas se llamarán *donantes* de x).

Definición 6.16 *Sea $C = (\mu, M)$ una configuración de un P sistema de transición Π. Sea $m \in \mathbf{M_{Ap}}(C)$ un multiconjunto de aplicabilidad sobre C. Para cada nodo $x \in V(\mu)$, se define los* donantes *de x para C por la aplicación de m, como sigue:*

$$Don(x, m, C) = \begin{cases} \emptyset & , si\ x \in \Delta(m, C) \\ \{y \in V(\mu) : y \in \Delta(m, C) \wedge x \rightsquigarrow_\mu y \wedge \\ \quad \wedge \forall z \in V(\mu)(x \rightsquigarrow_\mu z \rightsquigarrow_\mu y \rightarrow z \in \Delta(P, C))\} & , si\ x \notin \Delta(m, C) \end{cases}$$

Es decir, si un nodo x no se disuelve, entonces el nodo y es donante de x si y se disuelve por la aplicación de m, así como todo nodo que sea descendiente propio de x y antecesor propio de y.

A continuación se define la ejecución de un multiconjunto de aplicabilidad sobre una configuración dada.

Definición 6.17 *Sea $C = (\mu, M)$ una configuración de un P sistema de transición Π. Sea $m \in \mathbf{M_{Ap}}(C)$ un multiconjunto de aplicabilidad sobre C. Se define la* ejecución *de m sobre C, y se notará por $m(C)$, como la configuración $C' = (\mu', M')$ de Π, en donde:*

- *$\mu' = (V(\mu'), E(\mu'))$ es el árbol enraizado obtenido de μ como sigue:*
  - *$V(\mu') = V(\mu) - \Delta(m, C)$*
  - *Si $x, y \in V(\mu')$, entonces:*
    $(x, y) \in E^*(\mu') \Leftrightarrow \exists\, x_0, \ldots, x_n \in V(\mu)(x_1, \ldots, x_{n-1} \in \Delta(m, C) \wedge x_0 = x \wedge$
    $x_n = y \wedge \forall i\ (0 \leq i < n \rightarrow (x_i, x_{i+1}) \in E^*(\mu)))$

- $M'(x) = \begin{cases} M''(x) \cup \bigcup_{y \in Don(x,m,C)} M''(y) & , \text{ if } x \notin \Delta(m,C) \\ \emptyset & , \text{ if } x \in \Delta(m,C) \end{cases}$

Donde, recordemos, $E^*(\mu)$ es el direccionamiento inducido de manera natural sobre las aristas del árbol enraizado $\mu = (V(\mu), E(\mu))$ al distinguir un nodo como raíz (véase el apartado **6.2** de esta memoria).

Es decir, μ' es el árbol enraizado obtenido de μ eliminando los nodos que se disuelven por la aplicación del multiconjunto m, y restaurando las conexiones en la forma correcta. Si un nodo y es disuelto, entonces el nodo se elimina, junto con todas las aristas adyacentes a él, y se añade una arista entre su primer antecesor no disuelto y los primeros descendientes de y no disueltos. Si un nodo no se disuelve, entonces debemos añadir a su contenido el contenido de todos sus donantes por la aplicación de m.

En la figura siguiente se muestra un ejemplo del árbol de membranas resultante tras haber disuelto un conjunto de nodos (en este caso, $\Delta(m,C) = \{2,4,7,9\}$).

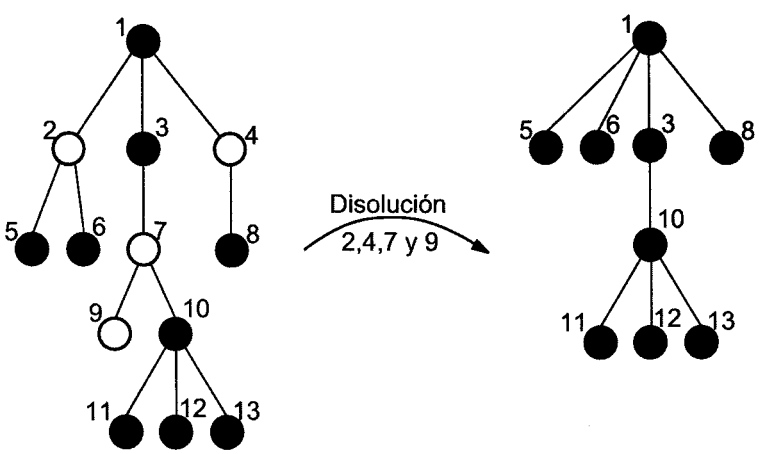


Figura 6.7. Ejemplo de disolución.

A continuación, formalizamos la ejecución de un paso en un P sistema; es decir, precisamos cómo se produce la transición de una configuración del P sistema a otra configuración tras ejecutar un paso. Se trata, pues, de precisar cómo calcula, cómo evoluciona un P sistema a lo largo del tiempo.

Definición 6.18 *Se dice que una configuración C_2 de un P sistema de transición, Π, deriva de una configuración C_1 por una* transición en un paso *de Π (o tras la ejecución de un paso), y se nota por $C_1 \Rightarrow_\Pi C_2$, si existe un multiconjunto, m, de aplicabilidad sobre C_1 tal que $m \neq \emptyset$ y $m(C_1) = C_2$.*

Intuitivamente, $C_1 \Rightarrow_\Pi C_2$ significa que en un paso de computación del P sistema Π, de la configuración C_1 resulta la configuración C_2.

Una vez definida la relación de *transición en un paso* entre configuraciones de un P sistema, se define la relación de *transición* como su clausura transitiva.

Definición 6.19 *Sean C y C' dos configuraciones de un P sistema Π y $k \in \mathbf{N}$. Diremos que la configuración C' se obtiene de la configuración C en k pasos de transición (es decir, al ejecutarse k pasos), si existen configuraciones $C_1, \ldots, C_{k+1}$ tales que*

$$C_1 = C \wedge C_{k+1} = C' \wedge \forall i \ (1 \leq i \leq k \rightarrow C_i \Rightarrow_\Pi C_{i+1})$$

Finalmente, se dice que una configuración C' se obtiene de una configuración C en una computación de Π, y se nota por $C \stackrel{*}{\Rightarrow}_\Pi C'$, si existe $k \in \mathbf{N}$ tal que C' se obtiene de C en k pasos de transición de Π.

A partir de la configuración inicial de un P sistema Π, se puede construir el árbol de computación asociado a Π, de tal forma que los nodos de dicho árbol son las configuraciones que aparecen a lo largo de la computación, y las aristas son los multiconjuntos de aplicabilidad que transforman un nodo en otro.

Definición 6.20 *El* árbol de computación *de un P sistema de transición, Π, que notaremos por $\mathbf{T}_\Pi$, es el árbol enraizado etiquetado maximal definido como sigue:*

- *la raíz del árbol es la configuración inicial, C_0, de Π;*
- *los hijos de un nodo son las configuraciones que se obtienen a partir de él por un paso de transición;*
- *los nodos y las aristas están etiquetadas por configuraciones y multiconjuntos de aplicabilidad, respectivamente, de tal forma que dos nodos etiquetados, C y C', son adyacentes en $\mathbf{T}_\Pi$, por medio de una arista etiquetada por m, si y sólo si $m \in \mathbf{M_{Ap}}(C) - \{\emptyset\} \wedge C' = m(C)$.*

Las ramas maximales de $\mathbf{T}_\Pi$ se denominan computaciones *de Π, y al conjunto de computaciones de Π lo notaremos por* $\mathbf{Comp}(\Pi)$. *Diremos que una computación de Π es de* parada *si es una rama finita de $\mathbf{T}_\Pi$. Las configuraciones que verifican $\mathbf{M_{Ap}}(C) = \{\emptyset\}$ se denominan* configuraciones de parada, *y denotaremos el conjunto de configuraciones de parada por $Halt(\Pi)$.*

Es decir, una computación, $\mathcal{C}$, de un P sistema de transición, $\Pi = (A, C_0, \mathcal{R}, i_0)$, es una sucesión (posiblemente infinita) de configuraciones del sistema $C_0 \Rightarrow_\Pi C_1 \Rightarrow_\Pi$

$\ldots \Rightarrow_\Pi C_r$ (con $r \in \mathbf{N} \cup \{\infty\}$) tal que C_0 es la configuración inicial de Π y, además, $\forall i$ $(i < r \rightarrow C_i \Rightarrow_\Pi C_{i+1})$. En tal situación, diremos que r es la longitud de $\mathcal{C}$ y notaremos $|\mathcal{C}| = r$. Además, si $r \in \mathbf{N}$ entonces diremos que C_r es la *configuración final* de la computación $\mathcal{C}$.

Notaremos por $\mathbf{Comp}_p(\Pi)$ al conjunto de *computaciones parciales* de Π; es decir, cualquier sucesión de la forma $C_0 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_n$, con $n \in \mathbf{N}$.

Habitualmente, si $\mathcal{C} \in \mathbf{Comp}(\Pi)$, entonces notaremos $\mathcal{C} \equiv C_0 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_n \Rightarrow_\Pi \ldots$; es decir, C_k representa la configuración obtenida tras ejecutar k pasos en la computación $\mathcal{C}$ de Π.

Definición 6.21 *Sea $\Pi = (A, C_0, \mathcal{R}, i_0)$ un P sistema de transición. Diremos que una computación, $\mathcal{C}$, de Π, es* exitosa *si $|\mathcal{C}| = n < \infty$ e i_0 es una hoja del árbol enraizado μ_n, donde $C_n = (\mu_n, M_n)$ es la configuración final asociada a $\mathcal{C}$. En este caso, diremos que C_n es una* configuración exitosa *de Π.*

Es decir, una computación es exitosa si es de parada y, además, la membrana de salida es elemental en la configuración final de dicha computación. Es claro que podemos construir P sistemas de transición en los que, dada una configuración concreta, existen distintos multiconjuntos de aplicabilidad para dicha configuración. En ese momento de la computación, el sistema tendrá varias posibilidades para continuar, apareciendo el no determinismo al que se ha hecho mención con anterioridad.

Definición 6.22 *Sea Π un P sistema de transición. Diremos que Π es un P sistema* determinista *si para cada configuración C de Π se tiene que $\mathbf{M_{Ap}}(C)$ es un conjunto unitario. Diremos que el P sistema Π es* no determinista *si existe una configuración, C, de Π tal que $|\mathbf{M_{Ap}}(C)| > 1$.*

En un P sistema de transición no determinista existen configuraciones que poseen más de una configuración siguiente. Por ello, estos P sistemas pueden evolucionar de acuerdo a varias posibilidades.

6.3. Tipos de P sistemas de transición

En esta sección se va a poner de manifiesto la versatilidad de los P sistemas de transición. Concretamente, se va a ver que los P sistemas de transición pueden ser considerados como dispositivos *generadores* de multiconjuntos, de números o de relaciones. Es también posible interpretar los P sistemas como dispositivos *reconocedores* de multiconjuntos, o incluso como máquinas *de calcular* funciones parciales de $\mathbf{N}$ en $\mathcal{P}(\mathbf{N})$.

6.3.1. P sistemas de transición generadores

Vamos a ver en primer lugar que los P sistemas de transición pueden ser considerados como dispositivos generadores de multiconjuntos, de números o de relaciones.

Definición 6.23 *Sea* $\Pi = (A, C_0, \mathcal{R}, i_0)$ *un P sistema de transición. Notaremos* $S(\Pi) = \{\mathcal{C} : \mathcal{C}$ *es computación exitosa de* $\Pi\}$. *Si* $\mathcal{C}$ *es una computación exitosa de* Π *de longitud* n, *entonces se define la salida de* $\mathcal{C}$, *que notaremos* $\mathcal{O}(\mathcal{C})$, *así:* $\mathcal{O}(\mathcal{C}) = M_{C_n}(i_0)$. *La salida generada por* Π, *que notaremos* $\mathcal{O}(\Pi)$, *es la siguiente colección de multiconjuntos:*

$$\mathcal{O}(\Pi) = \{\mathcal{O}(\mathcal{C}) : \mathcal{C} \in S(\Pi)\}$$

Es decir, $\mathcal{O}(\Pi)$ representa la colección de todos los multiconjuntos sobre A que aparecen en la membrana de salida de alguna configuración exitosa del P sistema de transición Π.

Nota 6.24 En [61], se consideran P sistemas con salida en el *entorno* (esto es, el multiconjunto de objetos que son enviados al exterior durante la computación). La formalización anterior puede ser fácilmente adaptada a este nuevo caso. Para ello, bastaría

- cambiar la definición 6.9, permitiendo, por ejemplo, que la membrana de salida sea el entorno; es decir, exigiendo que $i_0 = env \notin V(\mu_0)$;
- extender el concepto de célula, permitiendo que la segunda componente, M, sea una aplicación de $V(\mu) \cup \{env\}$ en $\mathbf{M}(A)$;
- interpretar $M(env)$ como el contenido del entorno;
- eliminar la última condición en la definición 6.21.

Definición 6.25 *Sea* $\Pi = (A, C_0, \mathcal{R}, i_0)$ *un P sistema de transición. El conjunto de los números naturales generados por* Π, *que notaremos* $\mathbf{N}(\Pi)$, *se define como sigue:*

$$\mathbf{N}(\Pi) = \{|\mathcal{O}(\mathcal{C})| : \mathcal{C} \in S(\Pi)\}$$

Es decir, $\mathbf{N}(\Pi)$ representa el conjunto de los tamaños de todos los multiconjuntos sobre A que aparecen en la membrana de salida de alguna configuración exitosa del P sistema Π.

Definición 6.26 *Sea* $\Pi = (A, C_0, \mathcal{R}, i_0)$ *un P sistema de transición. Sean* $a_1, \ldots, a_k$ *elementos distinguidos de* A *y distintos entre sí. La relación* k*–aria generada por* Π *asociada a la* k*–tupla* $(a_1, \ldots, a_k)$*, que notaremos* $R_k(\Pi)$*, está definida como sigue:*

$$R_k(\Pi) = \{(n_1, \ldots, n_k) \in \mathbf{N}^k : \exists \mathcal{C} \in S(\Pi)\ \forall j\ (1 \leq j \leq k \rightarrow (\mathcal{O}(\mathcal{C}))(a_j) = n_j)\}$$

Es decir, $R_k(\Pi)$ es el conjunto de todas las k–tuplas $(n_1, \ldots, n_k)$ de números naturales tales que en la membrana de salida de alguna configuración exitosa del P sistema Π, los elementos distinguidos $(a_1, \ldots, a_k)$ tienen multiplicidades $(n_1, \ldots, n_k)$, respectivamente.

6.3.2. P sistemas de transición reconocedores

Los P sistemas de transición también pueden ser considerados como dispositivos reconocedores de cadenas (que codifican multiconjuntos, números o relaciones). Un P sistema puede ser considerado como una máquina de reconocimiento para un predicado $\theta(n_1, \ldots, n_k)$ sobre $\mathbf{N}$, si se imponen las siguientes condiciones:

- El P sistema debe tener dos membranas distinguidas: una de *entrada* y otra de *salida.*
- El P sistema debe ser *coherente*, en el sentido de que todas las computaciones deben devolver la misma respuesta para un dato de entrada prefijado.
- El P sistema debe tener un procedimiento, o mecanismo, de *aceptación*; por ejemplo, a través de un predicado asociado a la salida del P sistema.
- La k-tupla $(n_1, \ldots, n_k)$ debe ser codificada en la membrana de entrada; por ejemplo, a través de k elementos distinguidos, $(a_1, \ldots, a_k)$, del alfabeto base y exigiendo que n_j sea la multiplicidad de a_j en la membrana de entrada de la configuración inicial.
- Cualquier computación posible del P sistema debe ser exitosa.

Definición 6.27 *Un P sistema de transición* reconocedor *de orden* k *es una 7-tupla,* $\Pi = (A, B, C_0, \mathcal{R}, i_0, j_0, \varphi)$*, tal que*

- *La 4-tupla* $\Pi' = (A, C_0, \mathcal{R}, i_0)$ *es un P sistema de transición tal que toda computación de* Π' *sea exitosa.*
- B *es un subconjunto ordenado de* A *de cardinal* k*, que representa los elementos distinguidos del alfabeto base con los que se codifica el dato de entrada.*

- j_0 *es un nodo de* μ_0 *que especifica la membrana de entrada de* Π.
- φ *es un predicado sobre* $\mathbf{M}(A)$.

Definición 6.28 *Diremos que un P sistema de transición reconocedor de orden* k, $\Pi = (A, B, C_0, \mathcal{R}, i_0, j_0, \varphi)$, acepta $(n_1, \ldots, n_k) \in \mathbf{N}^k$ *si*

- $\forall t\ (1 \leq t \leq k \rightarrow (M_{C_0}(j_0))(a_t) = n_t)$, *con* $B = \{a_1, \ldots, a_k\}$; *es decir, la membrana de entrada de la configuración inicial del P sistema codifica la* k*-tupla* $(n_1, \ldots, n_k)$.
- *La fórmula* $\varphi(\mathcal{O}(\mathcal{C}))$ *es verdadera, para cualquier computación,* $\mathcal{C}$, *del P sistema; es decir, la membrana de salida de la configuración final de* $\mathcal{C}$ *verifica el predicado* φ.

Definición 6.29 *Diremos que un P sistema de transición reconocedor de orden* k, $\Pi = (A, B, C_0, \mathcal{R}, i_0, j_0, \varphi)$, decide *el predicado* k*–ario* $\theta(n_1, \ldots, n_k)$ *sobre* $\mathbf{N}$ *si para cada* $(n_1, \ldots, n_k) \in \mathbf{N}^k$ *se verifica que:*

$$\theta(n_1, \ldots, n_k) \Leftrightarrow \Pi \text{ acepta } (n_1, \ldots, n_k)$$

Obsérvese que un predicado k-ario sobre $\mathbf{N}$ no es decidido, propiamente, por un P sistema de transición, sino por una colección de P sistemas *similares*; es decir, P sistemas en los que únicamente varía el multiconjunto de objetos de la membrana de entrada.

6.3.3. P sistemas de transición como máquinas de cálculo

Los P sistemas de transición también pueden ser interpretados como máquinas que calculan funciones parciales tales que a números naturales asocia conjuntos de números naturales. Para ello, podemos considerar un P sistema de transición que calcule una función parcial, $f : \mathbf{N}^k - \rightarrow P(\mathbf{N})$, exigiendo las siguientes condiciones:

- El P sistema debe tener dos membranas distinguidas: una de *entrada* y otra de *salida*.
- La k-tupla $(n_1, \ldots, n_k) \in \mathbf{N}^k$ debe ser codificada en la membrana de entrada; por ejemplo, siguiendo un método similar al del apartado anterior.

En esta situación, la salida de la aplicación será codificada por la colección de los multiconjuntos que aparecen en la membrana de salida de las computaciones exitosas del P sistema.

Definición 6.30 *Un P sistema de transición* de cálculo *de orden k es una 6-tupla*

$$\Pi = (A, B, C_0, \mathcal{R}, i_0, j_0)$$

tal que

- *La 4-tupla $\Pi' = (A, C_0, \mathcal{R}, i_0)$ es un P sistema de transición.*
- *B es un subconjunto ordenado de A de cardinal k, que representa los elementos distinguidos del alfabeto base con los que se codifica el dato de entrada.*
- *j_0 es un nodo de μ_0 que especifica la membrana de entrada de Π.*

Definición 6.31 *Diremos que un P sistema de transición de cálculo de orden k, $\Pi = (A, B, C_0, \mathcal{R}, i_0, j_0)$,* calcula *la función parcial $f : \mathbf{N}^k - \rightarrow \mathbf{N}$ si para cada $(n_1, \ldots, n_k) \in \mathbf{N}^k$ se tiene que*

- *$\forall t\ (1 \leq t \leq k \rightarrow (M_{C_0}(j_0))(a_t) = n_t)$, con $B = \{a_1, \ldots, a_k\}$; es decir, la membrana de entrada de la configuración inicial del P sistema codifica la k-tupla $(n_1, \ldots, n_k) \in \mathbf{N}^k$.*
- *Para cada $(n_1, \ldots, n_k) \in \mathbf{N}^k$, se tiene que $f(n_1, \ldots, n_k)$ converge si y sólo si el P sistema con entrada $(n_1, \ldots, n_k)$ para y, además,*

$$f(n_1, \ldots, n_k) = \{|M_C(i_0)| : C \in S(\Pi') \wedge C = (\mu_C, M_C)\}$$

De nuevo, obsérvese que una función parcial, $f : \mathbf{N}^k - \rightarrow P(\mathbf{N})$, no es calculada propiamente por un único P sistema sino por una colección de P sistemas *similares*; es decir, P sistemas en los que únicamente varía el multiconjunto de objetos de la membrana de entrada.

Usando elementos distinguidos para la membrana de salida se pueden definir, de forma similar, P sistemas de transición que calculan funciones parciales de $\mathbf{N}^k$ en $P(\mathbf{N}^j)$.

6.4. Universalidad de los P sistemas de transición

Los P sistemas de transición son unos dispositivos con una estructura bastante simple a pesar de lo cual son computacionalmente completos en el sentido de que pueden generar todos los conjuntos recursivamente enumerables de números naturales.

La mayoría de los resultados obtenidos hasta ahora acerca de la potencia computacional de los P sistemas de transición se han presentado módulo la *función de Parikh.*

Definición 6.32 *Sea $A = \{a_1, \ldots, a_k\}$ un alfabeto. La función de Parikh asociada a A es la aplicación $\psi_A : \mathbf{M}(A) \rightarrow \mathbf{N}^k$ definida por $\psi_A(m) = (m(a_1), \ldots, m(a_k))$, para cada $m \in \mathbf{M}(A)$. Diremos que $\psi_A(m)$ es el* vector de Parikh *asociado al multiconjunto m.*

Definición 6.33 *Para cada P sistema de transición, $\Pi = (A, C_0, \mathcal{R}, i_0)$, notaremos por $P_s(\Pi)$ el conjunto siguiente: $P_s(\Pi) = \psi_A(\mathcal{O}(\Pi))$. Esto es, $P_s(\Pi)$ es el conjunto de vectores de Parikh asociados a la salida del P sistema.*

Definición 6.34 *Para cada terna $(a, b, c) \in \{Pri, nPri\} \times \{Cat, nCat\} \times \{\delta, n\delta\}$, diremos que un P sistema de transición $\Pi = (A, C_0, \mathcal{R}, i_0)$ es de tipo (a, b, c) si se satisfacen las condiciones siguientes:*

- *Si $a = Pri$, entonces el P sistema usa relaciones de prioridad, esto es, $\rho \neq \emptyset$ (recuérdese que $\mathcal{R} = (R, \rho)$). En caso contrario, $a = nPri$.*
- *Si $b = Cat$, entonces el P sistema usa catalizadores. En caso contrario, se tendrá $b = nCat$.*
- *Si $c = \delta$, entonces el P sistema hace uso de la disolución. En caso contrario, $c = n\delta$.*

Notaremos por $NP_m(a, b, c) = \{P_s(\Pi) : \Pi$ es de tipo $(a, b, c) \wedge grad(\Pi) \leq m\}$.

Así, por ejemplo, si Π es de tipo $(Pri, nCat, n\delta)$, significará que el P sistema Π hace uso de relaciones de prioridad entre sus reglas, no hace uso de catalizadores y no hace uso de la disolución.

El siguiente teorema es el resultado básico de universalidad (módulo la función de Parikh) de los P sistemas de transición.

Teorema 6.35 $P_sRE = NP_2(Pri, Cat, n\delta) = NP_4(nPri, Cat, \delta)$

En donde RE es la familia de los lenguajes recursivamente enumerables.

Las pruebas de estas igualdades usan las técnicas habituales de la mayoría de los resultados que se han dado relativos a la universalidad de los P sistemas (véase [25] y [56]); esto es, por medio de *gramáticas matriciales con chequeo de apariciones*, una clase de gramáticas regulares libres de contexto que fue estudiada en la década de los sesenta (véase [21] y [80]).

Recientemente, A. Romero y M.J. Pérez en [74] han presentado otra vía para establecer la universalidad de los P sistemas de transición (en dicho artículo con la variante de P sistemas de transición con salida externa) a través de la simulación

de Máquinas de Turing en dichos sistemas, con la ventaja de probar la universalidad de los P sistemas generando directamente todos los conjuntos recursivamente enumerables, en vez de hacerlo módulo la función de Parikh, como ha sido lo usual hasta ahora.

6.5. La conjetura P=NP y los P sistemas de transición

De acuerdo con el teorema **6.35** resulta que los P sistemas de transición son computacionalmente completos; es decir, tienen la misma capacidad expresiva que las máquinas de Turing. Ahora bien, desde el punto de vista de la computabilidad práctica, los P sistemas de transición carecen, en determinados casos, de la potencia suficiente para poder resolver problemas computacionalmente intratables con una mejora cualitativa de los costes empleados.

C. Zandron, C. Ferreti y G. Mauri en [85] proporcionan un resultado que viene a ratificar este hecho.

Teorema 6.36 *Sea* Π *un P sistema de transición determinista que trabaja en un tiempo* t*. Denotemos por* A*,* B *y* C*, respectivamente, el número de membranas, el número de símbolos del alfabeto y la longitud de las reglas (es decir, el número total de símbolos que intervienen en las reglas, considerando ambas partes) del P sistema. Sea* $D = \text{máx}\{C, B+2\}$*. Entonces el P sistema* Π *puede ser simulado por una máquina de Turing determinista en tiempo* $t' = O(A \cdot B \cdot D \cdot t \cdot \log(A \cdot B \cdot C^t))$.

De este resultado se deduce una consecuencia importante. Supongamos que dado un problema, X, para cada instancia de tamaño n se puede construir un P sistema de transición determinista, con un número de membranas, un número de objetos y un número de reglas que son polinomiales en n. En tal situación, dicho P sistema puede ser simulado por una máquina de Turing determinista con coste en tiempo igual al del P sistema que simula, afectado de un factor polinomial en n.

Por tanto, del teorema de Zandron-Ferreti-Mauri se deduce lo siguiente: si existe *un* problema **NP**-completo que es resoluble en tiempo polinomial por un P sistema de transición determinista, cuyo número de membranas, número de objetos y número de reglas son polinomiales en el tamaño del dato de entrada del problema, entonces se verificará que **P** = **NP**. Ahora bien, ¿qué se puede deducir en el caso de que *no* exista un tal problema **NP**-completo?

En el trabajo de A. Romero y M.J. Pérez [74] se prueba que toda máquina de Turing determinista puede ser simulada por un P sistema de transición determinista con salida externa. Se puede probar que dicha simulación usa un tiempo polinomial y el P sistema consta de un número de membranas, número de objetos y de reglas que es polinomial en la instancia del problema.

Por tanto, si existe *un* problema **NP**-completo que no es resoluble en tiempo polinomial por un P sistema de transición con un número de membranas y de reglas polinomial en la entrada, entonces se tiene que $\mathbf{P} \neq \mathbf{NP}$.

6.6. Algunas variantes de los P sistemas de transición

Las limitaciones que ofrecen los P sistemas de transición deterministas, desde el punto de vista de la computabilidad práctica, han propiciado la aparición de diversos trabajos en los que se introducen modificaciones en los P sistemas que, sin abandonar lo que podríamos denominar la *inspiración biológica*, permiten resolver en tiempo polinomial y de modo determinista, problemas que son considerados computacionalmente intratables en los modelos convencionales de computación. Dichas modificaciones se denominan genéricamente *variantes mejoradas* y se pueden clasificar, básicamente, en tres grupos:

- Las que permiten el uso de objetos estructuralmente más complejos, en los que las reglas no transforman únicamente unos objetos en otros o los comunican entre membranas, sino que pueden modificar la estructura interna de los mismos.

- Las que modifican la estructura de membranas, permitiendo no tan solo la disolución, sino la creación o duplicación (división) de las mismas.

- Las que están orientadas al análisis del papel que juega la comunicación entre distintas membranas, y que tratan de estudiar los requisitos mínimos necesarios para conseguir un P sistema universal.

A continuación hacemos un breve análisis de algunas de las variantes con las que se trabaja en la actualidad.

6.6.1. Variantes en los objetos

En los P sistemas de transición vistos hasta ahora puede interpretarse que no se hace uso de estructuras de datos para codificar la información: los números son codificados como la cardinalidad de multiconjuntos, por consiguiente, están representados en base 1.

Esto puede ser adecuado para una implementación bioquímica, pero parece claramente insuficiente desde un punto de vista clásico. Más aún, por este camino sólo podrá tratarse (directamente, sin una codificación numérica) con problemas sobre números no con computaciones simbólicas. Por ello, se trata de usar sistemas que representen la información a través de una estructura de datos de tipo estándar: las cadenas.

De esta manera, en lugar de considerar objetos de un tipo atómico (sin *partes*), se usarán objetos que pueden ser descritos por cadenas finitas sobre un alfabeto finito prefijado. En tal situación, la evolución de un objeto correspondería propiamente a una transformación de la cadena.

P sistemas con reescritura

En estos P sistemas se procesan las cadenas de objetos a través de reglas del tipo $r \equiv X \rightarrow (v; tar)$, donde X es un símbolo, v es una cadena de símbolos y tar es la membrana destino de dicha cadena (eventualmente, la cadena v puede ir seguida de un símbolo δ, de disolución). La forma de considerar la ejecución de este tipo de reglas (que actúan en el *interior* de las cadenas) es la habitual en P sistemas (la aplicación de r sobre una cadena w reemplaza una ocurrencia de X por la cadena v, disolviendo la membrana si así se indica en r), con la siguiente puntualización: todas las cadenas son procesadas en paralelo, pero sobre cada cadena presente en la membrana sólo actúa una regla [38].

Teniendo presente que la generación natural de esta variante de P sistemas es un multiconjunto de cadenas, los resultados acerca de la universalidad y la capacidad generativa de los mismos no precisa considerar equivalencias módulo la función de Parikh. En este sentido, si denotamos por $RP_m(a,b)$, con $(a,b) \in \{Pri, nPri\} \times \{\delta, n\delta\}$, la familia de lenguajes generados por esta variante de P sistemas, de grado menor o igual a m, según las características usadas, se tiene el siguiente resultado de universalidad:

Teorema 6.37 $RE = RP_2(Pri, n\delta)$

Es decir, en este tipo de P sistemas es suficiente usar 2 membranas y reglas que no

usen disolución (aunque sí prioridades entre las mismas) para generar directamente la familia de lenguajes recursivamente enumerables.

P sistemas con replicación de cadenas

Este tipo de P sistemas es una extensión del procedimiento de reescritura antes descrito, ya que las reglas combinan la posibilidad de actuar en el interior de las cadenas con la capacidad de replicación de las mismas [39]. De esta forma, una regla adquiere la expresión general $r \equiv X \rightarrow (u_1, tar_1)||\ldots||(u_n, tar_n)$, y su aplicación sobre una cadena w produce la aparición de n cadenas, $w_1, \ldots, w_n$, de modo que w_i se obtiene de w reemplazando una aparición de X por u_i, y su destino es tar_i.

Si denotamos RRP_m la familia de lenguajes generados por esta variante de P sistemas, de grado menor o igual a m, sin hacer uso de prioridades entre las reglas, ni de disolución, entonces se tiene el siguiente resultado de universalidad:

Teorema 6.38 $RE = RRP_6$

Es decir, que con este tipo de P sistemas, bastan 6 membranas para generar directamente la familia de los conjuntos recursivamente enumerables, sin hacer uso de disoluciones ni prioridades.

Esta variante de los P sistemas de transición ha sido usada para dar soluciones deterministas, y con coste en tiempo polinomial, de problemas **NP**-completos clásicos, como son el problema **SAT** y el problema del camino hamiltoniano [47]. No obstante, la parte exponencial del proceso queda encubierto en la aplicación de reglas que permiten la replicación de las cadenas.

6.6.2. Variantes en las membranas

Otra posibilidad de mejorar las características de los P sistemas es añadir funcionalidad y dinamismo a la estructura de membranas, permitiendo la creación y división de membranas por medio de reglas.

División de Membranas

Este tipo de variante es conocida por el nombre de *P sistemas con membranas activas*. Se trata de un dispositivo útil para conseguir una cantidad exponencial de espacio partiendo de una estructura de membranas de grado polinomial (en función del dato de entrada que se considere). De esta forma, se pueden resolver proble-

mas **NP**-completos, de manera determinista, en un número polinomial de pasos, sacrificando el espacio por el tiempo.

En esta variante se permite el uso de reglas en las que su aplicación genera la división (análoga a la producida en la naturaleza) de membranas elementales (siempre que no sea la piel) en dos nuevas membranas que contienen un duplicado del contenido de la membrana original (salvo el multiconjunto de objetos que dispara la regla, que pueden ser transformados en multiconjuntos distintos en las nuevas membranas), y el mismo conjunto de reglas.

También en esta variante se consiguen resultados de universalidad similares a los vistos anteriormente, modulo la función de Parikh.

Si en esta variante se permite la división de membranas no elementales (y el P sistema se denomina *extendido*), entonces se puede probar el siguiente resultado [54]:

Teorema 6.39 *El problema del camino hamiltoniano puede ser resuelto en tiempo cuadrático y el problema SAT en tiempo lineal, por medio de P sistemas extendidos con membranas activas.*

Este resultado ha sido mejorado por C. Zandron y otros en [85], resolviendo dichos problemas en tiempo lineal a través de P sistemas que hacen uso de membranas activas únicamente en las membranas elementales.

Creación de membranas

La creación de membranas es un proceso relativamente habitual en la biología. En relación con los P sistemas, se considera la posibilidad de crear nuevas membranas por la acción de los objetos existentes. En este sentido, se pueden introducir reglas en las que un determinado multiconjunto de objetos producen, en el interior de la membrana a la que pertenecen, una nueva membrana con un cierto contenido. Este tipo de creación de membranas puede ser aplicado tanto a P sistemas que usan objetos atómicos como a aquellos que usan objetos de tipo cadena. Su principal interés radica en que proporciona otra forma de obtener una cantidad exponencial de espacio en tiempo de ejecución polinomial.

Esta variante de los P sistemas de transición fue considerada por M. Ito y otros en [33] para conseguir resultados de universalidad en los que se permitía un uso bastante moderado de esta nueva operación. También puede ser usada para la resolución en tiempo polinomial de problemas **NP**-completos, en el que disponer de una cantidad exponencial de espacio puede ser de gran interés (por ejemplo, el problema **SAT** de

la lógica proposicional o el problema **HPP** del camino hamiltoniano).

6.6.3. Variantes en las comunicaciones

Hay un tercer tipo de variantes que no pretenden introducir la mejora de un uso exponencial en la cantidad de espacio útil en los P sistemas, sino que tratan de estudiar el papel que juega la comunicación entre las membranas (por medio de los elementos que son transferidos de unas a otras).

Lo habitual es que estos modelos no presenten una mejora propiamente dicha, sino un efecto contrario; es decir, se intenta debilitar alguna capacidad inherente a los P sistemas de transición con el fin de estudiar cuáles son las características mínimas necesarias para seguir disponiendo de un modelo de computación universal (algo que podría ser útil si algún día se consigue una implementación práctica de estos modelos).

Una primera variante consiste en cambiar el destino in_j en las reglas de los P sistemas por un destino más general, in, que no especifique a qué membrana han de ir los objetos producidos, de modo que dicha elección es no determinista entre las posibles membranas existentes que sean hijas de la membrana asociada a la regla. Se debe hacer notar que muchos de los resultados de universalidad obtenidos para variantes mejoradas se siguen manteniendo para esta variante débil de P sistemas, debido a que en dichas pruebas se hace uso de estructuras de membranas de grado 2, en las que no es necesario especificar qué membrana hija es la que recibe los objetos.

Otra variante introducida en esta dirección consiste en dotar de una carga eléctrica a los objetos y membranas del P sistema (con tres cargas posibles: $+, -, 0$), de tal modo que *a priori* se elige una carga en las membranas (en la definición del P sistema) y los objetos pueden modificar sus cargas de acuerdo con las reglas de evolución que se aplican sobre ellos. Así, la relación entre las cargas de los objetos y las membranas determina si un objeto puede atravesar una membrana.

Teniendo presente que las membranas biológicas tienen una permeabilidad que es variable, se podría controlar el paso de objetos de una membrana a otra introduciendo un parámetro de *permeabilidad* de una membrana, como valor numérico que indica su proximidad o cercanía a un estado que permite su disolución. En dichos sistemas, se puede modificar dicha permeabilidad por medio de reglas, de tal manera que la ejecución de una regla puede hacer que la membrana engrose o adelgace. Si una membrana tiene un grosor muy elevado, se vuelve *impermeable*, impidiendo la transferencia de objetos a través de ella. Así, por ejemplo, el control de la permeabilidad de las membranas se puede implementar introduciendo una acción, que

notaremos τ, y de tal manera que es opuesta a δ, en el sentido de que *engrosa* la membrana, la aleja de la disolución. Se puede demostrar que bastan tres niveles de permeabilidad para estudiar el comportamiento de esta variante de los P sistemas.

En los *P sistemas con carriers* [47], los objetos no son transformados, sino que únicamente son transmitidos entre las membranas de acuerdo con ciertas reglas de transferencia. Usualmente, en esta variante se permite la transmisión con objetos del *entorno* (es decir, del exterior del P sistema), con el fin de poder disponer de una cantidad ilimitada de los mismos.

Una variante interesante de los P sistemas con carriers son los P sistemas con *porters* [55], en los que se restringe la capacidad en la transmisión de objetos, reduciendo el radio de las reglas.

Capítulo 7

Verificación formal en P sistemas

En el artículo fundacional de la computación con membranas [56], Gh. Păun ilustra el funcionamiento de los P sistemas con varios ejemplos que reflejan las características principales de este modelo, y que muestran la versatilidad de los mismos para ser interpretados como generadores de lenguajes, como dispositivos reconocedores y como máquinas de calcular funciones parciales. No obstante, en el artículo citado sólo se justifica de manera informal que los P sistemas presentados resuelven realmente los problemas planteados.

En este capítulo nos centraremos en establecer la verificación de los P sistemas allí introducidos (con pequeñas modificaciones) como ilustración de algunas técnicas que pueden resultar útiles para abordar, con cierto éxito, la verificación formal de P sistemas arbitrarios. Además, en este estudio se ponen de manifiesto las dificultades que surgen a la hora de sistematizar los procesos de verificación debido principalmente al paralelismo distribuido al que se hacía mención en el capítulo anterior y al hecho de que este modelo de computación es, básicamente, de tipo procedural.

Sea Π un P sistema de transición diseñado para generar un conjunto, B, de números naturales. Para establecer la verificación de Π en relación al conjunto B lo habitual será buscar un predicado sobre el conjunto de computaciones de Π, que sea, de alguna forma, un invariante a lo largo de la evolución del P sistema. Además, de la veracidad del predicado sobre las computaciones de Π debe poder extraerse la información relevante a fin de establecer la corrección y completitud de Π en relación con la generación del conjunto B.

El proceso de verificación de un P sistema de transición, $\Pi = (A, C_0, \mathcal{R}, i_0)$, está basado en el análisis del contenido de cada membrana de todas las configuraciones que se pueden obtener en el sistema.

De forma natural, se puede definir una función parcial, **STEP**, del conjunto

$\mathbf{Comp}(\Pi) \times \mathbf{N} \times V(\mu_0)$ en $\mathbf{M}(A)$, que asigna a cada computación $\mathcal{C}$, de Π, a cada número natural, k, y a cada membrana, x, del P sistema, el contenido de la membrana x tras la ejecución de k pasos de la computación $\mathcal{C}$. Si, tras la ejecución del paso k-ésimo, la membrana x se ha disuelto, entonces $\mathbf{STEP}(\mathcal{C}, k, x)$ no está definida, en este caso, notaremos $\mathbf{STEP}(\mathcal{C}, k, x) \uparrow$. En caso contrario, notaremos $\mathbf{STEP}(\mathcal{C}, k, x) \downarrow$. En general, notaremos $\mathbf{STEP}(\mathcal{C}, k, x) = \mathcal{C}_k(x)$.

Definición 7.1 *Para cada membrana, x, y cada computación $\mathcal{C}$ de Π, se define* $\delta(\mathcal{C}, x) = \min\{m : \mathcal{C}_m(x) \uparrow\}$

Es decir, δ es una función parcial de $\mathbf{Comp}(\Pi) \times V(\mu_0)$ en $\mathbf{N}$ de tal manera que $\delta(\mathcal{C}, x)$ representa el instante siguiente al momento en que se ha producido la disolución de la membrana x en la computación $\mathcal{C}$.

Teniendo presente que ninguna membrana está disuelta en la configuración inicial de un P sistema, Π, se tiene trivialmente que $\delta(\mathcal{C}, x) \geq 1$, para cada $\mathcal{C} \in \mathbf{Comp}(\Pi)$ y cada membrana x de Π.

Las herramientas antes definidas son las necesarias para poder analizar en profundidad las computaciones asociadas a los P sistemas, y poder caracterizar de entre ellas las computaciones exitosas; es decir, aquellas que producen el resultado deseado.

7.1. Verificación de un P sistema de transición generador

Como ejemplo de verificación formal en P sistemas de transición generadores se estudia un P sistema que genera el conjunto de cuadrados $\{n^2 : n \geq 1\}$. El P sistema de transición que presentamos a continuación es una ligera variante del que Gh. Păun da en [56]:

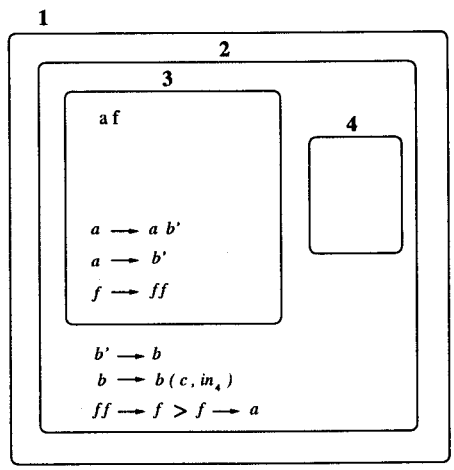


Figura 7.1. Representación del P sistema Π

En primer lugar vamos a describir formalmente la sintaxis de dicho sistema, de acuerdo con los conceptos y las notaciones introducidas en el capítulo anterior.

El P sistema de transición Π es la 4–tupla $(A, C_0, \mathcal{R}, i_0)$, en donde:

1. El alfabeto base es $A = \{a, b, b', c, f\}$.

2. La configuración inicial, $C_0 = (\mu_0, M_0)$, en donde

   - $\mu_0 = (1, ((1,2),(2,1,3,4),(3,2),(4,2)))$ es la estructura de membranas dada por medio del árbol enraizado (con las membranas etiquetadas por números naturales):

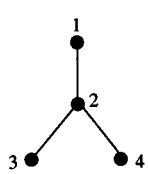


Figura 7.2. Árbol asociado a la estructura de membranas de Π

   - M_0 es la aplicación del conjunto de nodos $\{1,2,3,4\}$ en $\mathbf{M}(A)$ definida como sigue: $M_0(1) = M_0(2) = M_0(4) = \emptyset$ y $M_0(3) = \{af\}$.

3. $\mathcal{R} = (R, \rho)$, donde R es una colección de reglas asociadas a C_0; es decir, R es la aplicación de dominio $\{1,2,3,4\}$, definida por: $R(1) = R(4) = \emptyset$, $R(2) = \{r_1 \equiv b' \rightarrow b, r_2 \equiv b \rightarrow b(c, in_4), r_3 \equiv ff \rightarrow f, r_4 \equiv f \rightarrow a\delta\}$ y $R(3) = \{r_5 \equiv a \rightarrow ab', r_6 \equiv a \rightarrow b'\delta, r_7 \equiv f \rightarrow ff\}$. Y ρ es la aplicación de dominio $\{1,2,3,4\}$ definida por: $\rho(1) = \rho(3) = \rho(4) = \emptyset$ y $\rho(2) = \{(r_3, r_4)\}$. Las reglas $r_1, r_2, \ldots, r_7$ son las siguientes:

Regla	$\mathbf{d_r}$	$\mathbf{v_r}$						δ_r	x_r
		1	2	3	4	*here*	*out*		
r_1	b'	−	−	−	−	b	−	−	2
r_2	b	−	−	−	c	b	−	−	2
r_3	ff	−	−	−	−	f	−	−	2
r_4	f	−	−	−	−	a	−	+	2
r_5	a	−	−	−	−	ab'	−	−	3
r_6	a	−	−	−	−	b'	−	+	3
r_7	f	−	−	−	−	ff	−	−	3

4. La membrana de salida es $i_0 = 4$.

A continuación se trata de caracterizar las computaciones exitosas del P sistema Π. Para ello, y como se ha comentado al principio del capítulo, se va a considerar un predicado sobre las configuraciones de Π que será invariante a lo largo de la ejecución del P sistema Π.

Consideremos la fórmula

$$\theta(\mathcal{C}, n) \equiv (n < \delta(\mathcal{C}, 3) \rightarrow \mathcal{C}_n = (\mu_0, (\emptyset, \emptyset, ab'^n f^{2^n}, \emptyset))) \wedge$$
$$\wedge\ (n = \delta(\mathcal{C}, 3) \rightarrow \mathcal{C} \in S(\Pi) \wedge |\mathcal{O}(\mathcal{C})| = n^2)$$

Para facilitar las pruebas que vienen a continuación, representaremos los multiconjuntos sobre el alfabeto A a través de una palabra sobre A asociada de manera natural y unívoca.

Si $C = (\mu, M)$ es una célula, donde $V(\mu) = \{a_1, \ldots, a_n\} \subset \mathbf{N}$ con $a_1 < \ldots < a_n$, entonces notaremos $M = (M(a_1), \ldots, M(a_n))$.

7.1.1. Caracterización de las configuraciones exitosas

Con el fin de caracterizar las computaciones exitosas del P sistema, en primer lugar vamos a determinar cómo son las configuraciones de Π antes de que se disuelva la membrana 3.

Proposición 7.2 *Para cada computación* $\mathcal{C} \in \mathbf{Comp}(\Pi)$ *se tiene que*

$$\forall k\ (k < \delta(\mathcal{C}, 3) \rightarrow \mathcal{C}_k = (\mu_0, (\emptyset, \emptyset, ab'^k f^{2^k}, \emptyset)))$$

Demostración:

Probémoslo por inducción débil sobre k. Para ello, sea $\mathcal{C} \in \mathbf{Comp}(\Pi)$.

Para el caso base, $k = 0$, basta tener presente que $\delta(\mathcal{C}, 3) \geq 1$ y $\mathcal{C}_0 = (\mu_0, (\emptyset, \emptyset, af, \emptyset))$.

Sea $k \in \mathbf{N}$ tal que $k < \delta(\mathcal{C}, 3)$ y $\mathcal{C}_k = (\mu_0, (\emptyset, \emptyset, ab'^k f^{2^k}, \emptyset)$. Si $k+1 < \delta(\mathcal{C}, 3)$, entonces $k < \delta(\mathcal{C}, 3)$ y, por tanto, $\mathcal{C}_k = (\mu_0, (\emptyset, \emptyset, ab'^k f^{2^k}, \emptyset))$. Como $\mathcal{C}_{k+1}(3) \downarrow$, se deduce que la configuración $\mathcal{C}_{k+1}$ se obtiene a partir de $\mathcal{C}_k$ ejecutando el multiconjunto de aplicabilidad $m = r_5^1 r_7^{2^k}$ sobre $\mathcal{C}_k$, ya que no se ha aplicado la disolución sobre la membrana 3. Por tanto, se tiene que $\mathcal{C}_{k+1} = m(\mathcal{C}_k) = (\mu_0, (\emptyset, \emptyset, ab'^{(k+1)} f^{2^{k+1}}, \emptyset))$.

□

A continuación, probaremos que un *punto crítico* de las computaciones del P sistema Π se produce en el instante en que la membrana 3 se disuelve. Es decir, vamos a justificar que el estudio de la disolución de la membrana 3 es relevante para caracterizar las computaciones exitosas de Π.

Proposición 7.3 *Para cada computación $\mathcal{C} \in$ **Comp**(Π) tal que $\delta(\mathcal{C}, 3) = n < \infty$, se tiene:*

1. $\mathcal{C}_n = (\mu', (\emptyset, b'^n f^{2^n}, \emptyset))$, *donde* $\mu' = (1, ((1,2), (2,1,4), (4,2)))$.
2. *Para cada k tal que $0 \leq k \leq n-1$, se tiene que $\mathcal{C}_{n+1+k} = (\mu', (\emptyset, b^n f^{2^{n-k-1}}, c^{kn}))$, donde* $\mu' = (1, ((1,2), (2,1,4), (4,2)))$.
3. $\mathcal{C}_{2n+1} = (\mu'', (ab^n, c^{n^2}))$, *donde* $\mu'' = (1, ((1,4), (4,1)))$.
4. *La computación $\mathcal{C}$ es exitosa, su longitud es $|\mathcal{C}| = 2n+1$ y, además, la salida de esta computación verifica que $|\mathcal{O}(\mathcal{C})| = n^2$.*

Demostración:

1. Si $n = \delta(\mathcal{C}, 3) < \infty$, entonces $0 \leq n-1 < \delta(\mathcal{C}, 3)$. De la proposición 7.2, se deduce que $\mathcal{C}_{n-1} = (\mu_0, (\emptyset, \emptyset, ab'^{(n-1)} f^{2^{n-1}}, \emptyset))$. Como $\delta(\mathcal{C}, 3) = n$, resulta que la configuración $\mathcal{C}_n$ se obtiene de $\mathcal{C}_{n-1}$ aplicando el multiconjunto $m = r_6 r_7^{2^{n-1}}$ sobre $\mathcal{C}_{n-1}$. Por tanto, $\mathcal{C}_n = m(\mathcal{C}_{n-1}) = (\mu', (\emptyset, b'^n f^{2^n}, \emptyset))$, donde $\mu' = (1, ((1,2), (2,1,4), (4,2)))$.
2. Lo probaremos por inducción sobre k.

   Para probar el caso base, $k = 0$, basta tener presente que del apartado (1) resulta que $\mathcal{C}_n = (\mu', (\emptyset, b'^n f^{2^n}, \emptyset))$. En esta situación, como $n \geq 1$, es posible aplicar la regla r_3 a la membrana 2 y, por el sentido fuerte de

la prioridad, la regla r_4 no se puede aplicar a $\mathcal{C}_n$ (esta regla disolvería la membrana 2). Por tanto, el único multiconjunto de aplicabilidad sobre $\mathcal{C}_n$ será $m = r_1^n r_3^{2^{n-1}}$. En consecuencia, $\mathcal{C}_{n+1} = m(\mathcal{C}_n) = (\mu', (\emptyset, b^n f^{2^{n-1}}, \emptyset))$.

Sea k tal que $0 \leq k < n-1$, y supongamos que la configuración $\mathcal{C}_{n+1+k}$ es $(\mu', (\emptyset, b^n f^{2^{n-k-1}}, c^{kn}))$. Como $n-k-1 > 0$, se deduce que es posible aplicar la regla r_3 a la membrana 2 y, por tanto, el único multiconjunto de aplicabilidad sobre $\mathcal{C}_{n+1+k}$ es $m = r_2^n r_2^{2^{n-k-2}}$. De donde se obtiene que

$$\mathcal{C}_{n+1+k+1} = m(\mathcal{C}_{n+1+k}) = (\mu', (\emptyset, b^n f^{2^{n-k-2}}, c^{(k+1)n}))$$

3. Aplicando el apartado (2) al caso $k = n-1$, se deduce que la configuración $\mathcal{C}_{2n}$ es $(\mu', (\emptyset, b^n f, c^{(n-1)n}))$. Por tanto, el único multiconjunto de aplicabilidad sobre $\mathcal{C}_{2n}$ es $m = r_2^n r_4$. De donde resulta que $\mathcal{C}_{2n+1} = m(\mathcal{C}_{2n}) = (\mu'', (ab^n, c^{n^2}))$, con $\mu'' = (1, ((1,4),(4,1)))$.

4. Del apartado (3) se deduce que $\mathcal{C}_{2n+1} = (\mu'', (ab^n, c^{n^2}))$. Teniendo presente que $V(\mu'') = \{1,4\}$ y $R_1 = R_4 = \emptyset$, resulta que $\mathbf{M_{Ap}}(\mathcal{C}_{2n+1}) = \{\emptyset\}$. Entonces la configuración $\mathcal{C}_{2n+1}$ es de parada. Además, como $4 \in V(\mu'')$ y es una hoja del árbol, resulta que $\mathcal{C}_{2n+1}$ es exitosa. Por tanto, la computación $\mathcal{C}$ es exitosa, su longitud es $2n+1$ y la salida verifica $|\mathcal{O}(\mathcal{C})| = |\mathcal{C}_{2n+1}(4)| = n^2$.

□

Como primera consecuencia de esta proposición, vamos a ver que tras el instante en que la membrana 3 se disuelve, el P sistema evoluciona de forma *determinista.*

Corolario 7.4 *Para cada $n \geq 1$ y cada $\mathcal{C}, \mathcal{C}' \in$ **Comp**(Π) tal que $n = \delta(\mathcal{C}, 3) = \delta(\mathcal{C}', 3)$ se tiene que $\forall k\ (n \leq k \leq 2n+1 \rightarrow \mathcal{C}_k = \mathcal{C}'_k)$.*

Demostración:
El caso $k = n$ sigue del apartado (1) de la proposición 7.3, el caso $n < k \leq 2n$ sigue del apartado (2), y el caso $k = 2n+1$ sigue del apartado (3).

□

A continuación, veamos que si dos computaciones disuelven la membrana 3 en el mismo instante, entonces dichas computaciones son iguales.

Corolario 7.5 *Para cada $n \geq 1$ y cada $\mathcal{C}, \mathcal{C}' \in$ **Comp**(Π) tales que $n = \delta(\mathcal{C}, 3) = \delta(\mathcal{C}', 3)$ se tiene que $\mathcal{C} = \mathcal{C}'$.*

Demostración:
Sea $n \geq 1$ y $\mathcal{C}, \mathcal{C}' \in \mathbf{Comp}(\Pi)$ tales que $n = \delta(\mathcal{C}, 3) = \delta(\mathcal{C}', 3)$. Por el apartado (4) de la proposición 7.3 y el corolario 7.4, basta probar que $\forall k\ (0 \leq k \leq n-1 \rightarrow \mathcal{C}_k = \mathcal{C}'_k)$. Y esta relación se obtiene directamente de la proposición 7.2.

□

Corolario 7.6 *Existe a lo sumo una computación de* Π *que no es exitosa.*

Demostración:
Sea $\mathcal{C}$ una computación de Π que no es exitosa. De la proposición 7.3 se deduce que $\forall k\ (k < \delta(\mathcal{C}, 3))$. Por tanto, de la proposición 7.2 resulta que la configuración $\mathcal{C}_k$ es $(\mu_0, (\emptyset, \emptyset, ab'^k f^{2^k}, \emptyset)))$. En consecuencia, sólo podría existir una computación no exitosa de Π.

□

Probemos ahora que la fórmula $\theta(\mathcal{C}, n)$ es verdadera para toda computación $\mathcal{C}$ del P sistema de transición Π.

Corolario 7.7 *La fórmula* $\theta(\mathcal{C}, n)$ *es un invariante del P sistema* Π. *Es decir*

$$\forall \mathcal{C} \in \mathbf{Comp}(\Pi)\ \forall n \in \mathbf{N}\ (\theta(\mathcal{C}, n))$$

Demostración:
Se sigue directamente de la proposición 7.2 y del apartado (4) de la proposición 7.3.

□

A continuación, vamos a caracterizar las computaciones exitosas de Π a través del instante en el que la membrana 3 se disuelve.

Corolario 7.8 *Sea* $\mathcal{C}$ *una computación de* Π. *Son equivalentes:*

(a) $\mathcal{C}$ *es una computación exitosa.*

(b) $1 \leq \delta(\mathcal{C}, 3) < \infty$.

(c) $1 \leq \delta(\mathcal{C}, 3) < \infty$ *y* $|\mathcal{C}| = 2 \cdot \delta(\mathcal{C}, 3) + 1$.

Demostración:
Sea $\mathcal{C}$ una computación exitosa. Sea $k = |\mathcal{C}|$. Entonces $1 \leq k < \infty$. Veamos que $\delta(\mathcal{C}, 3) \leq k$. En caso contrario, de la proposición 7.2 se tendría que $\mathcal{C}_k$ sería

$(\mu_0, (\emptyset, \emptyset, ab'^k f^{2^k}, \emptyset))$. Lo que contradice que $k = |\mathcal{C}|$, ya que de la existencia de un multiconjunto de aplicabilidad sobre C_k (por ejemplo, $m = r_5 r_7^{2^k}$) se tendría que C_k no es una configuración de parada.

Si $1 \leq n = \delta(\mathcal{C}, 3) < \infty$, entonces del apartado (4) de la proposición 7.3 resulta que $|\mathcal{C}| = 2n + 1$.

La implicación $(c) \Rightarrow (a)$ resulta directamente del apartado (4) de la proposición 7.3.

□

7.1.2. Corrección y completitud

Para establecer que el conjunto de números naturales generados por Π es $\mathbf{N}(\Pi) = \{n^2 : n \geq 1\}$ debemos probar dos resultados:

- La salida de cualquier computación exitosa del P sistema Π codifica el cuadrado de un número natural mayor o igual que 1 (*corrección* del P sistema).
- Para cada $n \geq 1$ existe, al menos, una computación exitosa, $\mathcal{C}$, del P sistema Π cuya salida verifica que $|\mathcal{O}(\mathcal{C})| = n^2$ (*completitud* del P sistema).

Teorema 7.9 (Corrección) *Si $\mathcal{C}$ es una computación exitosa del P sistema Π, entonces existe $n \geq 1$ tal que la salida de $\mathcal{C}$ verifica que $|\mathcal{O}(\mathcal{C})| = n^2$.*

Demostración:
Sea $\mathcal{C}$ una computación exitosa de Π. Si $n = \delta(\mathcal{C}, 3)$, entonces del corolario 7.5 resulta que $1 \leq n < \infty$. Como la fórmula $\theta(\mathcal{C}, n)$ es verdadera y $n = \delta(\mathcal{C}, 3)$, se deduce que la computación $\mathcal{C}$ es exitosa y $|\mathcal{O}(\mathcal{C})| = n^2$.

□

Para establecer la completitud del P sistema de transición Π con respecto a la generación del conjunto $\{n^2 : n \geq 1\}$, consideremos la fórmula $\varphi(n) \equiv \exists \mathcal{C} \in \mathbf{Comp}(\Pi)\ (n = \delta(\mathcal{C}, 3))$. Veamos que esta fórmula es verdadera para cada número natural mayor o igual que 1.

Proposición 7.10 *Para cada número natural $n \geq 1$ existe una única computación, $\mathcal{C} \in \mathbf{Comp}(\Pi)$, tal que $\delta(\mathcal{C}, 3) = n$.*

Demostración:
Probemos la existencia por inducción sobre n.

Para el caso base, $n = 1$, se considera la configuración $\mathcal{C}_1$ que se obtiene de la configuración inicial, $\mathcal{C}_0$, aplicando la matriz $m = r_6 r_7$ (multiconjunto de aplicabilidad sobre $\mathcal{C}_0$). Como $r_6 \equiv a \to b'\delta$, se obtiene que $\delta(\mathcal{C}, 3) = 1$.

Sea $n \geq 1$ y supongamos que el resultado es cierto para n. Sea $\mathcal{C}$ una computación de Π verificando $\delta(\mathcal{C}, 3) = n$. De la proposición 7.2, se deduce que $\mathcal{C}_{n-1} = (\mu_0, (\emptyset, \emptyset, ab'^{(n-1)} f^{2^{n-1}}, \emptyset)))$. Entonces se tiene que los multiconjuntos de aplicabilidad de la configuración $\mathcal{C}_{n-1}$ son $\mathbf{M_{Ap}}(\mathcal{C}_{n-1}) = \{m_1, m_2\}$, donde $m_1 = r_6 r_7^{2^{n-1}}$ y $m_2 = r_5 r_7^{2^{n-1}}$. Sea $\mathcal{C}'_n = m_2(\mathcal{C}_{n-1}) = (\mu_0, (\emptyset, \emptyset, ab'^n f^{2^n}, \emptyset))$. Sea $\mathcal{C}'_{n+1} = m_3(\mathcal{C}'_n)$, donde $m_3 = r_6 r_7^{2^n}$. En este paso, la membrana 3 se disuelve. Entonces $\mathcal{C}'_{n+1} = (\mu', (\emptyset, b'^{(n+1)} f^{2^{n+1}}, \emptyset))$, donde la estructura de membranas μ' es $(1, ((1,2), (2,1,4), (4,2)))$. Por tanto, la computación $\mathcal{C}' \equiv \mathcal{C}_0 \Rightarrow_\Pi \mathcal{C}_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi \mathcal{C}_{n-1} \Rightarrow_\Pi \mathcal{C}'_n \Rightarrow_\Pi \mathcal{C}'_{n+1} \Rightarrow_\Pi \ldots$, verifica que $\delta(\mathcal{C}', 3) = n + 1$.

Dado $n \geq 1$, la unicidad de la computación $\mathcal{C}$ verificando $\delta(\mathcal{C}, 3) = n$, se sigue directamente del corolario 7.5.

□

Proposición 7.11 *Existe una única computación, $\mathcal{C}$, de Π que no es exitosa.*

Demostración:
Primero, probaremos que tal computación existe.

Para cada $k \in \mathbf{N}$, consideremos la configuración $\mathcal{C}_k = (\mu_0, (\emptyset, \emptyset, ab'^k f^{2^k}, \emptyset)))$ de Π. Si $m = r_5 r_7^{2^k}$, entonces se tiene que $\forall k \ (\mathcal{C}_{k+1} = m(\mathcal{C}_k))$. Entonces la sucesión $\mathcal{C} \equiv \mathcal{C}_0 \Rightarrow_\Pi \mathcal{C}_1 \Rightarrow_\Pi \ldots \mathcal{C}_k \Rightarrow_\Pi \mathcal{C}_{k+1} \Rightarrow_\Pi \ldots$ es una computación de Π. Además, por construcción, es obvio que $\mathcal{C}$ no es una computación de parada.

La unicidad de tal computación se sigue del corolario 7.6.

□

Corolario 7.12 *Para cada $n \geq 1$ la fórmula $\varphi(n)$ es verdadera.*

Teorema 7.13 (Completitud) *Para cada número natural $n \geq 1$ existe una computación exitosa, $\mathcal{C}$, del P sistema Π, cuya salida verifica que $|\mathcal{O}(\mathcal{C})| = n^2$.*

Demostración:
Sea $n \in \mathbf{N}$ tal que $n \geq 1$. Como la fórmula $\varphi(n)$ es verdadera, existe una computación, $\mathcal{C}$, de Π tal que $\delta(\mathcal{C}, 3) = n$. Como la fórmula $\theta(\mathcal{C}, n)$ es verdadera, se concluye que la computación $\mathcal{C}$ es exitosa y, además, $|\mathcal{O}(\mathcal{C})| = n^2$. □

7.2. Verificación de un P sistema de transición reconocedor

En esta sección damos un P sistema de transición reconocedor, Π_d, de orden 2 que reconoce el predicado de divisibilidad, tal y como fue propuesto por Gh. Păun en [56].

El P sistema de reconocimiento diseñado en [56] puede ser descrito gráficamente como sigue:

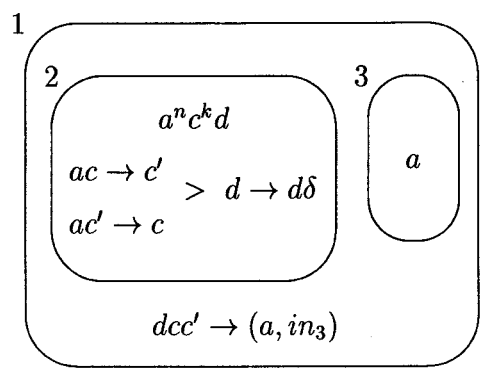


Figura 7.3. Representación del P sistema Π_d con entrada $(n, k) \in \mathbf{N}^2$

En donde la membrana 3 es la de salida, y $n, k \in \mathbf{N}$ son "variables de entrada".

A continuación vamos a formalizar la sintaxis del P sistema de reconocimiento Π_d, de acuerdo con lo desarrollado en el capítulo anterior.

El P sistema Π_d es una 6–tupla $(A, B, C_0, \mathcal{R}, i_0, j_0, \varphi)$, en donde:

1. El alfabeto base es $A = \{a, c, c', d\}$ y $B = \{a, c\}$, donde se supone en B el orden lexicográfico $(a < c)$.

2. La configuración inicial, $C_0 = (\mu_0, M_0)$, está definida como sigue:

   - $\mu_0 = (1, ((1, 2), (1, 3)))$ es la estructura de membranas dada por el árbol enraizado siguiente (con las membranas etiquetadas por números naturales):

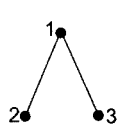


Figura 7.4. Árbol asociado a la estructura de membranas de Π_d

- $M_0 : \{1,2,3\} \to \mathbf{M}(A)$ es la aplicación definida por:

$$M_0(1) = \emptyset,\ M_0(2) = \{a^n c^k d\},\ M_0(3) = a$$

donde $n, k \in \mathbf{N}$ son los datos de entrada del predicado.

3. $\mathcal{R} = (R, \rho)$, donde R es una colección de reglas asociadas a C_0; es decir, R es la aplicación de dominio $\{1,2,3\}$, definida por: $R(1) = \{r_1 \equiv dcc' \to (a, in_3)\}$, $R(2) = \{r_2 \equiv ac \to c',\ r_3 \equiv ac' \to c,\ r_4 \equiv d \to d\delta\}$, $R(3) = \emptyset$. Y ρ es la aplicación de dominio $\{1,2,3\}$ definida por: $\rho(1) = \rho(3) = \emptyset$ y $\rho(2) = \{(r_2, r_4), (r_3, r_4)\}$.

Regla	$\mathbf{d_r}$	$\mathbf{v_r}$					δ_r	x_r
		1	2	3	*here*	*out*		
r_1	dcc'	−	−	a	−	−	−	1
r_2	ac	−	−	−	c'	−	−	2
r_3	ac'	−	−	−	c	−	−	2
r_4	d	−	−	−	d	−	+	2

4. La membrana de salida es $i_0 = 3$ y la membrana de entrada es $j_0 = 2$.

5. $\varphi : \mathbf{M}(A) \to \{0,1\}$ es el predicado definido por

$$\varphi(m) = \begin{cases} 1 & \text{, si } m = a \\ 0 & \text{, e.c.o.c} \end{cases}$$

Es decir, $\varphi(m) \equiv m = a$.

7.2.1. Caracterización de las configuraciones de parada

A continuación vamos a realizar un estudio de la evolución del P sistema reconocedor Π_d según la relación existente entre los datos de entrada, n y k.

Nota 7.14 Si $b \in \{c, c'\}$, entonces notaremos

$$b' = \begin{cases} c & \text{si } b = c' \\ c' & \text{si } b = c \end{cases}$$

Proposición 7.15 *Sean* $h = \lfloor \frac{n}{k} \rfloor$ *y* $\mathcal{C}$ *una computación de* Π_d. *Entonces:*

1. *Para cada* $j \leq h$, *se tiene que* $\mathcal{C}_j = (\mu_0, (\emptyset, a^{n-kj} b_j^k d, a))$, *en donde* $b_j = c$ *si* j *es par y* $b_j = c'$ *si* j *es impar.*

2. *Si* $k \mid n$, *entonces* $\delta(\mathcal{C}, 2) = h+1$, *la configuración* $\mathcal{C}_{h+1}$ *es de parada y, además, la salida de la computación es* $\mathcal{O}(\mathcal{C}) = a$.

3. *Si* $k \nmid n$, *entonces* $\delta(\mathcal{C}, 2) = h+2$, *la configuración* $\mathcal{C}_{h+3}$ *es de parada y, además, la salida de la computación es* $\mathcal{O}(\mathcal{C}) = a^2$.

Demostración:

1. Por inducción sobre j:

   Para el caso base, $j = 0$, el resultado se obtiene directamente de la construcción del P sistemas Π_d.

   Sea j tal que $j + 1 \leq h$ y supongamos que el resultado es cierto para j. Por hipótesis de inducción, se tiene que $\mathcal{C}_j = (\mu_0, (\emptyset, a^{n-kj} b_j^k d, a))$, en donde $b_j = c$ si j es par y $b_j = c'$ si j es impar. Como $j < j + 1 \leq h$, se verifica que $n - kj \geq k$. Por tanto, $\mathbf{M_{Ap}}(\mathcal{C}_j) = \{r^k\}$, donde $r = r_2$ si $b_j = c$ y $r = r_3$ si $b_j = c'$. De donde se deduce que $\mathcal{C}_{j+1} = r^k(\mathcal{C}_j) = (\mu_0, (\emptyset, a^{(n-kj)-k} b_{j+1}^k d, a))$, con $b_{j+1} = b'_j$.

2. Supongamos que $k \mid n$. En este caso, del apartado anterior resulta que $\mathcal{C}_h = (\mu_0, (\emptyset, b_h^k d, a))$, en donde $b_h = c$ si h es par y $b_h = c'$ si h es impar. Analizando las reglas del P sistema, se deduce que $\mathbf{M_{Ap}}(\mathcal{C}_h) = \{r_4\}$ y $\mathcal{C}_{h+1} = r_4(\mathcal{C}_h)(\mu_1, (b_h^k d, a))$, donde μ_1 es la estructura de membranas obtenida de μ_0 eliminando la membrana 2 (ya que la regla r_4 implica una disolución); es decir, $\mu_1 = (1, (1, 3))$. De donde se obtiene que $\delta(\mathcal{C}, 2) = h + 1$. Además, se tiene que $\mathbf{M_{Ap}}(\mathcal{C}_{h+1}) = \{\emptyset\}$. Luego la configuración $\mathcal{C}_{h+1}$ es de parada y, además, la salida de la computación $\mathcal{C}$ es $\mathcal{O}(\mathcal{C}) = M(3) = a$.

3. Supongamos que $k \nmid n$. En este caso, del apartado (1) resulta que la configuración $\mathcal{C}_h$ es $(\mu_0, (\emptyset, a^{n-kh} b^k d, a))$, donde $k > l = n - kh > 0$ y $b \in \{c, c'\}$. Entonces $\mathbf{M_{Ap}}(\mathcal{C}_h) = \{r^l\}$, con $r = r_2$ si $b = c$ y $r = r_3$ si $b = c'$. En consecuencia, $\mathcal{C}_{h+1} = r^l(\mathcal{C}_h) = (\mu_0, (\emptyset, b^{k-l} b'^l d, a))$. Analizando las reglas del P sistema, se deduce que $\mathbf{M_{Ap}}(\mathcal{C}_{h+1}) = \{r_4\}$ y $\mathcal{C}_{h+2} = r_4(\mathcal{C}_{h+1}) = (\mu_1, (b^{k-l} b'^l d, a))$, donde μ_1 es la misma estructura del apartado anterior. Por tanto $\delta(\mathcal{C}, 2) = h + 2$. Además, se tiene que $\mathbf{M_{Ap}}(\mathcal{C}_{h+2}) = \{r_1\}$. Luego $\mathcal{C}_{h+3} = r_1(\mathcal{C}_{h+2}) = (\mu_1, (b^{k-l-1} b'^{l-1}, a^2))$. Entonces $\mathbf{M_{Ap}}(\mathcal{C}_{h+3}) = \{\emptyset\}$. Por tanto, la configuración

$\mathcal{C}_{h+3}$ es de parada y, además, la salida de la computación $\mathcal{C}$ es $\mathcal{O}(\mathcal{C}) = M(3) = a^2$.

□

Analizando con detalle la demostración anterior se deduce que, implícitamente, se ha probado que el P sistema Π_d es determinista. Por tanto, Π_d es coherente con la definición 6.28 dada para P sistemas reconocedores.

7.2.2. Corrección y completitud

Como consecuencia inmediata del resultado anterior podemos establecer la verificación formal del P sistema Π_d respecto al predicado de divisibilidad.

Teorema 7.16 *El P sistema Π_d decide el predicado binario de divisibilidad, div, definido como sigue:*

$$div(n,k) = \begin{cases} 1 & si\ k \mid n \\ 0 & si\ k \nmid n \end{cases}$$

Demostración:
Sea $(n,k) \in \mathbf{N}^2$. De la proposición anterior, se deduce que:

$$k \mid n \Longleftrightarrow \mathcal{O}(\mathcal{C}) = a \Longleftrightarrow \varphi(\mathcal{O}(\mathcal{C})) = 1 \Longleftrightarrow \Pi_d \text{ acepta } (n,k)$$

En consecuencia, el P sistema Π_d decide el predicado *div*.

□

7.3. Verificación de un P sistema de cálculo

En esta sección se describe un P sistema de cálculo, Π, de orden 1 que calcula la función parcial $f : \mathbf{N}- \rightarrow P(\mathbf{N})$ definida como sigue:

$$f(n) = \begin{cases} \uparrow & \text{si } n = 0 \\ \{1^2, 2^2, \ldots, n^2\} & \text{si } n \neq 0 \end{cases}$$

El P sistema de cálculo Π que presentamos es una ligera variante del que Gh. Păun da en [56] y puede ser descrito gráficamente como sigue:

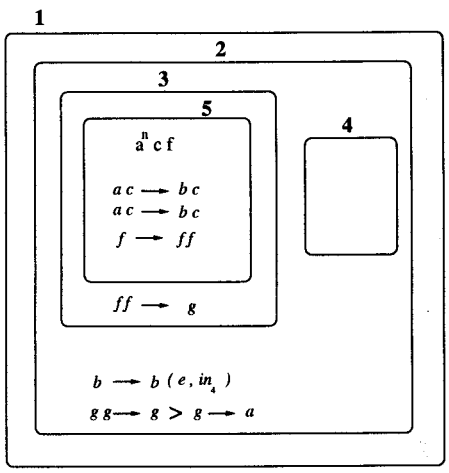


Figura 7.5. Representación del P sistema Π con dato de entrada $n \in \mathbf{N}$

en donde la membrana 4 es la de salida, y $n \in \mathbf{N}$ es el dato de entrada.

A continuación vamos a formalizar la sintaxis del P sistema de cálculo Π, de acuerdo con lo desarrollado en el capítulo anterior.

El P sistema Π es una 6–tupla $(A, B, C_0, \mathcal{R}, i_0, j_0)$, en donde:

1. El alfabeto base es $A = \{a, b, c, e, f, g\}$ y $B = \{a\}$.

2. La configuración inicial, $C_0 = (\mu_0, M_0)$, está definida como sigue:

   - $\mu_0 = (1, ((1, 2), (2, 1, 3, 4), (3, 2, 5), (4, 2), (5, 3)))$ es la estructura de membranas dada por el árbol enraizado siguiente (con las membranas etiquetadas por números naturales):

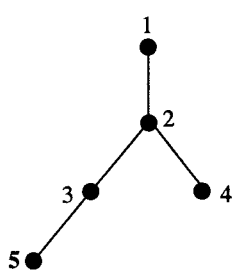


Figura 7.6. Árbol asociado a la estructura de membranas de Π

   - M_0 es la aplicación del conjunto de nodos $\{1, 2, 3, 4, 5\}$ en $\mathbf{M}(A)$ definida como sigue:

$$M_0(1) = M_0(2) = M_0(3) = M_0(4) = \emptyset,\ M_0(5) = \{a^n cf\}$$

   donde $n \in \mathbf{N}$ es el dato de entrada de la función.

3. $\mathcal{R} = (R, \rho)$, donde R es una colección de reglas asociadas a C_0; es decir, R es la aplicación de dominio $\{1,2,3,4,5\}$, definida por: $R(1) = R(4) = \emptyset$, $R(2) = \{r_1 \equiv b \to b(e, in_4), r_2 \equiv gg \to g, r_3 \equiv g \to a\delta\}$, $R(3) = \{r_4 \equiv ff \to g\delta\}$ y $R(5) = \{r_5 \equiv ac \to bc, r_6 \equiv ac \to bc\delta, r_7 \equiv f \to ff\}$. Y ρ es la aplicación de dominio $\{1,2,3,4,5\}$ definida por: $\rho(1) = \rho(3) = \rho(4) = \rho(5) = \emptyset$ y $\rho(2) = \{(r_2, r_3)\}$.

   Las reglas $r_1, r_2, \ldots, r_7$ son las siguientes:

Regla	$\mathbf{d_r}$	$\mathbf{v_r}$							δ_r	x_r
		1	2	3	4	5	*here*	*out*		
r_1	b	$-$	$-$	$-$	e	$-$	b	$-$	$-$	2
r_2	gg	$-$	$-$	$-$	$-$	$-$	g	$-$	$-$	2
r_3	g	$-$	$-$	$-$	$-$	$-$	a	$-$	$+$	2
r_4	ff	$-$	$-$	$-$	$-$	$-$	g	$-$	$+$	3
r_5	ac	$-$	$-$	$-$	$-$	$-$	bc	$-$	$-$	5
r_6	ac	$-$	$-$	$-$	$-$	$-$	bc	$-$	$+$	5
r_7	f	$-$	$-$	$-$	$-$	$-$	ff	$-$	$-$	5

4. La membrana de salida es $i_0 = 4$ y la membrana de entrada es $j_0 = 5$.

Notación: Dado $n \in \mathbf{N}$, notaremos $\Pi(n)$ al P sistema Π con dato de entrada n; es decir, el P sistema Π cuya configuración inicial es $\mathcal{C}_0 = (\mu_0, M_0)$, en donde μ_0 es la estructura de membranas asociada a Π, $\forall i\ (1 \leq i \leq 4 \to M_0(i) = \emptyset)$ y $M_0(5) = \{a^n cf\}$.

7.3.1. Caracterización de las configuraciones exitosas

En primer lugar, vamos a probar que existe una única computación del P sistema $\Pi(0)$ y, además, que dicha computación no es de parada (por tanto, no será exitosa).

La notación será similar a la usada en la sección anterior.

Proposición 7.17 *Sea $\mathcal{C}$ una computación de $\Pi(0)$. Entonces, para cada $k \geq 0$ se tiene que $\mathcal{C}_k = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, cf^{2^k}))$. Además, se verifica que $\mathbf{M_{Ap}}(\mathcal{C}_k) = \{r_7^{2^k}\}$.*

Demostración:
El resultado se obtiene directamente por inducción sobre k.

□

Corolario 7.18 *Existe una única computación de* $\Pi(0)$. *Además, dicha computación no es de parada.*

Demostración:
Tanto la existencia como la unicidad se deducen directamente de la proposición 7.17. Además, dicha computación no es de parada, ya que para todo k se tiene que $\mathbf{M_{Ap}}(\mathcal{C}_k) \neq \{\emptyset\}$ y, por tanto, existe la configuración siguiente $\mathcal{C}_{k+1}$.

□

Para caracterizar las computaciones exitosas del P sistema $\Pi(n)$, con $n \geq 1$, vamos a estudiar qué sucede en el instante en que la membrana 5 se disuelve. Para ello, en primer lugar vamos a determinar el contenido de la membrana 5 en las configuraciones de la computación en las que todavía no se ha disuelto.

Proposición 7.19 *Sea* $n \geq 1$. *Para cada* $k \in \mathbf{N}$ *tal que* $k \leq n$ *y cada computación* $\mathcal{C}$ *de* $\Pi(n)$ *tal que* $k < \delta(\mathcal{C}, 5)$, *se verifica que* $\mathcal{C}_k = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-k}cb^k f^{2^k}))$.

Demostración:
Sea $n \geq 1$. Consideremos el P sistema $\Pi(n)$ con dato de entrada n. Probemos el resultado por inducción débil acotada ascendente sobre k.

> Para el caso base, $k = 0$, basta tener presente que la configuración inicial de $\Pi(n)$ es $\mathcal{C}_0 = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^n cf))$.
>
> Sea $k \in \mathbf{N}$ tal que $k < n$ y supongamos cierto el resultado para k. Sea $\mathcal{C}$ una computación de $\Pi(n)$ tal que $k+1 < \delta(\mathcal{C}, 5)$. Entonces $k < \delta(\mathcal{C}, 5)$ y, por tanto, de la hipótesis de inducción se deduce que $\mathcal{C}_k = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-k}cb^k f^{2^k}))$. Como $k + 1 < \delta(\mathcal{C}, 5)$ y $k < n$, resulta que el multiconjunto de aplicabilidad sobre $\mathcal{C}_k$ usado para obtener $\mathcal{C}_{k+1}$ es $m = r_5 r_7^{2^m}$, Por tanto, $\mathcal{C}_{k+1} = m(\mathcal{C}_k) = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-k-1}cbb^k f^{2^{k+1}})) = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-(k+1)}cb^{k+1} f^{2^{k+1}}))$. Lo que prueba que el resultado es válido para $k + 1$.

□

En primer lugar, vamos a dar un predicado sobre configuraciones que sea, en cierto sentido, un invariante del P sistema. Consideremos la fórmula:

$$\theta(\mathcal{C}, p, n) \equiv (\mathcal{C} \in \mathbf{Comp}(\Pi(n)) \wedge 1 \leq p \leq n \wedge p = \delta(\mathcal{C}, 5)) \rightarrow (\mathcal{C} \text{ exitosa} \wedge |\mathcal{O}(\mathcal{C})| = p^2)$$

Seguidamente establecemos una condición necesaria para que una computación, $\mathcal{C}$, de $\Pi(n)$, con $n \geq 1$, sea exitosa.

Proposición 7.20 *Para cada $n \geq 1$ existe una única computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) > n$. Además, en tal situación se tiene que $\delta(\mathcal{C}, 5) = \infty$ y, por tanto, la computación $\mathcal{C}$ no es de parada.*

Demostración:
Sea $n \geq 1$. En primer lugar probemos la existencia de dicha computación.

Sea $\mathcal{C}_0$ la configuración inicial de $\Pi(n)$. Para cada $k \in \mathbf{N}$ tal que $1 \leq k \leq n$ notemos $m_k = r_5 r_7^{2^{k-1}}$ y $D_k = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-k} c b^k f^{2^k}))$. Además, convendremos que $D_0 = \mathcal{C}_0$.

> **Lema 1:** *Para cada k tal que $1 \leq k \leq n$ se tiene que m_k es un multiconjunto de aplicabilidad sobre D_{k-1} y, además, $\mathcal{C}_k = m_k(\mathcal{C}_{k-1}) = D_k$.*
>
> *Prueba del lema:* Por inducción sobre k.
>
> Para probar el caso base, $k = 1$, basta tener presente que $m_1 = r_5 r_7$ es un multiconjunto de aplicabilidad sobre D_0. Además,
>
> $$\mathcal{C}_1 = m_1(\mathcal{C}_0) = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-1} c b f^2)) = D_1$$
>
> Sea k tal que $1 \leq k < n$ y supongamos cierto el resultado para k. Se tiene que $m_k = r_5 r_7^{2^{k-1}}$ es un multiconjunto de aplicabilidad sobre D_{k-1}. Además, $\mathcal{C}_k = m_k(\mathcal{C}_{k-1}) = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-k} c b^k f^{2^k})) = D_k$. Entonces $m_{k+1} = r_5 r_7^{2^k}$ es un multiconjunto de aplicabilidad sobre D_k. Además, $\mathcal{C}_{k+1} = m_{k+1}(\mathcal{C}_k) = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-k-1} c b^{k+1} f^{2^{k+1}})) = D_{k+1}$.

Del lema anterior resulta que $\mathcal{C}_n = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, c b^n f^{2^n})) = D_n$. Para cada número natural $q \geq 1$, notemos $m_{n+q} = r_5 r_7^{2^{n+q-1}}$ y $D_{n+q} = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, c b^n f^{2^{n+q}}))$.

> **Lema 2:** *Para cada $q \geq 1$ se tiene que m_{n+q} es un multiconjunto de aplicabilidad sobre D_{n+q-1} y, además, $\mathcal{C}_{n+q} = m_{n+q}(\mathcal{C}_{n+q-1}) = D_{n+q}$.*
>
> *Prueba del lema:* Por inducción sobre q.
>
> Para probar el caso base, $q = 1$, basta tener presente que $m_{n+1} = r_5 r_7^{2^n}$ es un multiconjunto de aplicabilidad sobre D_n. Además, $\mathcal{C}_{n+1} = m_{n+1}(\mathcal{C}_n) = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, c b^n f^{2^{n+1}})) = D_{n+1}$.
>
> Sea $q \geq 1$ y supongamos cierto el resultado para q. Se tiene que $\mathcal{C}_{n+q} = D_{n+q} = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, c b^n f^{2^{n+q}}))$. Entonces $m_{n+q+1} = r_5 r_7^{2^{n+q}}$ es un multiconjunto de aplicabilidad sobre D_{n+q}. Además,
>
> $$\mathcal{C}_{n+q+1} = m_{n+q+1}(\mathcal{C}_{n+q}) = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, c b^n f^{2^{n+q+1}})) = D_{n+q+1}$$

La computación $\mathcal{C}$, de $\Pi(n)$, así construida verifica que $\delta(\mathcal{C}_n, 5) = \infty$ y, por tanto, no es de parada.

Finalmente veamos la unicidad. Para ello, supongamos que $\mathcal{C}, \mathcal{C}'$ son dos computaciones de $\Pi(n)$ tales que $\delta(\mathcal{C}, 5) > n$ y $\delta(\mathcal{C}', 5) > n$. De la proposición 7.19 se deduce que para cada $k \in \mathbf{N}$ tal que $k \leq n$ se verifica la siguiente condición $\mathcal{C}_k = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-k}cb^k f^{2^k})) = \mathcal{C}'_k$. Como $\mathcal{C}_n = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, cb^n f^{2^n}))$, resulta que para cada $q \geq 1$ se verifica que $\mathcal{C}_{n+q} = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, cb^n f^{2^{n+q}})) = \mathcal{C}'_{n+q}$.

□

Corolario 7.21 *Sea $n \geq 1$. Sea $\mathcal{C}$ una computación exitosa de $\Pi(n)$. Entonces*

$$\forall k \; (k < \delta(\mathcal{C}, 5) \rightarrow k < n)$$

Demostración:
Sea $n \geq 1$. Supongamos que $\mathcal{C}$ fuese una computación exitosa de $\Pi(n)$ para la que existe $k \geq n$ verificando que $k < \delta(\mathcal{C}, 5)$. Entonces, $n < \delta(\mathcal{C}, 5)$ y, por tanto, de la proposición 7.20 se deduciría que la computación $\mathcal{C}$ no es de parada. Y, en consecuencia, no sería exitosa.

□

A continuación veamos que para cada $n \geq 1$ se puede construir una computación de $\Pi(n)$ que sea exitosa y cuya salida sea 1.

Proposición 7.22 *Sea $n \geq 1$. Existe una única computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) = 1$. Además, la computación $\mathcal{C}$ es exitosa, tiene longitud 3 y su salida verifica que $|\mathcal{O}(\mathcal{C})| = 1$.*

Demostración:
Sea $n \geq 1$. Para construir una computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) = 1$, en el primer paso de la computación $\mathcal{C}$ deberán ejecutarse obligatoriamente las reglas $r_6 \equiv ac \rightarrow bc\delta$ y $r_7 \equiv f \rightarrow ff$ (de manera maximal). Es decir, para conseguir que $\delta(\mathcal{C}, 5) = 1$, el único multiconjunto de aplicabilidad que se podrá aplicar a la configuración inicial $\mathcal{C}_0$ es $m_1 = r_6 r_7$. Por tanto,

$$\mathcal{C}_1 = m_1(\mathcal{C}_0) = (\mu_1, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-1}cbf^2))$$

siendo $\mu_1 = (1, ((1,2), (2,3,4), (3,2), (4,2)))$. Ahora bien, el único multiconjunto de aplicabilidad sobre $\mathcal{C}_1$ es $m_2 = r_4$. Por tanto, $\mathcal{C}_2 = m_2(\mathcal{C}_1) = (\mu_2, (\emptyset, a^{n-1}cbg, \emptyset))$,

siendo $\mu_2 = (1,((1,2),(2,1,4),(4,2)))$. Entonces, el único multiconjunto de aplicabilidad sobre $\mathcal{C}_2$ es $m_3 = r_1 r_3$. Por tanto, $\mathcal{C}_3 = m_3(\mathcal{C}_2) = (\mu_3, (a^n cb, e))$, siendo $\mu_3 = (1,((1,4),(4,1)))$.

Teniendo presente que las membranas 1 y 4 carecen de reglas, resulta que la configuración $\mathcal{C}_3$ es de parada. Más aún, como $i_0 = 4 \in \mu_3$, resulta que $\mathcal{C}_3$ es una configuración exitosa. Por tanto, $|C| = 3$ y la salida de la computación $\mathcal{C}$ verifica que $|\mathcal{O}(\mathcal{C})| = |\mathcal{C}_3(4)| = |\{e\}| = 1$.

Obsérvese que de acuerdo con la construcción realizada, la computación $\mathcal{C}$ de $\Pi(n)$ verificando la condición $\mathcal{C}_1(5)\uparrow$ es única.

□

A continuación vamos a probar que el punto crítico de las computaciones del P sistema $\Pi(n)$, con $n \geq 1$, se produce en el instante en que la membrana 5 se disuelve.

Proposición 7.23 *Sea $n \geq 1$. Sea p tal que $2 \leq p \leq n$. Para cada computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C},5) = p$, se verifica:*

1. $\mathcal{C}_p = (\mu', (\emptyset, \emptyset, a^{n-p} cb^p f^{2^p}, \emptyset))$, *siendo* $\mu' = (1,((1,2),(2,1,3,4),(3,2),(4,2)))$.
2. *Para cada k tal que $0 \leq k \leq p-1$ se tiene que tras $p+1+k$ pasos, se verifica que* $\mathcal{C}_{p+1+k} = (\mu'', (\emptyset, a^{n-p} cb^p g^{2^{p-1-k}}, e^{kp}))$, *siendo μ'' la estructura de membranas* $(1,((1,2),(2,1,4),(4,2)))$.
3. $\mathcal{C}_{2p+1} = (\mu''', (a^{n-p+1} cb^p, e^{p^2}))$, *siendo* $\mu''' = (1,((1,4),(4,1)))$.
4. *La computación $\mathcal{C}$ es exitosa, su longitud es $|\mathcal{C}| = 2p+1$, y, además, su salida verifica que $|\mathcal{O}(\mathcal{C})| = p^2$.*

Demostración:

1. Por inducción débil acotada ascendente sobre p. Para probar el caso base, $p = 2 \leq n$, consideremos una computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C},5) = 2$. Como $\delta(\mathcal{C},5) > 1$, de la proposición 7.19 se deduce que $\mathcal{C}_1 = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-1} cbf^2))$. Teniendo presente que $\delta(\mathcal{C},5) = 2$ y que $n \geq 2$, resulta que en el segundo paso de la computación $\mathcal{C}$ se ejecutarán obligatoriamente las reglas $r_6 \equiv ac \rightarrow bc\delta$ y $r_7 \equiv f \rightarrow ff$ (de manera maximal). Es decir, si $m = r_6 r_7^2$, entonces $\mathcal{C}_2 = m(\mathcal{C}_1) = (\mu', (\emptyset, \emptyset, a^{n-2} cb^2 f^{2^2}, \emptyset))$, siendo la estructura de membranas $\mu' = (1,((1,2),(2,1,3,4),(3,2),(4,2)))$.

   Sea $p \in \mathbf{N}$ tal que $2 \leq p < n$ y supongamos cierto el resultado para p. Sea $\mathcal{C}$ una computación de $\Pi(n)$ tal que $\delta(\mathcal{C},5) = p+1$. Teniendo presente

que $p < \delta(\mathcal{C}, 5)$ y que $p < n$, de la proposición 7.19 se deduce que $\mathcal{C}_p = (\mu_0, (\emptyset, \emptyset, \emptyset, \emptyset, a^{n-p}cb^p f^{2^p}))$. Como $\delta(\mathcal{C}, 5) = p+1$, resulta que en el paso $(p+1)$–ésimo de la computación $\mathcal{C}$, la membrana 5 debe disolverse; es decir, en ese paso deberá ejecutarse obligatoriamente las reglas $r_6 \equiv ac \to bc\delta$ y $r_7 \equiv f \to ff$ (de manera maximal). Es decir, si $m = r_6 r_7^{2^p}$, entonces se verifica que $\mathcal{C}_{p+1} = m(\mathcal{C}_p) = (\mu', (\emptyset, \emptyset, a^{n-p-1}cb^{p+1}f^{2^{p+1}}, \emptyset))$, siendo μ' la estructura de membranas $(1, ((1,2), (2,1,3,4), (3,2), (4,2)))$.

2. Sea $n \geq 1$. Sea $p \in \mathbf{N}$ tal que $2 \leq p \leq n$. Sea $\mathcal{C}$ una computación de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) = p$. Veamos que para cada k tal que $0 \leq k \leq p-1$ se tiene que

$$\mathcal{C}_{p+1+k} = (\mu'', (\emptyset, a^{n-p}cb^p g^{2^{p-1-k}}, e^{kp}))$$

siendo $\mu'' = (1, ((1,2), (2,1,4), (4,2)))$. Probémoslo por inducción sobre k.

Para probar el caso base, $k = 0$, observemos en primer lugar que del apartado (1) resulta que $\mathcal{C}_p = (\mu', (\emptyset, \emptyset, a^{n-p}cb^p f^{2^p}, \emptyset))$, siendo la estructura de membranas $\mu' = (1, ((1,2), (2,1,3,4), (3,2), (4,2)))$.

Entonces, el único multiconjunto de aplicabilidad sobre $\mathcal{C}_p$ es $m = r_4^{2^{p-1}}$. Luego, $\mathcal{C}_{p+1} = m(\mathcal{C}_p) = (\mu'', (\emptyset, a^{n-p}cb^p g^{2^{p-1}}, \emptyset))$, siendo la estructura de membranas $\mu'' = (1, ((1,2), (2,1,4), (4,2)))$.

Sea k tal que $k < n-1$ y supongamos cierto el resultado para k. De la hipótesis de inducción se deduce que $\mathcal{C}_{p+1+k} = (\mu'', (\emptyset, a^{n-p}cb^p g^{2^{p-1-k}}, e^{kp}))$ siendo la estructura de membranas $\mu'' = (1, ((1,2), (2,1,4), (4,2)))$.

Como $p-1-k > 0$ y $p > 0$ resulta que el único multiconjunto de aplicabilidad sobre $\mathcal{C}_{p+1+k}$ es $m = r_1^p r_2^{2^{p-1-k-1}}$. Luego,

$$\begin{aligned} \mathcal{C}_{p+1+k+1} = m(\mathcal{C}_{p+1+k}) &= (\mu'', (\emptyset, a^{n-p}cb^p g^{2^{p-1-k-1}}, e^p e^{kp})) \\ &= (\mu'', (\emptyset, a^{n-p}cb^p g^{2^{p-1-(k+1)}}, e^{(k+1)p})) \end{aligned}$$

Por tanto, el resultado es válido para $k+1$.

3. Sea $n \geq 1$. Sea $p \in \mathbf{N}$ tal que $2 \leq p \leq n$. Sea $\mathcal{C}$ una computación de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) = p$. Del apartado (2) se deduce que tras $2p$ pasos, $\mathcal{C}_{2p} = (\mu'', (\emptyset, a^{n-p}cb^p g, e^{(p-1)p}))$, siendo $\mu'' = (1, ((1,2), (2,1,4), (4,2)))$.

En tal situación, el único multiconjunto de aplicabilidad sobre $\mathcal{C}_{2p}$ es $m = r_1^p r_3$. Luego,

$$\begin{aligned} \mathcal{C}_{2p+1} = m(\mathcal{C}_{2p}) &= (\mu''', (\emptyset, a^{n-p+1}cb^p, e^p e^{(p-1)p})) \\ &= (\mu''', (\emptyset, a^{n-p+1}cb^p, e^{p^2})) \end{aligned}$$

siendo la estructura de membranas $\mu''' = (1, ((1,4),(4,1)))$.

4. Sea $n \geq 1$. Sea $p \in \mathbf{N}$ tal que $2 \leq p \leq n$. Sea $\mathcal{C}$ una computación de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) = p$. Del apartado (3) se deduce que la configuración $\mathcal{C}_{2p+1}$ es de parada, ya que las únicas membranas con objetos son la 1 y la 4, y ambas carecen de reglas. Por tanto, $|\mathcal{C}| = 2p + 1$. Además, la computación $\mathcal{C}$ es exitosa, ya que es de parada y la membrana de salida es elemental y está en la configuración final $\mathcal{C}_{2p+1}$. Además, la salida de dicha computación verifica que $|\mathcal{O}(\mathcal{C})| = |\mathcal{C}_{2p+1}(4)| = |\{e^{p^2}\}| = p^2$.

□

Corolario 7.24 *Sea $n \geq 1$. Para cada p tal que $1 \leq p \leq n$ existe, a lo sumo, una computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) = p$.*

Demostración:
Sea $n \geq 1$. Si $p = 1$, entonces de la proposición 7.22 se deduce que existe una única computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) = p$.

Sea p tal que $2 \leq p \leq n$. Sean $\mathcal{C}, \mathcal{C}'$ computaciones de $\Pi(n)$ tales que $\delta(\mathcal{C}, 5) = \delta(\mathcal{C}', 5) = p$. En primer lugar, del apartado (4) de la proposición 7.23 resulta que $|\mathcal{C}| = |\mathcal{C}'| = 2p + 1$. Por otra parte, de la proposición 7.19 se deduce que $\forall k\ (0 \leq k < p \rightarrow \mathcal{C}_k = \mathcal{C}'_k)$. Veamos, finalmente, que $\forall k\ (p \leq k \leq 2p + 1 \rightarrow \mathcal{C}_k = \mathcal{C}'_k)$. En efecto:

- El caso $k = p$ sigue del apartado (1) de la proposición 7.22.
- El caso $p + 1 \leq k \leq 2p$ sigue del apartado (2) de la proposición 7.22.
- El caso $k = 2p + 1$ sigue del apartado (3) de la proposición 7.22.

En consecuencia, $\mathcal{C} = \mathcal{C}'$.

□

Seguidamente vamos a caracterizar las computaciones exitosas de Π a través del instante en que se disuelve la membrana 5.

Corolario 7.25 *Sea $n \geq 1$. Sea $\mathcal{C}$ una computación de $\Pi(n)$. Son equivalentes:*

(a) $\mathcal{C}$ es una computación exitosa.

(b) $1 \leq \delta(\mathcal{C}, 5) \leq n$.

(c) $1 \leq \delta(\mathcal{C}, 5) \leq n \wedge |\mathcal{C}| = 2\delta(\mathcal{C}, 5) + 1 \wedge |\mathcal{O}(\mathcal{C})| = \delta(\mathcal{C}, 5)^2$.

Demostración:
Sea $n \geq 1$. Sea $\mathcal{C}$ una computación exitosa de $\Pi(n)$. De la proposición 7.20 se deduce que $\delta(\mathcal{C}, 5) \leq n$.

Sea $\mathcal{C}$ una computación exitosa de $\Pi(n)$ tal que $1 \leq \delta(\mathcal{C}, 5) \leq n$. Si $\delta(\mathcal{C}, 5) = 1$, entonces de la proposición 7.22 se deduce que $|\mathcal{C}| = 3$ y $|O(\mathcal{C})| = 1$. Si $2 \leq \delta(\mathcal{C}, 5) \leq n$, entonces del apartado (4) de la proposición 7.23 se concluye que $|\mathcal{C}| = 2\delta(\mathcal{C}, 5) + 1 \wedge |\mathcal{O}(\mathcal{C})| = \delta(\mathcal{C}, 5)^2$.

Sea $\mathcal{C}$ una computación exitosa de $\Pi(n)$ tal que $1 \leq \delta(\mathcal{C}, 5) \leq n \wedge |\mathcal{C}| = 2\delta(\mathcal{C}, 5) + 1 \wedge |\mathcal{O}(\mathcal{C})| = \delta(\mathcal{C}, 5)^2$. Si $\delta(\mathcal{C}, 5) = 1$, entonces de la proposición 7.22 se deduce que $\mathcal{C}$ es exitosa. Si $2 \leq \delta(\mathcal{C}, 5) \leq n$, entonces del apartado (4) de la proposición 7.23 se concluye que la computación $\mathcal{C}$ es exitosa.

□

Corolario 7.26 *Para cada $n \geq 1$ existe una única computación, $\mathcal{C}$, de $\Pi(n)$ que no es de parada.*

Demostración:
Sea $n \geq 1$. Si $\mathcal{C}$ es una computación de $\Pi(n)$ que no es de parada, entonces no es exitosa y, por tanto, $\delta(\mathcal{C}, 5) > n$. De la proposición 7.20 se deduce que dicha computación existe y, además, es única.

□

7.3.2. Corrección y completitud

Para establecer que el P sistema de cálculo $\Pi(n)$ calcula la función parcial, f, de $\mathbf{N}$ en $P(\mathbf{N})$ definida por

$$f(n) = \begin{cases} \uparrow & \text{si } n = 0 \\ \{1^2, 2^2, \ldots, n^2\} & \text{si } n \neq 0 \end{cases}$$

hemos de probar las siguientes condiciones:

(a) Ninguna computación de Π con entrada $n = 0$ es exitosa.

(b) Si $n \geq 1$, entonces:

- Para cada computación exitosa, $\mathcal{C}$, de $\Pi(n)$ existe $p \in \mathbf{N}$ tal que $1 \leq p \leq n$ y la salida de la computación $\mathcal{C}$ codifica p^2 (*corrección*).

- Para cada $p \in \mathbf{N}$ tal que $1 \leq p \leq n$, existe una computación, $\mathcal{C}$, de $\Pi(n)$ que es exitosa y, además, su salida codifica p^2 (*completitud*).

Dicho con otras palabras, hemos de probar que ninguna computación de $\Pi(0)$ es de parada y, además, que para cada número natural $n \geq 1$ se verifica que $\mathbf{N}(\Pi(n)) \subseteq f(n)$ (*corrección*) y que $f(n) \subseteq \mathbf{N}(\Pi(n))$ (*completitud*).

Teorema 7.27 (Corrección) *Para cada $n \geq 1$ y cada computación exitosa, $\mathcal{C}$, de $\Pi(n)$, existe $p \in \mathbf{N}$ tal que $1 \leq p \leq n$ y, además, la salida de la computación $\mathcal{C}$ verifica que $|\mathcal{O}(\mathcal{C})| = p^2$. Es decir, $\forall n \geq 1$ $(\mathbf{N}(\Pi(n)) \subseteq f(n))$.*

Demostración:
Sea $n \geq 1$. Sea $\mathcal{C}$ una computación exitosa de $\Pi(n)$. Del corolario 7.26 se deduce que $1 \leq \delta(\mathcal{C},5) \leq n \wedge |\mathcal{C}| = 2\delta(\mathcal{C},5)+1 \wedge |\mathcal{O}(\mathcal{C})| = \delta(\mathcal{C},5)^2$. Si $\delta(\mathcal{C},5) = 1$, entonces de la proposición 7.22 resulta que $|\mathcal{O}(\mathcal{C})| = 1$. Si $\delta(\mathcal{C},5) = p$, con $2 \leq p \leq n$, entonces del apartado (4) de la proposición 7.22 se deduce que $|\mathcal{O}(\mathcal{C})| = p^2$.

□

Para establecer la completitud de Π consideremos la siguiente fórmula

$$\varphi(n,p) \equiv \exists \mathcal{C} \in \mathbf{Comp}(\Pi(n))\ (p = \delta(\mathcal{C},5))$$

Proposición 7.28 *Sea $n \geq 1$. Para cada $p \in \mathbf{N}$ tal que $1 \leq p \leq n$ existe una única computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C},5) = p$.*

Demostración:
Sea $n \geq 1$. Probemos la existencia de una tal computación por inducción sobre p.

El caso base, $p = 1$, se deduce directamente de la proposición 7.22.

Sea p tal que $1 \leq p < n$ y supongamos cierto el resultado para p. Sea $\mathcal{C}$ una computación de $\Pi(n)$ tal que $\delta(\mathcal{C},5) = p$. Como $0 \leq p-1 < p = \delta(\mathcal{C},5)$, de la proposición 7.19 resulta que $\mathcal{C}_{p-1} = (\mu_0, (\emptyset,\emptyset,\emptyset,\emptyset, a^{n-p+1}cb^{p-1}f^{2^{p-1}}))$. Consideremos el multiconjunto de aplicabilidad $m_1 = r_5 r_7^{2^{p-1}}$ sobre $\mathcal{C}_{p-1}$ y sea $\mathcal{C}'_p = m_1(\mathcal{C}_{p-1}) = (\mu_0, (\emptyset,\emptyset,\emptyset,\emptyset, a^{n-p}cb^p f^{2^p}))$ Consideremos el multiconjunto de aplicabilidad $m_2 = r_6 r_7^{2^p}$ sobre $\mathcal{C}'_p$ y sea

$$\mathcal{C}'_{p+1} = m_2(\mathcal{C}'_p) = (\mu', (\emptyset,\emptyset,\emptyset, a^{n-p-1}cb^{p+1}f^{2^{p+1}}, \emptyset)$$

siendo la estructura de membranas $\mu' = (1,((1,2),(2,1,3,4),(3,2),(4,2)))$.
Sea $\mathcal{C}'$ la computación de $\Pi(n)$: $\mathcal{C}_0 \Rightarrow_\Pi \mathcal{C}_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi \mathcal{C}'_p \Rightarrow_\Pi \mathcal{C}'_{p+1} \Rightarrow_\Pi \ldots$.

Esta computación verifica que $\delta(\mathcal{C}',5) = p+1$. Lo que prueba el resultado para $p+1$.

Finalmente, la unicidad de una computación, $\mathcal{C}$, de $\Pi(n)$ verificando que $\delta(\mathcal{C}, 5) = p$ $(1 \leq p \leq n)$ se deduce del corolario 7.24.

□

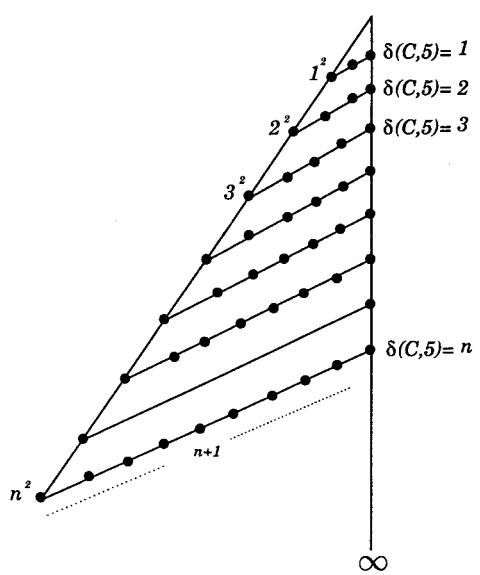
Corolario 7.29 *Para cada $n \geq 1$ se verifica que $\forall p\ (1 \leq p \leq n \rightarrow \varphi(n,p))$.*

Teorema 7.30 (Completitud) *Sea $n \geq 1$. Para cada $p \in \mathbf{N}$ tal que $1 \leq p \leq n$ existe una computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\mathcal{C}$ es exitosa y, además, su salida verifica que $|\mathcal{O}(\mathcal{C})| = p^2$. Es decir, $\forall n \geq 1\ (f(n) \subseteq \mathbf{N}(\Pi(n)))$.*

Demostración:
Sea $n \geq 1$. Sea p tal que $1 \leq p \leq n$. De la proposición 7.28 se deduce que existe una única computación, $\mathcal{C}$, de $\Pi(n)$ tal que $\delta(\mathcal{C}, 5) = p$. Si $p = 1$, entonces de la proposición 7.22 resulta que la computación $\mathcal{C}$ es exitosa y, además, $|\mathcal{O}(\mathcal{C})| = 1$. Si $2 \leq p \leq n$, entonces del apartado (4) de la proposición 7.23 se deduce que $\mathcal{C}$ es exitosa y, además, la salida verifica que $|\mathcal{O}(\mathcal{C})| = p^2$.

□

En resumen, podemos describir gráficamente el conjunto de *todas* las computaciones del P sistema $\Pi(n)$, para cada $n \geq 1$, como sigue:



Obsérvese que para cada $n \geq 1$, el P sistema $\Pi(n)$ tiene exactamente $n + 1$ computaciones, de las cuales sólo una no es de parada (y, por tanto, no es exitosa).

Capítulo 8

Verificación de P sistemas que resuelven problemas NP-completos

Este capítulo está dedicado al diseño y verificación de P sistemas de transición no deterministas que resuelven, en tiempo polinomial, el problema **SAT** de la satisfactibilidad de la lógica proposicional y el problema **HPP** del camino hamiltoniano en su versión dirigida y con dos nodos distinguidos.

En la primera sección del capítulo se hacen algunas consideraciones acerca de las soluciones deterministas y las soluciones no deterministas que pueden proporcionar los P sistemas.

En la segunda sección se presentan dos soluciones no deterministas del problema **SAT** mediante P sistemas de transición que lo resuelven en sentido fuerte: a cada fórmula proposicional se le asocia un P sistema que genera, básicamente, el conjunto de valoraciones que hacen satisfactible la fórmula.

En la última sección se presenta una solución no determinista del problema **HPP** a través de un P sistema de transición que lo resuelve en sentido fuerte: a cada grafo dirigido se le asocia un P sistema que genera exactamente los caminos hamiltonianos que son soluciones correctas del problema.

En las correspondientes secciones se establece la verificación de los P sistemas de transición diseñados en un intento de aportar ideas y conceptos que permitan sistematizar los procesos de verificación en P sistemas.

8.1. Soluciones no deterministas de problemas de decisión

Si un P sistema de transición determinista resuelve un problema de decisión, entonces no hay dudas acerca del significado de la respuesta que da el sistema, ya que al disponer de una única computación, el resultado de la misma *determina* la respuesta del sistema.

Sin embargo, cuando se usan P sistemas de transición no deterministas para resolver problemas de decisión, la interpretación de la respuesta que da el sistema no es tan directa. En estos casos, pueden existir distintas computaciones exitosas que devuelvan diferentes respuestas. Esta característica, que es aprovechada por los P sistemas de transición para generar lenguajes, puede ser poco deseable en el uso de los mismos como herramientas para resolver problemas de decisión.

Es por ello que, cuando se interpretan los P sistemas de transición como reconocedores de predicados, se suele introducir el concepto de *coherencia*, exigiendo que un P sistema de transición, para que pueda ser utilizado como reconocedor de un predicado, debe devolver una única respuesta independientemente de la computación resultante en caso de no determinismo. Es decir, las soluciones de un P sistema de transición no determinista y coherente proporciona, en realidad, una respuesta determinista.

Esta interpretación no coincide con la idea clásica del no determinismo, por la cual un dato de entrada del problema es aceptado si existe, al menos, una computación relativa a esa entrada que devuelve una respuesta positiva, y es rechazado en caso contrario.

Desde un punto de vista práctico, parece más natural la interpretación que hace uso de la coherencia ya que, en caso de que algún día se consiguiera una implementación de este modelo de computación, bastaría una computación concreta en el modelo para conocer la respuesta real (en caso de que estuviera verificado).

Las soluciones que presentamos de los problemas **SAT** y **HPP** hacen uso de la interpretación clásica de solución no determinista, entre otras razones, porque se pretende resolver el problema en dos direcciones. Por una parte, los P sistemas diseñados dan respuesta al problema de decisión planteado y, por otra, dichos P sistemas generan el lenguaje de todas las soluciones del dato de entrada.

Así, en el caso del problema **SAT**, dada una fórmula proposicional en FNC, se diseña un P sistema de transición asociado que genera el lenguaje de todas las valoraciones que hacen verdadera la fórmula (veremos que realmente no devuelve

todas las valoraciones, sino aquellos *trozos* de ellas que son precisos para verificar la fórmula). En el caso del problema del camino hamiltoniano, dado un grafo no dirigido y dos vértices distinguidos, se diseña un P sistema que genera todos los caminos hamiltonianos presentes en el grafo que van de uno de los vértices distinguidos al otro.

Hasta ahora se han presentado *soluciones deterministas* que resuelven los problemas **SAT** y **HPP** en tiempo polinomial, usando variantes de P sistemas que permiten, de alguna manera, conseguir espacio exponencial en un tiempo polinomial. Entre dichas soluciones deterministas destacamos las siguientes:

1. J. Castellanos, Gh. Păun y A. Rodríguez-Patón [14] resuelven el problema **SAT** en tiempo polinomial y el problema **HPP** en tiempo cuadrático, usando P sistemas que trabajan con cadenas en lugar de hacerlo con objetos atómicos (permitiendo procesar las cadenas a través de operaciones como replicación, splitting, mutación y recombinación).

2. Gh. Păun [57], S.N. Krishna y R. Rama [37] proporcionan soluciones deterministas en tiempo polinomial de los problemas **SAT** y **HPP** usando una variante de P sistemas que permite la división de cualquier tipo de membranas de la estructura. Este resultado ha sido mejorado por C. Zandron, C. Ferreti y G. Mauri [85] al obtener soluciones deterministas de complejidad similar pero usando únicamente la división para membranas elementales. Además, como hemos indicado en **6.5**, en [85] se prueba que si **P≠NP**, entonces los problemas **SAT** y **HPP** no pueden ser resueltos por P sistemas de transición deterministas en tiempo polinomial.

3. S. N Krishna y R. Rama [38] resuelven los problemas **SAT** y **HPP** en tiempo lineal, a través de P sistemas con reescritura replicada.

8.2. El problema SAT

En esta sección presentamos dos soluciones no deterministas, con sus correspondientes verificaciones, del problema **SAT** de la Lógica proposicional, a través de P sistemas de transición cooperativos que usan disolución y prioridad.

Definición 8.1 *Dada una fórmula proposicional φ, diremos que σ es una* valoración parcial *para φ si es una función parcial del conjunto de variables de φ, $Var(\varphi)$, en $\{0,1\}$.*

Diremos que una valoración parcial, σ, para φ, satisface *(no satisface) a φ, si toda valoración, σ', de φ que sea extensión de σ, verifica que $\sigma'(\varphi) = 1$ (resp., $\sigma'(\varphi) = 0$). Lo denotaremos por $\sigma(\varphi) = 1$ (resp. $\sigma(\varphi) = 0$).*

Nota 8.2 Sea φ una fórmula proposicional dada en forma normal conjuntiva (FNC), $\varphi = C_1 \wedge \ldots \wedge C_p$, donde C_i es una disyunción de literales y $Var(\varphi) = \{x_1, \ldots, x_n\}$. En este caso, podemos identificar cada cláusula C_i con un subconjunto de $Lit(\varphi) = \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$. Convendremos que $C_0 = \emptyset$.

Nota 8.3 Podemos identificar las valoraciones parciales para φ con subconjuntos *funcionales* de $Val(\varphi) = \{1, \ldots, n\} \times \{0, 1\}$. Habitualmente, al elemento $(i, a) \in Val(\varphi)$ lo notaremos por a_i. Es decir, $Val(\varphi) = \{1_i, 0_i : \ 1 \leq i \leq n\}$

Definición 8.4 *Se definen las aplicaciones v^+ y v^-, de $Lit(\varphi)$ en $Val(\varphi)$ como sigue:*

$$v^+(l) = \begin{cases} 1_i & , \text{si } l = x_i \\ 0_i & , \text{si } l = \bar{x}_i \end{cases} \qquad v^-(l) = \begin{cases} 0_i & , \text{si } l = x_i \\ 1_i & , \text{si } l = \bar{x}_i \end{cases}$$

Es decir, para cada literal, l, $v^+(l)$ (resp. $v^-(l)$) codifica el valor de verdad de la variable que interviene en el literal y que hace verdadero (resp. falso) dicho literal.

8.2.1. Diseño de un P sistema que resuelve SAT

A continuación se diseña una familia de P sistemas de transición cooperativos, con disolución y prioridad, que resuelven el problema **SAT** en el siguiente sentido *fuerte*:

A cada fórmula proposicional, φ, dada en FNC, se le asocia de manera unívoca, un P sistema de transición, Π_φ, que satisface las condiciones siguientes:

1. Para cada valoración, σ, asociada a φ tal que $\sigma(\varphi) = 1$, existe una computación exitosa, $\mathcal{C}$, de Π_φ, tal que en su salida, $\mathcal{O}(\mathcal{C})$, está codificada una valoración parcial, $\sigma_\mathcal{C}$, para φ verificando que $\sigma_\mathcal{C}(\varphi) = 1$ y $\sigma_\mathcal{C} \subseteq \sigma$.

2. Para cada computación exitosa, $\mathcal{C}$, de Π_φ, se tiene que en su salida, $\mathcal{O}(\mathcal{C})$, está codificada una valoración parcial, $\sigma_\mathcal{C}$, para φ tal que $\sigma_\mathcal{C}(\varphi) = 1$.

3. Para cada computación de parada y no exitosa de Π_φ, la configuración final codifica una asignación parcial, σ_f, para φ tal que $\sigma_f(\varphi) = 0$.

Sea $\Pi_\varphi = (A_\varphi, \mathcal{C}_0, \mathcal{R}, i_0)$ el P sistema de transición definido como sigue:

- El alfabeto de trabajo será $A_\varphi = Act \cup Lit(\varphi) \cup Val(\varphi)$, en donde $Act = \{T, T^*, F\}$, que llamaremos el conjunto de *activadores* (obsérvese que el alfabeto depende únicamente de las variables de φ).

- La configuración inicial, $\mathcal{C}_0 = (\mu_0, M_0)$, está definida como sigue:

  - La estructura de membranas inicial, $\mu_0 = (V_0, E_0)$, de grado $p+1$, viene dada por:

    $$V_0 = V(\mu_0) = \{0, \ldots, p\}, \ y \ E_0 = E(\mu_0) = \{(i, i+1) : \ 0 \leq i < n\}$$

    Es decir, $\mu_0 = (0, ((0,1), (1,0,2), (2,1,3), \ldots, (p, p-1)))$.

  - M_0 es la aplicación del conjunto de nodos $\{0, \ldots, p\}$ en $\mathbf{M}(A)$ definida como sigue:

    $$\forall i \ (0 \leq i \leq p \rightarrow M_0(i) = C_i), \ \text{y} \ M_0(p) = C_p \cup \{T\}$$

- $\mathcal{R}$ es un par ordenado (R, ρ), donde $R = \cup\{R_j : \ 0 \leq j \leq p\}$ es una colección de reglas de evolución asociadas a $\mathcal{C}_0$ y $\rho = \cup\{\rho_j : \ 0 \leq j \leq p\}$ es una colección de relaciones de prioridad sobre R.

  - Para cada nodo j, con $j > 0$, el conjunto de reglas de evolución, R_j, es el siguiente:

    $$R_j = \{r_1(l), \ r_2(l), \ r_3(l), \ r_4(l) : \ l \in Lit(\varphi)\} \cup \{r_5, r_6\}$$

    donde, para cada literal $l \in Lit(n)$, definimos informalmente:

    $$\begin{aligned} r_1(l) &\equiv \ Tlv^+(l) \rightarrow T^*v^+(l) \\ r_2(l) &\equiv \ Tlv^-(l) \rightarrow Tv^-(l) \\ r_3(l) &\equiv \ Tl \rightarrow T^*v^+(l) \\ r_4(l) &\equiv \ T^*l \rightarrow T^* \end{aligned}$$

    y $r_5 \equiv T^* \rightarrow T\delta$, $r_6 \equiv T \rightarrow F$.

    Con la notación formal dada anteriormente, las reglas de evolución R_j pueden describirse como sigue:

Regla	$\mathbf{d_r}$	$\mathbf{v_r(here)}$	$\delta_\mathbf{r}$	x_r
$r_1(l)$	$Tlv^+(l)$	$T^*v^+(l)$	$-$	j
$r_2(l)$	$Tlv^-(l)$	$Tv^-(l)$	$-$	j
$r_3(l)$	Tl	$T^*v^+(l)$	$-$	j
$r_4(l)$	T^*l	T^*	$-$	j
r_5	T^*	T	$+$	j
r_6	T	F	$-$	j

Para el nodo 0, el conjunto de reglas de evolución es vacío; es decir, $R_0 = \emptyset$ (y, por tanto, $\rho_0 = \emptyset$).

- La relación de prioridad, ρ_j, para cada nodo j ($j > 0$) viene dada por:

$$\{r_1(l)\}_{l \in Lit} > \{r_2(l)\}_{l \in Lit} > \{r_3(l)\}_{l \in Lit} > \{r_4(l)\}_{l \in Lit} > r_5 > r_6$$

Esto es, toda regla de la forma $r_1(l)$ tiene mayor prioridad que cada regla de la forma $r_2(l')$, toda regla de la forma $r_2(l)$ tiene mayor prioridad que cada regla de la forma $r_3(l')$, etc.

- La membrana de salida es la piel; es decir, $i_0 = 0$.

A continuación se describe una idea informal de cómo funciona el P sistema de transición anterior, analizando los distintos pasos que realiza dicho P sistema durante una computación genérica y mostrando algunas de las propiedades que satisface y que, posteriormente, demostraremos con todo detalle.

- Cada cláusula de la fórmula está asociada a una membrana del P sistema (salvo la piel).
- Una membrana tiene actividad (es decir, se aplica alguna regla en su interior en algún momento de la computación) si tiene activadores (obsérvese que no son catalizadores). En este caso, diremos que la membrana está *activa* en ese momento.
- En cada paso de la computación una única membrana está activa, por tanto, las transformaciones se producen en una sola membrana del P sistema. Esto quiere decir que el P sistema presentado no hace uso del paralelismo intrínseco a los P sistemas. Posteriormente veremos cómo se puede obtener un P sistema de transición a partir de éste que haga uso del paralelismo.
- Al principio de la computación la membrana activa es la etiquetada por p.
- Tras la disolución de una membrana activa, su padre pasa a ser la membrana activa. En este caso, tan sólo le son transmitidos el activador y los objetos de $Val(\varphi)$, que determinan una asignación parcial para φ.
- Cuando una membrana está activa se realizan las siguientes operaciones, encaminadas a extender, si es posible, la asignación parcial presente para verificar la cláusula asociada a dicha membrana:

1. Al comienzo de su actividad el activador presente es T.

2. Si la membrana activa tiene literales, entonces comprobamos, por medio de las reglas del tipo $r_1(l)$, si la asignación parcial presente satisface alguno de sus literales. Si es así, el activador T se transforma en T^* y la asignación parcial no cambia.

3. Si una membrana tiene el activador T^*, significa que la cláusula asociada ha sido satisfecha. En este caso, por medio de las reglas $r_4(l)$, se eliminan todos los literales presentes en la membrana. Tras ello, T^* es de nuevo transformado en T y se disuelve la membrana.

4. Si la asignación parcial no satisface la cláusula asociada a la membrana activa, por medio de las reglas $r_2(l)$ se eliminan todos los literales que estén asignados (ya que no aportan información relevante para una posible extensión de la asignación parcial que verifique la clásula). A continuación, si todavía quedan literales, por medio de las reglas $r_3(l)$ se extiende la asignación parcial con el fin de verificar la cláusula. Si no quedaran literales, entonces no hay forma de extender la asignación parcial presente para verificar la cláusula (y, por tanto, la fórmula). Entonces, T se transforma en F y la computación para pero no exitosamente (ya que la membrana de salida no es elemental).

- Si se disuelve la membrana 1, entonces todas las cláusulas han sido satisfechas y, por tanto, φ es satisfecha por la asignación parcial generada. En este caso, la membrana de salida es elemental y contiene el activador T, así como la asignación parcial que verifica φ. Como la membrana 0 no tiene reglas y es elemental, la computación obtenida es exitosa.

8.2.2. Verificación formal del P sistema diseñado

A continuación procedemos a establecer la verificación formal de Π_φ respecto a la satisfactibilidad de la fórmula φ.

El primer paso consistirá en probar que para cada asignación parcial σ que verifique $\sigma(\varphi) = 1$, existe una computación exitosa de Π_φ de forma que su salida codifica una asignación parcial, σ', verificando la fórmula φ y tal que $\sigma' \subseteq \sigma$. Para ello, consideramos la siguiente fórmula, que será, en cierto sentido, un invariante del P sistema a lo largo de la ejecución del mismo.

Definición 8.5 *Si φ es satisfactible y σ es una valoración que la hace válida, para cada i tal que $1 \leq i \leq p$, sea $\theta_\sigma(i)$ la fórmula:*

Existen $s_i \in \mathbf{N}$ *y* $C \equiv C_0 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} C_{s_i}$, *computación parcial de* Π_φ, *tales que*

(a) $\delta(C, i) = s_i \wedge \forall j \ (i < j \leq p \rightarrow \delta(C, j) < s_i)$.

(b) $C_{s_i}(i-1) = C_{i-1} \cup \{T\} \cup \sigma_i$, *donde* $\sigma_i \subseteq \sigma$ *es una valoración parcial para* φ *tal que* $\forall j \ (i \leq j \leq p \rightarrow \sigma_i(C_j) = 1)$.

Queremos probar que si la fórmula es satisfactible por una asignación σ, entonces existe una computación exitosa que genera una asignación parcial que también verifica la fórmula y que es restricción de σ. Para ello, veamos en primer lugar un lema técnico.

Lema 8.6 *Sea* φ *una fórmula satisfactible y* σ *una valoración para* φ *tal que* $\sigma(\varphi) = 1$. *Sea* i *tal que* $1 \leq i \leq p$. *Si* $C \equiv C_0 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} C_s$ *es una computación parcial de* Π_φ *tal que* $C_s(i) = C_i \cup \{T\} \cup \sigma_0$, *donde* $\sigma_0 \subseteq \sigma$ *valoración parcial para* φ, *entonces existen* $s' \in \mathbf{N}$, *con* $s' > s$, *y* $C' \equiv C_0 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} C_s \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} C_{s'}$, *computación parcial de* Π_φ *que es extensión de* C *y tal que* $\delta(C', i) = s'$ *y* $C'_{s'}(i-1) = C_{i-1} \cup \{T\} \cup \sigma'$, *donde* $\sigma_0 \subseteq \sigma' \subseteq \sigma$ *y* $\sigma'(C_i) = 1$.

Demostración:
Sea φ una fórmula satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$. Sea i tal que $1 \leq i \leq p$ y C una computación parcial de Π_φ verificando que $C_s(i) = C_i \cup \{T\} \cup \sigma_0$, siendo $\sigma_0 \subseteq \sigma$ una valoración parcial de φ.

Vamos a construir una extensión de C verificando las condiciones requeridas. Para ello, supongamos que $C_i = \{l_1, \ldots, l_{|C_i|}\}$, y definimos

$$C_i^+ = \{l \in C_i : \ v^+(l) \in \sigma_0\}, \ C_i^- = \{l \in C_i : \ v^-(l) \in \sigma_0\}$$

Puede ocurrir uno de los siguientes casos:

- $C_i^+ \neq \emptyset$. Es decir, existe algún literal de C_i que es verdadero por σ_0.

  Sin pérdida de generalidad, podemos suponer que $l_1 \in C_i^+$. Lo que hemos de hacer es aplicar la regla $Tl_1v^+(l_1) \rightarrow T^*v^+(l_1)$ y a continuación borrar, con las reglas $T^*l \rightarrow T^*$, los restantes literales. Sea C'_{s+1} la configuración que se obtiene de C_s al aplicar la regla $r_1(l_1) \equiv Tl_1v^+(l_1) \rightarrow T^*v^+(l_1)$. Sea C'_{s+j}, con $2 \leq j \leq |C_i|$, la configuración que se obtiene de C'_{s+j-1} por la aplicación de la regla $r_4(l_j) \equiv T^*l_j \rightarrow T^*$ (es decir, sea $m_1 = r_1(l_1)$, y para cada j tal que $2 \leq j \leq |C_i|$, definimos $m_j = r_4(l_j)$. Entonces, a través de estos multiconjuntos de aplicabilidad se define una extensión de la computación C

como sigue: si $\mathcal{C}'_{s+j} = m_j(\mathcal{C}'_{s+j-1})$, entonces se tiene que $\forall j\ (1 \leq j \leq |C_i| \rightarrow m_j \in \mathbf{M_{Ap}}(\mathcal{C}'_{s+j-1}))$ y, por tanto, definen una computación parcial para Π_φ).

Por último, sea $\mathcal{C}'_{s+|C_i|+1}$ la configuración que se obtiene de $\mathcal{C}'_{s+|C_i|}$ por la aplicación de la regla $r_5 \equiv T^* \rightarrow T\delta$. Si tomamos $s' = s + |C_i| + 1$, entonces $\delta(\mathcal{C}', i) = s'$ y $\mathcal{C}'_{s'}(i-1) = C_{i-1} \cup \{T\} \cup \sigma'$, donde $\sigma' = \sigma_0$ verifica las condiciones deseadas.

- $C_i^+ = \emptyset \wedge C_i^- = \emptyset$. Es decir, no hay literales en C_i que tengan una asignación por la valoración parcial σ_0.

  En este caso, como $\sigma(C_i) = 1$, existe un literal $l \in C_i$ tal que $\sigma(l) = 1$ y, por tanto, $v^+(l) \in \sigma$. Sin pérdida de generalidad, podemos suponer que $l_1 \in C_i$ y $v^+(l_1) \in \sigma$. Siguiendo un razonamiento similar al del caso anterior, es suficiente considerar la configuración $\mathcal{C}'_{s+1}$ que se obtiene de $\mathcal{C}_s$ por la aplicación de la regla $r_3(l_1) \equiv Tl_1 \rightarrow T^*v^+(l_1)$, y para cada j tal que $2 \leq j \leq |C_i| + 1$, se considera $\mathcal{C}'_{s+j}$ como en el caso anterior. Tomando $s' = s + |C_i| + 1$, resulta que $\delta(\mathcal{C}', i) = s'$ y $\mathcal{C}'_{s'}(i-1) = C_{i-1} \cup \{T\} \cup \sigma'$, donde $\sigma' = \sigma_0 \cup \{v^+(l_1)\}$ verifica las condiciones deseadas.

- $C_i^+ = \emptyset \wedge C_i^- \neq \emptyset$. Es decir, no hay literales en C_i con una asignación por la valoración parcial σ_0 que los haga verdaderos, pero existen literales en C_i que tienen una asignación por σ_0 que los hace falsos.

  De nuevo, sin pérdida de generalidad, podemos suponer que $C_i^- = \{l_1, \ldots, l_r\}$ y $\sigma(l_{r+1}) = 1$ (ya que, al menos, debe haber un literal en C_i verificando lo anterior, y como $\sigma_0 \subseteq \sigma$, este literal no es de C_i^-). Para conseguir una extensión adecuada de $\mathcal{C}$ se eliminan los literales de C_i^- por medio de las reglas r_2 y se añade la asignación adecuada para l_{r+1}. Finalmente, se eliminan los restantes literales por medio de las reglas r_4.

  Entonces se sigue un razonamiento similar al del primer caso. Consideramos para cada j tal que $1 \leq j \leq r$, la configuración $\mathcal{C}'_{s+j}$ que se obtiene de $\mathcal{C}'_{s+j-1}$ aplicando la regla $r_2(l_j) \equiv Tl_jv^-(l_j) \rightarrow Tv^-(l_j)$. Sea $\mathcal{C}'_{s+r+1}$ la configuración que se obtiene de $\mathcal{C}'_{s+r}$ aplicando la regla $r_3(l_{r+1}) \equiv Tl_{r+1} \rightarrow T^*v^+(l_{r+1})$. Para cada j tal que $r+1 < j \leq |C_i|$, sea $\mathcal{C}'_{s+j}$ la configuración que se obtiene de $\mathcal{C}'_{s+j-1}$ aplicando la regla $r_4(l_j) \equiv T^*l_j \rightarrow T^*$. Sea $\mathcal{C}'_{s+|C_i|+1}$ la configuración que se obtiene de $\mathcal{C}'_{s+|C_i|}$ aplicando la regla $r_5 \equiv T^* \rightarrow T\delta$. Sea $s' = s + |C_i| + 1$. Entonces resulta que $\delta(\mathcal{C}', i) = s'$ y $\mathcal{C}'_{s'}(i-1) = C_{i-1} \cup \{T\} \cup \sigma'$, donde $\sigma' = \sigma_0 \cup \{v^+(l_{r+1})\}$ verifica las condiciones deseadas.

□

Teorema 8.7 *La fórmula θ_σ es un invariante del P sistema Π_φ, en el sentido siguiente*

$$\forall i\ (1 \leq i \leq p \rightarrow \theta_\sigma(i))$$

Demostración:
Vamos a probarlo por inducción descendente sobre i.

Para probar el caso base, $i = p$, comencemos observando que $\mathcal{C} \equiv \mathcal{C}_0$ es una computación parcial de Π_φ tal que $\mathcal{C}_0(p) = C_p \cup \{T\} \cup \sigma_0$, en donde $\sigma_0 = \emptyset$. Del lema 8.6 se deduce la existencia de $s' \in \mathbf{N}$, $s' > 0$, y de $\mathcal{C}' \equiv \mathcal{C}_0 \Rightarrow_{\Pi_\varphi} \mathcal{C}'_1 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} \mathcal{C}'_{s'}$, computación parcial de Π_φ que es extensión de $\mathcal{C}$, tales que $\delta(\mathcal{C}', p) = s' \wedge \mathcal{C}'_{s'}(p-1) = C_{p-1} \cup \{T\} \cup \sigma'$, donde $\sigma' \subseteq \sigma$ y $\sigma'(C_p) = 1$. Por tanto, la fórmula $\theta_\sigma(p)$ es verdadera.

Sea i tal que $1 < i \leq p$ y supongamos que la fórmula $\theta_\sigma(i)$ es verdadera. Entonces existen $s_i \in \mathbf{N}$ y $\mathcal{C} \equiv \mathcal{C}_0 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} \mathcal{C}_{s_i}$, computación parcial de Π_φ, tales que

(a) $\delta(\mathcal{C}, i) = s_i \wedge \forall j\ (i < j \leq p \rightarrow \delta(\mathcal{C}, j) = s_i)$.

(b) $\mathcal{C}_{s_i}(i-1) = C_{i-1} \cup \{T\} \cup \sigma_i$, donde $\sigma_i \subseteq \sigma$ es una valoración parcial para φ tal que $\forall j\ (i \leq j \leq p \rightarrow \sigma_i(C_j) = 1)$.

Del lema 8.6 se deduce la existencia de $s' \in \mathbf{N}$, $s' > s_i$, y de $\mathcal{C}' \equiv \mathcal{C}_0 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} \mathcal{C}_{s_i} \Rightarrow_{\Pi_\varphi} \mathcal{C}'_{s_i+1} \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} \mathcal{C}'_{s'}$, computación parcial de Π_φ que es extensión de $\mathcal{C}$ y tal que $\delta(\mathcal{C}', i-1) = s'$ y $\mathcal{C}'_{s'}(i-2) = C_{i-2} \cup \{T\} \cup \sigma'$, donde $\sigma_i \subseteq \sigma' \subseteq \sigma$ y $\sigma'(C_{i-1}) = 1$. Teniendo presente que $\forall j\ (i-1 < j \leq p \rightarrow \delta(\mathcal{C}', j) < s')$ se concluye que la fórmula $\theta_\sigma(i-1)$ es verdadera.

□

Como consecuencia de este teorema se establece la existencia de computaciones exitosas verificando las condiciones deseadas.

Corolario 8.8 *Si la fórmula φ es satisfactible, entonces existe una computación, $\mathcal{C}$, de Π_φ verificando que:*

(a) $\mathcal{C}$ es exitosa.

(b) $\forall i\ (1 \leq i \leq p \rightarrow \delta(\mathcal{C}, i) < \infty)$.

(c) $\mathcal{O}(\mathcal{C}) = \{T\} \cup \sigma_{\mathcal{C}}$, *donde* $\sigma_{\mathcal{C}}$ *es una valoración parcial para* φ *tal que* $\sigma_{\mathcal{C}}(\varphi) = 1$.

Demostración:
Si φ es satisfactible, entonces existe una valoración, σ, para φ tal que $\sigma(\varphi) = 1$. Del teorema 8.7 resulta que se verifica $\theta_\sigma(1)$. Por tanto:

> Existen $s_1 \in \mathbf{N}$ y $\mathcal{C} \equiv \mathcal{C}_0 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} \mathcal{C}_{s_1}$, computación parcial de Π_φ, tales que
>
> (a) $\delta(\mathcal{C}, 1) = s_1 \wedge \forall j \ (1 < j \leq p \rightarrow \delta(\mathcal{C}, j) < s_1)$.
>
> (b) $\mathcal{C}_{s_1}(0) = \mathcal{C}_0 \cup \{T\} \cup \sigma_1 = \{T\} \cup \sigma_1$, en donde $\sigma_1 \subseteq \sigma$ es una valoración parcial para φ tal que $\forall j (1 \leq j \leq p \rightarrow \sigma_1(C_j) = 1)$.

Como la membrana 0 carece de reglas y todas las demás membranas han sido disueltas, la computación anterior es exitosa para Π_φ y, además, verifica las condiciones del enunciado. Entonces, basta tomar $\sigma_{\mathcal{C}} = \sigma_1$.

□

Nota 8.9 Téngase presente que se ha probado la siguiente propiedad: *para cada valoración,* σ, *que satisface* φ, *existe, al menos, una computación exitosa de* Π_φ *tal que su salida codifica una valoración parcial que es restricción de* σ *y que también satisface* φ. Es decir, se ha probado que el lenguaje generado por Π_φ contiene, de forma implícita, todas las valoraciones que satisfacen la fórmula φ.

Nota 8.10 Obsérvese que si r es una regla de Π, entonces los elementos de *Act* actúan globalmente a modo de catalizadores (aunque no queda el mismo activador, en su lugar aparece otro del mismo conjunto y en la misma multiplicidad). Es decir, para la ejecución de una regla es necesaria la presencia de algún activador que puede ser, eventualmente, transformado en otro activador.

A continuación se establece un lema técnico que nos permite asegurar que las disoluciones en el P sistema se producen de las membranas más internas a las más externas, de forma progresiva.

Lema 8.11 *Sean* i, j *tales que* $1 \leq i < j \leq p$. *Sea* $\mathcal{C}$ *una computación parcial de* Π_φ *tal que* $\delta(\mathcal{C}, i) < \infty$. *Entonces* $\delta(\mathcal{C}, j) < \delta(\mathcal{C}, i)$.

Demostración:
Basta probar el resultado para el caso $j = i+1$. Para ello, supongamos que $\delta(\mathcal{C}, i) < \infty$. Para que la membrana i-ésima se disuelva es necesario que, a lo largo de la

computación, dicha membrana haya recibido un elemento activador (ya que la única regla de disolución es la regla r_5). Teniendo presente que inicialmente la membrana i no tiene elementos activadores, es necesario que alguna otra membrana le haya transmitido el activador. Observando las reglas presentes en el P sistema, se concluye que la única posibilidad consiste en que la membrana $i+1$ se lo haya transmitido por medio de su disolución en un paso anterior. Es decir, $\delta(\mathcal{C}, i+1) < \delta(\mathcal{C}, i)$.

□

Como resumen de todo lo anterior, podemos dar una caracterización de las computaciones exitosas del P sistema Π_φ.

Proposición 8.12 *Sea $\mathcal{C}$ una computación de Π_φ. Las siguientes condiciones son equivalentes:*

1. *$\mathcal{C}$ es exitosa.*
2. $\delta(\mathcal{C}, 1) < \infty$.
3. $\delta(\mathcal{C}, p) < \delta(\mathcal{C}, p-1) < \ldots < \delta(\mathcal{C}, 2) < \delta(\mathcal{C}, 1) < \infty$.

Demostración:
Supongamos que $\mathcal{C}$ es exitosa. Teniendo presente que la membrana de salida es la membrana 0, se debe verificar que dicha membrana es elemental en la configuración de parada. Por tanto, todas las membranas contenidas en ella deben haber sido disueltas en un paso anterior. En particular, $\delta(\mathcal{C}, 1) < \infty$.

Supongamos ahora que $\delta(\mathcal{C}, 1) < \infty$. Aplicando reiteradamente el lema 8.11 se obtiene que $\delta(\mathcal{C}, p) < \delta(\mathcal{C}, p-1) < \ldots < \delta(\mathcal{C}, 2) < \delta(\mathcal{C}, 1) < \infty$.

Por último, supongamos que $\delta(\mathcal{C}, p) < \delta(\mathcal{C}, p-1) < \ldots < \delta(\mathcal{C}, 2) < \delta(\mathcal{C}, 1) = n < \infty$. Teniendo presente que todas las membranas interiores a la membrana 0 se han disuelto, se tendrá que en el paso n de la computación, la membrana 0 es elemental. Puesto que la membrana 0 no tiene reglas asociadas, se deduce que la configuración obtenida es de parada y exitosa.

□

A continuación vamos a definir una fórmula que actúe a modo de invariante, en el mismo sentido descrito anteriormente, y de tal manera que de su veracidad se pueda deducir la completitud del P sistema diseñado.

Definición 8.13 *Para cada i tal que $1 \leq i \leq p$, consideramos la fórmula $\gamma_\sigma(i)$ siguiente:*

Para cada computación parcial, $\mathcal{C} \equiv \mathcal{C}_0 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} \mathcal{C}_s$, *de* Π_φ *verificando que* $\delta(\mathcal{C}, i) = s$, *se tiene que*

(a) $\forall j\ (i < j \leq p \rightarrow \delta(\mathcal{C}, j) < s)$.

(b) $\mathcal{C}_s(i-1) = C_{i-1} \cup \{T\} \cup \sigma_i$, *en donde* σ_i *es una valoración parcial para* φ *verificando que* $\forall j\ (i \leq j \leq p \rightarrow \sigma_i(C_j) = 1)$.

(c) $\forall j\ \forall k\ (j < i \wedge k \leq s \rightarrow \mathcal{C}_k(j) = \mathcal{C}_0(j))$.

Teorema 8.14 *La fórmula* γ_σ *es un invariante del P sistema. Es decir,*

$$\forall i\ (1 \leq i \leq p \rightarrow \gamma_\sigma(i))$$

Demostración:
Vamos a probarlo por inducción descendente sobre i.

- Para probar el caso base, $i = p$, procedemos como sigue:

  Sea $\mathcal{C}$ una computación parcial de Π_φ verificando que $\delta(\mathcal{C}, p) = s$. Si en este último paso de la computación la membrana p se ha disuelto, entonces ha de verificarse que $T^* \in \mathcal{C}_{s-1}(p)$ y, además, $Lit(\varphi) \cap \mathcal{C}_{s-1}(p) = \emptyset$ (ya que el multiconjunto de aplicabilidad por el que se ha obtenido $\mathcal{C}_s$ a partir de $\mathcal{C}_{s-1}$ debe haber sido $m = r_6$). Teniendo presente que $T^* \notin \mathcal{C}_0$, deben existir $k < s$ y $l \in C_p$ tales que, para obtener $\mathcal{C}_{k+1}$ se ha aplicado la regla $r_3(l)$ a $\mathcal{C}_k$ (en los siguientes pasos, hasta el $s-1$, las reglas han eliminado todos los demás literales presentes en la membrana por medio de reglas del tipo r_4). Por tanto, $\mathcal{C}_{s-1}(p-1) = C_{p-1} \cup \{T\} \cup \sigma_1$, donde $\sigma_1 = \{v^+(l)\}$. Luego $\sigma_1(C_p) = 1$.

- Sea i tal que $1 < i \leq p$ y supongamos que la fórmula $\gamma_\sigma(i)$ es verdadera. Veamos que también es verdadera la fórmula $\gamma_\sigma(i-1)$.

  Para ello, sea $\mathcal{C} \equiv \mathcal{C}_0 \Rightarrow_\Pi \ldots \Rightarrow_\Pi \mathcal{C}_s$ una computación parcial de Π tal que $\delta(\mathcal{C}, i-1) = s$. Por el lema 8.11, se tiene que $\delta(\mathcal{C}, i) = s_i < s$. Por tanto, la computación $\mathcal{C}' \equiv \mathcal{C}_0 \Rightarrow_\Pi \ldots \Rightarrow_\Pi \mathcal{C}_{s_i}$ satisface las condiciones de la hipótesis de inducción; es decir,

  (a) $\forall j\ (i < j \leq p \rightarrow \delta(\mathcal{C}', j) < s_i)$.

  (b) $\mathcal{C}'_{s_i}(i-1) = C_{i-1} \cup \{T\} \cup \sigma_i$, en donde σ_i es una valoración parcial para φ verificando que $\forall j\ (i \leq j \leq p \rightarrow \sigma_i(C_j) = 1)$.

  Puesto que $\mathcal{C}$ es una extensión de $\mathcal{C}'$, del lema 8.11 se deduce que

(a') $\forall j\ (i-1 < j \leq p \rightarrow \delta(\mathcal{C}, j) < s)$.

Si $\delta(\mathcal{C}, i) = s < \infty$, entonces en el último paso de la computación $\mathcal{C}$ la membrana i se ha disuelto. Por tanto ha de verificarse que $T^* \in \mathcal{C}_{s-1}(i)$ y, además, $Lit(\varphi) \cap \mathcal{C}_{s-1}(i) = \emptyset$. Ahora bien, por hipótesis de inducción, $T^* \notin \mathcal{C}_{s_i}(i)$. Luego existen $s_i < k < s$ y $l \in C_i$ tales que, para obtener $\mathcal{C}_{k+1}$ se ha aplicado la regla $r_1(l)$ o $r_3(l)$ a $\mathcal{C}_k$ (en los siguientes pasos, hasta el $s-1$, las reglas han eliminado todos los demás literales presentes en la membrana). Por tanto, $\mathcal{C}_{s-1}(i-1) = C_{i-1} \cup \{T\} \cup \sigma_{i-1}$, donde $\sigma_{i-1} = \sigma_i \cup \{v^+(l)\}$. Luego $\sigma_{i-1}(C_i) = 1$. Obsérvese que, en este caso, $v^-(l) \notin \sigma_i$, ya que:

- Si se ha aplicado la regla $r_1(l)$, por hipótesis de inducción, σ_i es una valoración parcial para φ que contiene $v^+(l)$ y, en consecuencia, no puede contener $v^-(l)$.
- Si se ha aplicado la regla $r_3(l)$, quiere decir que no se ha podido aplicar la regla $r_2(l)$ y, por tanto, $v^-(l) \notin \sigma_{i-1}$.

Así pues, se tiene que:

(b') $\mathcal{C}_s(i-2) = C_{i-2} \cup \{T\} \cup \sigma_{i-1}$, en donde σ_{i-1} es una valoración parcial para φ verificando que $\forall j\ (i-1 \leq j \leq p \rightarrow \sigma_{i-1}(C_j) = 1)$.

Por tanto, se verifica $\gamma_\sigma(i-1)$.

□

Como consecuencia del teorema anterior, se deduce la completitud del P sistema de transición Π_φ.

Corolario 8.15 *Para cada computación exitosa, $\mathcal{C}$, de Π_φ, se tiene que en su salida, $\mathcal{O}(\mathcal{C})$, está codificada una valoración parcial, $\sigma_\mathcal{C}$, para φ tal que $\sigma_\mathcal{C}(\varphi) = 1$.*

Demostración:
Sea $\mathcal{C}$ una computación exitosa de Π_φ. Entonces $\delta(\mathcal{C}, 1) = k < \infty$. Como la fórmula $\gamma_\sigma(1)$ es verdadera, aplicando el teorema 8.14 se deduce que $\mathcal{C}_k(0) = \{T\} \cup \sigma_1$, donde σ_1 es una valoración parcial para φ verificando que $\forall j\ (1 \leq j \leq p \rightarrow \sigma_1(C_j) = 1)$. Es decir, $\sigma_1(\varphi) = 1$. Entonces basta tomar $\sigma_\mathcal{C} = \sigma_1$.

□

Finalmente se complementa el estudio del P sistema Π_φ caracterizando sus computaciones no exitosas.

Proposición 8.16 *Sea C una computación del P sistema Π_φ tal que es de parada pero no es exitosa. Sea $k = |\mathcal{C}|$. Entonces existe i tal que $1 \leq i < p$ y, además, verifica:*

1. $\delta(\mathcal{C}, i)\uparrow$.
2. $\forall j\ (i+1 \leq j \leq p \rightarrow \delta(\mathcal{C}, j) < \infty)$.
3. $\mathcal{C}_k(i) = \{F\} \cup \sigma_F$, *donde σ_F es una asignación parcial para φ verificando que* $\forall j\ (i < j \leq p \rightarrow \sigma_F(C_j) = 1)$ y $\sigma_F(C_i) = 0$.

Demostración:
Teniendo presente que $\mathcal{C}$ no es una computación exitosa de Π_φ, de la proposición 8.12 se deduce que $\delta(\mathcal{C}, 1)\uparrow$. Sea $i = \text{máx}\{j : 1 \leq j \leq p \wedge \delta(\mathcal{C}, j)\uparrow\}$. Entonces, por la elección de i, y aplicando el lema 8.11, se tiene que:

1. $\delta(\mathcal{C}, i)\uparrow$.
2. $\forall j\ (i < j \leq p \rightarrow \delta(\mathcal{C}, j) < \infty)$.

Veamos que $\mathcal{C}_k(i) = \{F\} \cup \sigma_F$, donde σ_F es una asignación parcial para φ verificando que $\forall j\ (i < j \leq p \rightarrow \sigma_F(C_j) = 1)$ y $\sigma_F(C_i) = 0$.

Para ello, consideremos la computación parcial $\mathcal{C}_0 \Rightarrow_{\Pi_\varphi} \ldots \Rightarrow_{\Pi_\varphi} \mathcal{C}_s$ tal que $s = \delta(\mathcal{C}, i+1) < \infty$. Teniendo presente que la fórmula $\gamma_\sigma(i+1)$ es verdadera, se deduce que $\mathcal{C}_s(i) = C_i \cup \{T\} \cup \sigma_{i+1}$, en donde σ_{i+1} es una valoración parcial para φ verificando que $\forall j\ (i+1 \leq j \leq p \rightarrow \sigma_{i+1}(C_j) = 1)$.

Como $\delta(\mathcal{C}, i)\uparrow$, resulta que la membrana i nunca se disuelve. Por tanto, se verifica que $\forall n\ (s \leq n \leq k \rightarrow T^* \notin \mathcal{C}_n(i))$ ya que, en caso contrario, tras un número finito de pasos en los que se aplicaría la regla r_4, deberíamos aplicar la regla r_5 provocando la disolución de la membrana i. Ahora bien, si $T \in \mathcal{C}_s(i)$, entonces las únicas reglas que se han podido aplicar entre los pasos s y k son las de tipo $r_2(l)$, con $l \in C_i$. Luego se verifica que $\forall l \in C_i\ (v^-(l) \in \sigma_{i+1})$. Es decir, $\sigma_{i+1}(C_i) = 0$. De donde se deduce que $\mathcal{C}_{s+|C_i|}(i) = \{T\} \cup \sigma_{i+1}$.

Por tanto, $\mathbf{M_{Ap}}(\mathcal{C}_{s+|C_i|}) = \{r_6\}$ y $\mathcal{C}_k = r_6(\mathcal{C}_{s+|C_i|}) = \{F\} \cup \sigma_F$, en donde $\sigma_F = \sigma_{i+1}$ verifica que $\forall j\ (i < j \leq p \rightarrow \sigma_F(C_j) = 1)$ y $\sigma_F(C_i) = 0$.

□

8.2.3. Análisis de la complejidad del P sistema

En primer lugar, vamos a calcular el tiempo de ejecución de cualquier computación exitosa de Π_φ. Por la proposición 8.12, sabemos que en dicha ejecución todas las membranas han sido disueltas. Sea r_i el número de literales de la cláusula C_i. La membrana p se disuelve en $1+r_p$ pasos, y una vez disuelta la membrana i de Π_φ se precisan $1+r_{i-1}$ pasos para disolver la membrana $i-1$. Por tanto, el número total de pasos de cualquier computación exitosa es $p+(r_1+\cdots+r_p) \in O(p \cdot n)$, en donde p es el número de literales y n es el número de variables de φ.

Es interesante observar que el número total de pasos obtenido es una cota superior de la longitud de cualquier computación del P sistema, sea exitosa o no, ya que en caso de no ser exitosa, el sistema se detiene en un paso anterior.

A continuación vamos a probar que el P sistema Π_φ puede ser construido a partir de la fórmula proposicional φ en un tiempo polinomial en el número total de variables y literales de φ. Para ello observemos que:

1. El número de símbolos del alfabeto es $|A_\varphi| = 4n+3$.

2. El número total de membranas de Π_φ es $p+1$.

3. El número total de reglas de evolución es $4(r_1+\cdots+r_p)+2p \in O(p \cdot n)$, y la longitud de cada regla es menor o igual a 5.

4. El número total de relaciones de prioridad es del orden de $O((r_1+\cdots+r_p)^2) = O((p \cdot n)^2)$.

Es interesante analizar el número de posibles computaciones que se realiza en el P sistema Π_φ. El caso mejor se dará cuando cada variable de φ esté asociada a algún literal de la cláusula C_p. En este caso $r_p = n$ y, además, el número total de computaciones es r_p.

El caso peor se dará cuando sean disjuntos entre sí los conjuntos de variables asociados a dos cláusulas distintas arbitrarias. En este caso, se ha de verificar que $r_1+\cdots+r_p = n$. Además, el número total de computaciones del P sistema Π_φ será $r_1 \cdot r_2 \cdot \cdots \cdot r_p \in O(n^p)$.

De este último análisis que se ha realizado conviene resaltar que el número total de computaciones del P sistema Π_φ depende de la estructura sintáctica de la fórmula. Esto se debe a que la búsqueda de soluciones correctas no se ha realizado a través de un mecanismo de fuerza bruta, sino mediante una estrategia basada en el contenido de las cláusulas, ordenadas desde p hasta 1.

8.2.4. Posibles mejoras

Los resultados anteriores pueden ser mejorados, en cierto sentido, introduciendo el paralelismo en la ejecución del P sistema. Cuando T^* está en una membrana (y, por tanto, la cláusula asociada es satisfecha por la asignación parcial generada) se usa este activador para eliminar todos los literales de la membrana. Introduciendo elementos del tipo T y T^* (en total $2n$) se pueden eliminar todos los literales en un sólo paso. Además, cuando se eliminan literales por medio de reglas del tipo r_2, usamos el activador T. Este proceso puede ser realizado en un sólo paso introduciendo algunos elementos adicionales de control. Una vez realizados estos cambios, conseguimos un P sistema de transición que usa $O(p)$ pasos para cualquier posible computación.

A continuación, veamos que es posible construir un P sistema de transición cooperativo, sin disolución y con prioridad, no determinista que resuelva **SAT** y en el que toda computación sea de parada y realice 2 pasos. Para ello, consideramos un P sistema de transición, Π'_φ, asociado a cada fórmula proposicional φ en FNC, en el que toda computación sea exitosa: algunas de ellas tendrán Y como salida (*de aceptación*) y otras N(*de rechazo*).

Sea $\Pi'_\varphi = (A, C_0, \mathcal{R}, i_0)$ el siguiente P sistema de transición:

- El alfabeto base es $A = \{T, Y, N\} \cup Lit(\varphi) \cup Val(\varphi)$.
- La configuración inicial $C_0 = (\mu_0, M_0)$ está definida como sigue:
  - La estructura de membranas inicial, $\mu_0 = (V_0, E_0)$, de grado $p+2$, viene dada por:

    $$V_0 = \{0, \ldots, p+1\},\ E_0 = \{(0, i) :\ 1 \leq i \leq p+1\}$$

    Es decir, $\mu_0 = (0, ((0,1), (1,0), (2,0), \ldots, (p+1, 0)))$.
  - M_0 es la aplicación del conjunto de nodos $\{0, \ldots, p+1\}$ en $\mathbf{M}(A)$ definida como sigue:

    $$\forall i\ (1 \leq i \leq p \rightarrow M_0(i) = C_i \cup \{T\}),\ \text{y}\ M_0(0) = M_0(p+1) = \emptyset$$

- $\mathcal{R}$ es un par ordenado (R, ρ), en donde $R = \cup\{R_j :\ 0 \leq j \leq p+1\}$ es una colección de reglas de evolución asociadas a C_0 y $\rho = \cup\{\rho_j :\ 0 \leq j \leq p+1\}$ es una colección de relaciones de prioridad sobre R.
  - Para cada nodo j tal que $1 \leq j \leq p$, consideramos el conjunto de reglas $R_j = \{Tl \rightarrow (Tv^+(l), out) :\ l \in Lit(\varphi)\}$, sin relaciones de prioridad ($\rho_j = \emptyset$)

- Además, $R_0 = \{T^p 1_i 0_i \rightarrow (N, in_{p+1}) : 1 \leq i \leq n\} \cup \{T^p \rightarrow (Y, in_{p+1})\}$, con la prioridad, ρ_1, $T^p 1_i 0_i \rightarrow (N, in_{p+1}) > T^p \rightarrow (Y, in_{p+1})$, para cada i tal que $1 \leq i \leq n$.
- Para el nodo $p+1$, consideramos $R_{p+1} = \emptyset$ y, por tanto, $\rho_{p+1} = \emptyset$.

Con la notación formal las reglas de evolución pueden describirse como sigue:

Regla	$\mathbf{d_r}$	$\mathbf{v_r(out)}$	$\mathbf{v_r(p+1)}$	δ_r	x_r
$r_j(l)$	Tl	$Tv^+(l)$	–	–	j
r_{0i}	$T^p 1_i 0_i$	–	N	–	0
r_Y	T^p	–	Y	–	0

- La membrana de salida es $i_0 = p+1$.

El siguiente teorema establece la verificación formal de este P sistema de transición.

Teorema 8.17 *Sea φ una forma proposicional dada en FNC y Π'_φ el P sistema de transición antes descrito. Las condiciones siguientes son equivalentes:*

1. *φ es satisfactible.*
2. *Existe una computación de aceptación de Π'_φ.*

Demostración:
Sea $\varphi \equiv C_1 \wedge \ldots \wedge C_p$, donde $C_j = l_{j,1} \vee \ldots \vee l_{j,r_j}$, con $1 \leq j \leq p$.

- Supongamos que φ es satisfactible. Sea σ una asignación tal que $\sigma(\varphi) = 1$. Para cada j tal que $1 \leq j \leq p$, sea $i_j = \text{mín}\{i : \sigma(l_{j,i}) = 1\}$. Sea $\mathcal{C}_1$ la configuración obtenida de $\mathcal{C}_0$ aplicando, en cada membrana j tal que $1 \leq j \leq p$, la regla $Tl_{j,i_j} \rightarrow (Tv^+(l_{j,i_j}), out)$. Como $\sigma(l_{j,i_j}) = 1$ $(1 \leq j \leq p)$, se tiene que $v^+(l_{j,i_j})$ codifica el valor que σ asigna a la variable asociada con l_{j,i_j} y que lo hace verdadero. Entonces $\sigma' = \{v^+(l_{1,i_1}), \ldots, v^+(l_{p,i_p})\} \subseteq \sigma$ y, por tanto, σ' es una asignación parcial para φ. De donde se deduce que $\{i : 1 \leq i \leq n \wedge \{1_i, 0_i\} \subseteq \sigma'\} = \emptyset$. En consecuencia, en la piel no se puede aplicar una regla del tipo $T^p 1_i 0_i \rightarrow (N, in_{p+1})$. Teniendo presente que $\mathcal{C}_1(0) = \{T^p\} \cup \sigma'$, que $\forall j\ (1 \leq j \leq p \rightarrow T \notin \mathcal{C}_1(j))$ y que $\mathcal{C}_1(p+1) = \emptyset$, resulta que la única regla que puede ser aplicada a $\mathcal{C}_1$ es $T^p \rightarrow (Y, in_{p+1})$. Si $\mathcal{C}_2$ es la configuración que se obtiene de $\mathcal{C}_1$ aplicando esta regla, entonces $\mathcal{C} \equiv \mathcal{C}_0 \Rightarrow_{\Pi'_\varphi} \mathcal{C}_1 \Rightarrow_{\Pi'_\varphi} \mathcal{C}_2$ es una computación exitosa de Π'_φ con salida $\mathcal{O}(\mathcal{C}) = \{Y\}$. Por tanto, la computación $\mathcal{C}$ es de aceptación de Π'_φ.

- Sea $\mathcal{C}$ una computación de aceptación para Π'_φ. Entonces $\mathcal{C}_1$ se ha obtenido de $\mathcal{C}_0$ por la aplicación en paralelo de reglas de la forma $Tl_j \rightarrow (Tv^+(l_j), out)$ en cada membrana j $(1 \leq j \leq p)$, en donde l_j es un literal de la cláusula C_j.
  Como la computación $\mathcal{C}$ es de aceptación, no puede existir i tal que $1 \leq i \leq p$ y $\{1_i, 0_i\} \subseteq \mathcal{C}_1(0)$. Por tanto, $\mathcal{C}_2$ se obtiene de $\mathcal{C}_1$ aplicando la regla $T^p \rightarrow (Y, in_{p+1})$. Es fácil probar que $\sigma' = \{v^+(l_1), \ldots, v^+(l_p)\}$ es una asignación parcial para φ verificando que $\forall j$ $(1 \leq j \leq p \rightarrow \sigma'(C_j) = 1)$. Por tanto, σ' verifica φ. Además, se verifica que $|\mathcal{C}| = 2$.

□

Analicemos el coste que se necesita para construir el P sistema de transición Π'_φ a partir de la fórmula φ.

1. El número de símbolos del alfabeto es $3 + 2n$.
2. El número total de membranas de Π_φ es $p + 2$.
3. El número total de reglas de evolución es $(r_1 + \cdots + r_p) + n + 1 \in O(p \cdot n)$, y la longitud de cada regla es menor o igual que $p + 4$.
4. El número total de relaciones de prioridad es n.

Además, el número total de posibles computaciones que se realizan en el P sistema Π'_φ es $r_1 \cdot \ldots \cdot r_p \in O(n^p)$. A diferencia de lo que sucedía en el sistema Π_φ, este número no depende de la estructura sintáctica de la fórmula.

Es interesante resaltar otra diferencia entre los dos métodos presentados para resolver el problema **SAT**. En el P sistema de transición Π'_φla estrategia usada es una traslación de la búsqueda exhaustiva sobre las posibles asignaciones para φ. En cambio en Π_φ se realiza una búsqueda que está condicionada por su estructura. El coste para construir Π_φ es ligeramente superior que el necesario para construir Π'_φ debido a las relaciones de prioridad entre las reglas. Ahora bien, Π_φ puede ser considerado como un P sistema más eficiente que Π'_φ, en el siguiente sentido: sobre las fórmulas satisfactibles, la razón entre las buenas computaciones (*exitosas* en Π_φ y *de aceptación* en Π'_φ) y el número total de las computaciones posibles, es mayor en Π_φ que en Π'_φ.

Finalmente observemos que en los dos P sistemas se trabaja con conjuntos uniformes de reglas en cada membrana, pero este hecho puede ser modificado con facilidad hacia un conjunto específico de reglas para cada membrana, considerando únicamente las reglas de los literales presentes en la cláusula asociada. No obstante, este proceso de reducción no altera los valores asintóticos de los costes obtenidos.

8.3. El problema del camino hamiltoniano

En esta sección se estudia un P sistema de transición no determinista, cooperativo, con disolución y sin prioridad, que resuelve el problema del camino hamiltoniano (**HPP**), en su versión dirigida y con dos nodos distinguidos. La solución que se presenta resuelve el problema **HPP** en un sentido similar al problema **SAT** en la sección anterior.

Es decir, a cada grafo dirigido, G, con dos nodos distinguidos v_i, v_f, se le asocia un P sistema de transición, Π_G, de tal manera que:

- El árbol de computación de Π_G es de profundidad acotada; es decir, toda computación de Π_G es de parada.
- Para cada camino hamiltoniano, γ, en G que va desde v_i hasta v_f, existe una computación exitosa de Π_G cuya salida codifica el camino γ.
- Toda computación exitosa de Π_G codifica en su salida un camino hamiltoniano de G que va desde v_i hasta v_f.
- Si una configuración de Π_G no es exitosa, entonces en una de las membranas está codificado un camino simple que comienza en v_i, pero que no llega a v_f.

8.3.1. Diseño de un P sistema que resuelve HPP

Sea $G = (V, E)$, con $V = \{v_0, \ldots, v_n\}$, un grafo dirigido Sean v_0 y v_n dos nodos distinguidos de G. Vamos a describir un P sistema, Π_G, que resuelve el problema del camino hamiltoniano para (G, v_0, v_n) de tal manera que el lenguaje generado por Π_G es el conjunto de caminos hamiltonianos de G que va desde v_0 hasta v_n.

Para ello, sea $\Pi_G = (A, C_0, \mathcal{R}, i_0)$ el P sistema de transición definido como sigue:

- El alfabeto base es $A = V \cup V' \cup \{0, \ldots, n\} \cup (V \times \{0, \ldots, n\})$, en donde $V' = \{v' : v \in V\}$ es un conjunto de *clones* de los vértices de G.
- La configuración inicial, $C_0 = (\mu_0, M_0)$, está definida como sigue:
  - La estructura de membranas inicial es $\mu_0 = (0, ((0, 1), (1, 0)))$.
  - M_0 es la aplicación del conjunto de nodos $\{0, 1\}$ en $\mathbf{M}(A)$ definida como sigue: $M_0(0) = (v_0, 0)$ y $M_0(1) = v_0' v_1 \ldots v_n 0$.
- $\mathcal{R}$ es un par ordenado (R, ρ), en donde $R = \{R_0, R_1\}$ es una colección de reglas de evolución asociadas a C_0 y $\rho = \{\rho_0, \rho_1\}$ es una colección de relaciones de prioridad.

- Para el nodo 0, la colección de reglas asociadas es $R_0 = \emptyset$ y $\rho_0 = \emptyset$.
- Para el nodo 1, la colección de reglas asociadas es $R_1 = \{r_{ijh} \equiv v_i' v_j h \rightarrow ((v_j, h+1), out) v_j'(h+1) : \ v_i v_j \in E, 0 \leq h < n\} \cup \{r_n \equiv v_n' n \rightarrow \delta\}$, y $\rho_1 = \emptyset$.

Con la notación formal las reglas de evolución pueden describirse como sigue:

Regla	$\mathbf{d_r}$	$\mathbf{v_r(here)}$	$\mathbf{v_r(out)}$	$\delta_\mathbf{r}$	x_r
r_{ijh}	$v_i' v_j h$	$v_j'(h+1)$	$(v_j, h+1)$	$-$	1
r_n	$v_n' n$	$-$	$-$	$+$	1

- La membrana de salida es $i_0 = 0$ (es decir, la piel del P sistema).

Antes de establecer la verificación formal del P sistema Π_G, vamos a dar una idea informal de cómo funciona Π_G.

- El proceso general consiste en ir *construyendo* de forma no determinista todos los posibles caminos simples que parten del nodo v_0.
- Cada computación está asociada a uno de los caminos simples que se pueden construir.
- En cada paso de la computación hay un único elemento de V', que indicará el último vértice añadido al camino que está siendo construido.
- Las reglas que se pueden aplicar a una configuración determinada son las asociadas a las aristas que enlazan el vértice v', presente en la membrana, con los demás vértices presentes y por los que el camino en construcción todavía no ha pasado. Por tanto, la aplicación de cualquiera de las reglas proporciona una posible extensión del camino simple.
- Si se escoge una arista $v_i v_j$, entonces el vértice v_j pasa a la membrana exterior y el vértice v_j' pasa a ser el nodo a considerar.
- Cuando un vértice ha pasado a la membrana exterior, entonces desaparece de la interior. Por tanto, las siguientes elecciones no proporcionarán caminos que pasen por ese vértice.
- Si la elección de los vértices permite crear un camino hamiltoniano que va desde v_0 hasta v_n, entonces debe ocurrir que al final se ejecute la regla $v_n' n \rightarrow \delta$. Por tanto, la membrana 1 se disolverá. Entonces la piel pasa a ser una membrana

elemental y la computación es exitosa. En tal situación, los elementos de la piel serán de la forma (v, i) que representa el vértice y la posición, respectivamente, que ocupa en el camino codificado por el contenido de la piel en la configuración final.

8.3.2. Verificación formal del P sistema diseñado

A continuación se procede a establecer la verificación formal del P sistema de transición no determinista, Π_G, respecto a la existencia de un camino hamiltoniano en G que va desde v_0 hasta v_n.

En primer lugar, veamos que todas las computaciones de Π_G son de parada.

Lema 8.18 *Sea $\mathcal{C}$ una computación del P sistema Π_G. Para cada $k < |\mathcal{C}|$ tal que $\mathcal{C}_{k+1}(1)\downarrow$, se tiene que $|\mathcal{C}_{k+1}(1) \cap V| < |\mathcal{C}_k(1) \cap V|$.*

Demostración:
Basta tener presente que $\forall r \in R_1$ $(|d_r \cap V| < |v_r(here) \cap V|)$, y que si $m \in \mathbf{M_{Ap}}(\mathcal{C}_k)$ verifica que $\mathcal{C}_{k+1} = m(\mathcal{C}_k)$, entonces $supp(m) \subseteq R_1$.

□

Corolario 8.19 *Si $\mathcal{C}$ es una computación de Π_G, entonces $|\mathcal{C}| \leq n + 1$.*

Demostración:
Supongamos lo contrario; es decir, que $|\mathcal{C}| > n + 1$.

Entonces $\forall k$ $(k \leq n + 1 \rightarrow \mathbf{M_{Ap}}(\mathcal{C}_k) \neq \emptyset)$ y, por tanto, $\forall k \leq n + 1$ $(\mathcal{C}_{k+1}(1)\downarrow)$. Del lema 8.18 se deduce que $\{|\mathcal{C}_k(1) \cap V|\}_{0 \leq k \leq n+1}$ es una sucesión estrictamente decreciente de números naturales verificando que $|\mathcal{C}_0(1) \cap V| = n$. Lo que es una contradicción.

□

Definición 8.20 *Para cada subconjunto $A = \{(v_{i_0}, 0), \ldots, (v_{i_k}, k)\} \subseteq V \times \{0, \ldots, n\}$, se define el camino, $\Gamma(A)$, asociado a A como sigue: $\Gamma(A) = v_{i_0} \ldots v_{i_k}$.*

Proposición 8.21 *Sea $\mathcal{C}$ una computación de Π_G. Para cada k tal que $\mathcal{C}_k(1)\downarrow$, se tiene que:*

1. *$\Gamma_k = \Gamma(\mathcal{C}_k(0)) = v_{j_0} \ldots v_{j_k}$ es un camino simple de G de longitud k (donde convendremos que $j_0 = 0$).*

2. $\mathcal{C}_k(1) = V - \Gamma_k \cup \{v'_{j_k}, k\}$.

Demostración:
Probaremos los dos resultados simultáneamente, por inducción débil sobre k.

El caso base, $k = 0$, se obtiene trivialmente de la definición del P sistema.

Sea $k \in \mathbf{N}$ y supongamos cierto el resultado para k. Si $\mathcal{C}_{k+1}(1) \downarrow$, entonces $\mathcal{C}_k(1) \downarrow$ y, por tanto, de la hipótesis de inducción resulta que $\Gamma_k = v_{j_0} \dots v_{j_k}$ es un camino simple de G de longitud k y, además, $\mathcal{C}_k(1) = V - \Gamma_k \cup \{v'_{j_k}, k\}$. Sea $m \in \mathbf{M_{Ap}}(\Pi_G)$ tal que $m(\mathcal{C}_k) = \mathcal{C}_{k+1}$. Como $\mathcal{C}_{k+1}(1) \downarrow$, se tiene que $r_n \notin m$. En consecuencia existe j_{k+1} tal que $v_{j_{k+1}} \in \mathcal{C}_k(1) \wedge v_{j_k} v_{j_{k+1}} \in E \wedge m = r_{j_k j_{k+1} k}$. De donde resulta que $\mathcal{C}_{k+1}(0) = \mathcal{C}_k(0) \cup \{(v_{j_{k+1}}, k+1)\}$ y $\Gamma_{k+1} = v_{j_0} v_{j_1} \dots v_{j_k} v_{j_{k+1}}$ es un camino simple de G de longitud $k+1$. Además, $\mathcal{C}_{k+1}(1) = \mathcal{C}_k(1) \cup \{v'_{j_{k+1}}, k+1\} - \{v'_{j_k}, k\} = V - \Gamma_{k+1} \cup \{v'_{j_{k+1}}, k+1\}$. Lo que establece la validez del resultado para $k+1$.

□

Teorema 8.22 *Sea $\mathcal{C}$ una computación de Π_G. Entonces $\mathcal{C}$ es exitosa si y sólo si $|\mathcal{C}| = n + 1$.*

Demostración:
Supongamos que $\mathcal{C}$ es una computación de Π_G tal que $|\mathcal{C}| = n + 1$. Como $\mathcal{C}_n(1) \downarrow$, aplicando la proposición 8.21 para $k = n$ resulta que $\Gamma_n = \Gamma(\mathcal{C}_n(0)) = v_0 v_{j_1} \dots v_{j_n}$ es un camino simple de G de longitud n y $\mathcal{C}_n(1) = \{v'_{j_n}, n\}$. Por tanto, como $\mathcal{C}_n$ no es una configuración de parada se tiene que $\mathbf{M_{Ap}}(\mathcal{C}_n) = \{r_n\}$ y $j_n = n$. Luego, necesariamente $\mathcal{C}_{n+1} = r_n(\mathcal{C}_n)$. Ahora bien, $\mathcal{C}_{n+1}(1) \uparrow$ y $\mathcal{C}_{n+1}(0) = \mathcal{C}_n(0)$. En consecuencia, $\mathcal{C}$ es una computación exitosa de Π_G.

Supongamos ahora que $\mathcal{C}$ es una computación exitosa de Π_G. Sea $k = |\mathcal{C}|$. Como $\mathcal{C}$ es exitosa, se tiene que verificar que $\mathcal{C}_k(1) \uparrow$. Luego en la configuración $\mathcal{C}_{k-1}$ se ha tenido que aplicar la regla r_n. Ahora bien, de la proposición 8.21 se tiene que $\mathcal{C}_{k-1}(1) = V - \Gamma_{k-1} \cup \{v'_{j_{k-1}}, k-1\}$. Por tanto, para que se aplique la regla r_n, es necesario que $k - 1 = n$. En consecuencia, $|\mathcal{C}| = n + 1$.

□

Corolario 8.23 *Si $\mathcal{C}$ es una computación exitosa de Π_G, entonces $\Gamma(\mathcal{O}(\mathcal{C}))$ es un camino hamiltoniano de G de longitud n que va desde v_0 hasta v_n.*

Teorema 8.24 *Sea γ un camino hamiltoniano de G que va desde v_0 hasta v_n. Entonces existe una computación exitosa, $\mathcal{C}$, de Π_G tal que $\Gamma(\mathcal{O}(\mathcal{C})) = \gamma$.*

Demostración:
Supongamos que $\gamma \equiv v_{i_0}v_{i_1}\dots v_{i_n}$ es un camino hamiltoniano de G que va desde v_0 hasta v_n. Vamos a construir una computación exitosa, $\mathcal{C}$, de Π_G, de tal manera que $\Gamma(\mathcal{O}(\mathcal{C})) = \gamma$.

Teniendo presente que $v_0 = v_{i_0}$, $v_{i_0}v_{i_1} \in E$ y que $v'_0 0 \subseteq \mathcal{C}_0(1)$ resulta que $m_0 = r_{0i_00} \in \mathbf{M_{Ap}}(\mathcal{C}_0)$. Si $\mathcal{C}_1 = m_0(\mathcal{C}_0)$, entonces se verifica que $\mathcal{C}_1(0) = (v_0, 0)(v_{i_1}, 1)$ y $\Gamma(\mathcal{C}_1(0)) = v_0v_{i_1}$.

Supongamos que tenemos construida la configuración $\mathcal{C}_k$, con $k < n$, verificando que $\Gamma(\mathcal{C}_k(0)) = v_0v_{i_1}\dots v_{i_k}$. Vamos a construir la configuración $\mathcal{C}_{k+1}$ verificando que $\mathcal{C}_k \Rightarrow_{\Pi_G} \mathcal{C}_{k+1}$ y $\Gamma(\mathcal{C}_{k+1}(0)) = v_0v_{i_1}\dots v_{i_k}v_{i_{k+1}}$. Teniendo presente que $v_{i_k}v_{i_{k+1}} \in E$ y $v'_{i_k}k \subseteq \mathcal{C}_k(1)$ resulta que $m_k = r_{i_ki_{k+1}k} \in \mathbf{M_{Ap}}(\mathcal{C}_k)$. Sea $\mathcal{C}_{k+1} = m_k(\mathcal{C}_k)$. Se prueba fácilmente que la configuración $\mathcal{C}_{k+1}$ satisface las condiciones deseadas.

Por tanto $\Gamma(\mathcal{C}_n(0)) = \Gamma$ y $\mathcal{C}_n(1) = v'_n n$. De donde resulta que $\mathbf{M_{Ap}}(\mathcal{C}_n) = \{r_n\}$ y $\mathcal{C}_{n+1} = r_n(\mathcal{C}_n)$ verifica que $\mathcal{C}_{n+1}(1) \uparrow$. Como la membrana 0 no tiene reglas, la configuración $\mathcal{C}_{n+1}$ es de parada. En consecuencia, la computación $\mathcal{C}$ así construida es exitosa y, además, $\Gamma(O(\mathcal{C})) = \gamma$.

□

Nota 8.25 Obsérvese que la familia de P sistemas diseñados para resolver el problema del camino hamiltoniano no sólo resuelve dicho problema de decisión, sino que el lenguaje generado por dicho P sistema codifica todos los posibles caminos que son solución correcta del problema **HPP**. Más aún, de la proposición 8.21 se deduce que incluso las computaciones no exitosas generan caminos simples que parten de v_0, aunque no son caminos hamiltonianos del grafo G.

8.3.3. Análisis de la complejidad del P sistema

En primer lugar, del teorema 8.22 se deduce que toda computación exitosa de Π_G ejecuta exactamente $n+1$ pasos. Del corolario 8.19 se deduce que toda computación de Π_G es de parada y ejecuta, a lo sumo, $n+1$ pasos.

En segundo lugar, analicemos el coste de la construcción del P sistema Π_G a partir del grafo G. Se tiene que:

1. El número de símbolo del alfabeto base es $3|V| + |V|^2$.
2. El número de membranas de Π_G es 2.
3. El número de reglas de evolución es $|V| + |E| \cdot (1 + |V|) \in O(|E| \cdot |V|)$.

4. El número de relaciones de prioridad es 0.

Es decir, hemos presentado una solución no determinista del problema **HPP** a través de P sistemas de transición cooperativos, con disolución y sin prioridad, con coste en tiempo lineal en el tamaño del grafo y coste de construcción lineal en el producto del número de aristas por el número de nodos del grafo.

Capítulo 9

Conclusiones y trabajo futuro

Vamos a finalizar esta memoria presentando algunas conclusiones que se pueden extraer de la misma, así como algunos objetivos y trabajos futuros que representan una continuación de la labor desarrollada.

9.1. Análisis y conclusiones

Al diseñar algoritmos para resolver problemas en modelos no convencionales, suele ser habitual verificar informalmente el procedimiento a través de una simple *comprobación intuitiva* de que el proceso funciona correctamente.

Esto viene a ser claramente insuficiente cuando dichos modelos usan paralelismo masivo, como suele ser el caso de los procedimientos diseñados en el marco de la Computación Natural. Además, las estructuras de datos que se manipulan (tubos de ensayo, multiconjuntos de objetos y de reglas, etc.) no son usuales en el desarrollo de una metodología clásica de verificación. Por todo ello, hemos considerado necesario la elaboración de unos métodos y técnicas que permitan atacar con cierta solvencia los procesos de verificación en el ámbito de los modelos de Computación Natural.

Esta necesidad ha sido, quizás, la principal impulsora para desarrollar el trabajo contenido en esta memoria. Presentamos una primera aproximación al estudio sistemático de la verificación de procedimientos en modelos no convencionales. Concretamente, lo hacemos en dos modelos bien diferenciados: el modelo sticker y el modelo básico de computación celular con membranas.

En el caso del modelo sticker se han descrito previamente como sistemas formales los programas diseñados para resolver problemas de decisión. La metodología desarrollada consta de las siguientes fases:

1. **Re-etiquetado de tubos**: que permite realizar un seguimiento pormenoriza-

do de cada uno de los tubos a lo largo de la ejecución.

2. **Función *STEP* de evolución**: que ayuda a estudiar las transformaciones que sufren las moléculas a lo largo de todo el proceso.

3. **Fórmulas invariantes**: que capturen las propiedades que verifican las moléculas de los tubos *relevantes* en cualquier instante de la ejecución. Estas fórmulas se pueden construir a partir de la semántica del programa, en particular de las propiedades de la función *STEP*.

4. Uso de **técnicas inductivas** para establecer la invariancia de las fórmulas consideradas.

5. Establecimiento de la **adecuación** y **completitud** del programa molecular respecto del problema para el que ha sido diseñado, a partir de la veracidad de las fórmulas invariantes al finalizar la ejecución del programa.

Esta metodología ha sido aplicada a una serie de programas moleculares diseñados para resolver algunos problemas numéricos clásicos **NP**-completos. Para ello, se ha desarrollado previamente una batería de programas moleculares relativos a conjuntos numéricos finitos que han sido usados como subrutinas en la resolución de dichos problemas.

En el caso del modelo celular con membranas, los procesos de verificación suelen ser más complejos debido, principalmente, a que se trata de un modelo de computación básicamente de tipo procedural, no imperativo. En la memoria se presenta una primera aproximación al estudio sistemático de la verificación de P sistemas de transición.

Para ello, en primer lugar, se presenta una formalización de dicho modelo celular básico que, con ligeras modificaciones, se puede adaptar a las distintas variantes que se han desarrollado recientemente. Esta formalización ha sido motivada por dos razones fundamentales:

1. Facilitar el desarrollo de una metodología para la verificación de P sistemas, en la línea descrita para el modelo sticker.

2. Diseñar una batería de procedimientos en lenguajes de programación convencionales que permitan la elaboración de simuladores de ejecución de P sistemas (motivado por el hecho de no existir en la actualidad ninguna implementación práctica del modelo celular).

El primer paso realizado en la dirección de sistematizar la verificación de P sistemas ha sido la demostración de la verificación formal de tres P sistemas de transición, en sus posibles interpretaciones como dispositivos generadores de lenguajes, reconocedores de predicados y como máquinas de cálculo de funciones. Estas verificaciones se han basado en el estudio de una serie de *puntos críticos* que aparecen en la ejecución de las distintas computaciones y que han permitido caracterizar las computaciones exitosas del sistema. En los problemas estudiados, los puntos críticos han sido determinados por la disolución de ciertas membranas relevantes. Desde luego, pensamos que la búsqueda de puntos críticos a partir de la disolución de membranas no debe ser considerada como una técnica de carácter general, si bien puede ser una herramienta útil en P sistemas que usan la disolución como un mecanismo más del proceso evolutivo.

La estructura compleja de las computaciones en modelos celulares hace presagiar que no sea posible desarrollar una metodología para la verificación de sistemas, de carácter general. No obstante, el estudio de puntos críticos en las computaciones (bien por medio de instantes de disolución, por la presencia de determinados objetos en ciertas membranas, etc.) puede aportar información relevante para la obtención de fórmulas invariantes que nos permitan establecer la verificación del sistema.

Finalmente se presentan P sistemas de transición que proporcionan *soluciones no deterministas* del problema **SAT** y del problema **HPP**. En realidad se diseñan P sistemas de transición para cada dato de entrada del P sistema considerado y se genera el conjunto de soluciones válidas asociado a dicho dato. La verificación de estos P sistemas sigue la pauta de la aproximación antes descrita.

Debido al carácter no determinista de los procesos evolutivos que se producen en las células vivas, la posible implementación de los P sistemas en ordenadores convencionales debe pasar, ineludiblemente, por la simulación de computaciones no deterministas a través de las máquinas deterministas.

Una consecuencia importante que se deduce del análisis de los P sistemas diseñados, es la necesidad de definir nuevas medidas de complejidad y de eficiencia en este modelo, que proporcionen un nuevo concepto de clases de complejidad, pues las definiciones clásicas son claramente insuficientes.

9.2. 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 por miembros del grupo de investigación en Computación Natural de la Universidad de Sevilla.

1. Diseño de programas que simulen en ordenadores convencionales, las computaciones que se generan en la ejecución de procedimientos diseñados en modelos de Computación Natural. En este contexto, miembros del grupo de investigación antes citado, están desarrollando varios proyectos:

   - Elaboración de un entorno de ejecución virtual que pueda simular las computaciones de programas moleculares en el modelo sticker.
   - Generación del árbol de computación asociado a un P sistema de transición en SCHEME y en JAVA, así como la simulación de ejecuciones en CLIPS.

2. Desarrollo de prototipos ejecutables de modelos moleculares y de modelos celulares con membranas, en sistemas de razonamiento automático (ACL2 y PVS).

3. Elaboración de una metodología de carácter general para la verificación de P sistemas.

4. Establecimiento de la universalidad de los P sistemas a través de la simulación de máquinas de Turing, conjuntos diofánticos, funciones recursivas u otros modelos de computación convencionales.

5. Estudio de la decidibilidad de cuestiones relativas a P sistemas usando las simulaciones obtenidas en el punto anterior.

6. Desarrollo de un Teoría de la Complejidad en modelos celulares con membranas, en la línea iniciada recientemente por Gh. Păun [59].

7. Adaptación de la metodología desarrollada al estudio de la verificación en otros modelos de computación no convencionales (sistemas evolutivos [19], redes de procesadores evolutivos [15], sistemas de prohibidores-obligadores [23], etc.).

Bibliografía

[1] ADLEMAN, L. Molecular Computation of Solutions to Combinatorial Problems. *Science*, 268 (November 1994), 1021–1024.

[2] ADLEMAN, L. On constructing a molecular computer. En *DNA based computers* (R.J. Lipton and E.B. Baum, eds.), American Mathematical Society, 1996, 1–22.

[3] AMOS, M. *DNA computing.* Ph. D. Thesis, University of Warwick, 1997.

[4] AMOS, M.; WILSON, S.; HODGSON, D.A.; OWENSON, G.; GIBBONS, A. Practical implementation of DNA computations. En *Unconventional Models of Computation* (C.S. Calude, J. Casti y M.J. Dinnen, eds.), Springer, 1998, 1–18.

[5] BALCÁZAR, J.L. *Programación Metódica.* Mc Graw Hill, 1993.

[6] BARANDA, A.V.;CASTELLANOS, J.;ARROYO, F.;GONZALO, R. Towards an electronic implementation of membrane computing: A formal description of nondeterministic evolution in transition P systems. *Proceedings of the 7th International Meeting on DNA Based Computers* (N. Jonoska, N.C. Seeman, eds.), Tampa, Florida, USA, 2001, 273–282.

[7] BEAVER, D. A universal molecular computer. En *DNA based computers* (R.J. Lipton and E.B. Baum, eds.), American Mathematical Society, 1996, 29–36.

[8] BONEH, D.; DUNWORTH, C.; LIPTON, R.J. Breaking DES using a molecular computing. En *DNA based computers* (R.J. Lipton and E.B. Baum, eds.), American Mathematical Society, 1996, 37–66.

[9] BONEH, D.; DUNWORTH, C.; LIPTON, R.J.; SGALL, J. On the computational power of DNA. *Discrete Applied Mathematics*, 71 (1996), 79–94.

[10] BERRY, G.; BOUDOL, G. The chemical asbtract machine. *Theoretical Computer Science*, 96 (1992), 217–248.

[11] BLUM, M. A machine-independent theory of the complexity of recursive functions. *Journal ACM*, vol. 14, núm. 2 (1967), 322-336.

[12] BOREK, E. *The Code of Life.* Columbia University Press, 1965.

[13] CALUDE, C.S.; CASTI, J.; DINNEEN, eds. *Unconventional Models of Computation*, Springer, 1998.

[14] CASTELLANOS, J.; PĂUN, GH.; RODRÍGUEZ-PATÓN, A. Computing with membranes: P systems with worm-objects. CDMTCS, Technical Report N° 123, 2000.

[15] CASTELLANOS, J.; MARTÍN-VIDE, C.; MITRANA, V.; SEMPERÉ, J.M. Networks of Evolutionary Processors. Submitted, 2002.

[16] CHURCH, A. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1 (1936), 40–41.

[17] CHURCH, A. An unsolvable problem of elementary number systems. *American Journal of Mathematics*, 58 (1936), 345–363.

[18] COOK, S.A. The complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium in Theory of Computing*, (1971), 151–158.

[19] CSUHAJ-VARJÚ, E.; MITRANA, V. Evolutionary Systems: A Language Generating Device Inspired by Evolving Communities of Cells. *Acta Informatica*, vol. 36, núm. 11 (2000), 913-926.

[20] DIJKSTRA, E.W. *A discipline of programming.* Prentice Hall, 1976.

[21] DASSOW, J.; PĂUN, G. *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[22] EGHOLM, M.; CHRISTENSEN, L.; DUEHOLM, K.L.; BUCHARDT, O.; COULL, J.; NIELSEN, P.E. Efficient pH-independent sequence-specific DNA binding by pseudoisocytosine-containing bis-PNA. *Nucleic Acids Research*, vol. 23, num. 2 (1995), 217–222.

[23] EHRENFEUCHT, A; HOOGEBOOM, H.J.; ROZENBERG, G.; VAN VUGT, N. Forbidding and Enforcing. En *DNA Based Computers V* (E. Winfree, D. Gifford, eds.). DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 54 (2000), 195–206.

[24] FEYNMAN R.P. There's plenty of room at the bottom. En *Miniaturization* (D.H. Hilbert, ed.), Reinhold, 1961, 282–296.

[25] FREUND, R.; MARTÍN-VIDE, C; PĂUN, G. From Regulated Rewriting to Computing with membranes: Collapsing Hierarchies. Submitted, 2001.

[26] GAREY M.R.; JOHNSON D.S. *Computers and intractability*, W.H. Freeman and Company, New York, 1979.

[27] GÖDEL K. Uber formal unentscheidbare Sätze Principia Mathematica und verwandter Systeme, I. *Monast. Math. Phys.*, 38 (1931), 173–198.

[28] GRACIANI, C.; MARTÍN-MATEOS, F.J.; PÉREZ-JIMÉNEZ, M.J.; Specification of Adleman's Restricted Model Using An Automated Reasoning System : Verification of Lipton's Experiment. En fase de aceptación en el *8th International Meeting on DNA Based Computers*, Sapporo, Japón, 2002.

[29] HARTMANIS, J.; LEWIS, P.M.; STEARN, R.E. Hierarchies of memory limited computations. *Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logical Design*, 1965, 179–190.

[30] HEAD, T., Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49 (1987), 737–759.

[31] HEAD, T. Aqueous Simulations of Membrane Computations. *Romanian Journal of Information Science and Technology*, vol. 4, núm. 1-2 (2001).

[32] HOLLAND J.H. *Adaptation in natural and artificial systems.* MIT Press, 1975.

[33] ITO, M.; MARTÍN-VIDE, C.; PĂUN, G. A characterization of Parikh sets of ETOL languages in terms of P systems. Submitted, 2001.

[34] KARI, L. DNA Computing: Arrival of Biological Mathematics. *The Mathematical Intelligencer*, Springer–Verlag, New York, vol. 19, num. 2 (1997), 9–22.

[35] KARP. R.M. Reducibility among combinatorial problems. *Complexity of Computer Computations*, Plenum Press, New York, 1972, 85–104.

[36] KRISHNA, S. N. *Languages of P Systems: Computability and Complexity*, Ph.D. Thesis, Indian Institute of Technology, Madras, 2001.

[37] KRISHNA, S. N.; RAMA, R. A variant of P systems with active membranes: Solving NP-complete problems. *Romanian Journal of Information Science and Technology*, vol. 2, núm. 4 (1999), 357–368.

[38] KRISHNA, S. N.; RAMA, R. On the power of P systems with sequential and parallel rewriting. *International Journal of Computer Mathematics*, vol. 77, núm. 1-2 (2000), 1–14.

[39] KRISHNA, S. N.; RAMA, R. P systems with replicated rewriting, *Journal of Automata, Languages and Combinatorics*, vol. 6, núm. 3 (2001), 345–350.

[40] LANDWEBER, L.F.; BAUM, E.B., eds. *DNA Based Computers II.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, vol. 44, 1998.

[41] LEWIS, P.M.; STEARN, R.E.; HARTMANIS, J. Memory bounds for recognition of context-free and context-sensitive languages. *Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logical Design*, 1965, 191–202.

[42] LIPTON R.J. DNA Solution of Hard Computational Problems. *Science*, 268 (April 1995), 542–545.

[43] LIPTON, R.J.; BAUM E.B., eds. *DNA Based Computers.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, vol 27, 1996.

[44] LUNDH, D.; OLSSON, B.; NARAYANAN, A., eds. *Biocomputing and emergent computation*, World Scientific, 1997.

[45] MANCA, V. String rewriting and metabolism: A logical perspective. En *Computing with Bio-Molecules. Theory and Experiments*, Springer-Verlag, Singapore, 1998, 36–60.

[46] MARTÍN-MATEOS, F.J.; ALONSO, J.A.; PÉREZ-JIMÉNEZ, M.J.; SANCHO, F. Molecular computation models in ACL2: a simulation of Lipton's experiment in the Adleman's restricted model. *Third International Workshop on the ACL2 Theorem Prover and its Applications.* Grenoble, 2002, 175–187.

[47] MARTÍN-VIDE, C.; MITRANA, V.; PĂUN, G. On the power of P systems with valuations. *Computación y sistemas*, to appear.

[48] MARTÍN-VIDE, C.; PĂUN, G.; ROZENBERG, G. Membrane systems with carriers. *Theoretical Computer Science*, 270 (2002), 779–796.

[49] MCCULLOCH, W.S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, vol. 5 (1943), 115–133.

[50] MEADE. *Scientific American*, Mayo 1995, 33–34.

[51] MINSKY, M.; PAPERT, S. *Perceptions*, Cambridge, M.A. MIT Press.

[52] NIELSEN, P.E.; EGHOLM, M.; BERG, R.H.; BUCHARDT, O. Sequence-Selective Recognition of DNA by Strand Displacement with a Thymine-Substituted Polyamide. *Science*, 254 (1991), 1497–1500.

[53] OBTULOWICZ, A. Membrane computing and one-way functions. *International Journal of Foundations of Computer Science*, 12, 4 (2001), 551–558.

[54] PĂUN, A. On P systems with membrane division. En *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinnen, eds.), Springer-Verlag, London, 2000, 187–201.

[55] PĂUN, A.; PĂUN, G. The power of communication: P systems with Symporters-Antiporters. *New Generation Computers.*

[56] PAŬN, GH. Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, y *Turku Center for Computer Science – TUCS Report Nº 208*, 1998 (www.tucs.fi)

[57] PĂUN, GH. P systems with active membranes: attacking NP complete problems. *CDMTCS Research report*, Nº 102, 1999, Auckland Univ., New Zeland, `www.cs.auckland.ac.nz/CDMTCS`.

[58] PĂUN, GH. Further research topics about P systems. *Pre-Proceedings of Workshop on Membrane Computing*, Curtea de Arges, Rumania, Agosto 2001, Technical Report 17/01 of Research Group on Mathematical Mathematical Linguistics, Rovira i Virgili University, Tarragona, España, 2001, 243–250.

[59] PĂUN, GH. *Membrane Computing. An introduction.* Springer-Verlag, to appear.

[60] PĂUN, GH.; PÉREZ-JIMÉNEZ, M.J.; SANCHO, F. On the Reachability Problem for P systems with Porters. Submitted, 2002.

[61] PĂUN GH., ROZENBERG, G. A guide to membrane computing. *Theoretical Computer Science*, to appear.

[62] PĂUN, GH.; ROZENBERG, GR.; SALOMAA, A. *DNA Computing. New Computing Paradigms*, Springer, 1998.

[63] PÉREZ-JIMÉNEZ, M.J. Computación molecular sin memoria basada en ADN. En *Información: Tratamiento y Representación.* (A. Nepomuceno, J.F. Quesada y F. Salguero, eds.), Servicio de Publicaciones, Universidad de Sevilla, 2001, capítulo 15, 271-313.

[64] PÉREZ-JIMÉNEZ, M.J.; SANCHO, F. Solving Knapsack Problems in a Sticker Based Model, *Proceeedings of the Seventh International Meeting on DNA Based Computers* (N. Jonoska and N.C. Sedman, eds.), Tampa, Florida, USA, 2001, 94-104. *Lecture Notes in Computer Science*, 2340, Springer, to appear.

[65] PÉREZ-JIMÉNEZ, M.J.; SANCHO, F. Minimal Set Cover Problem: On a DNA Solution of Selection Stage. *Pre-proceedings of Workshop on Membrane Computing*, Curtea de Arges, Romania, August 20-25, 2001, (C. Martín-Vide, Gh. Paun, eds.) Report 17/01 of the Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 251-258.

[66] PÉREZ-JIMÉNEZ, M.J.; SANCHO, F. A formalization of transition P systems. *Fundamenta Informaticae*, vol. 49 (2002), 261–272.

[67] PÉREZ-JIMÉNEZ, M.J.; SANCHO, F. Verifying a P system generating squares. *Romanian Journal of Information Science and Technology*, vol. 5, núm. 2-3 (2002), 181–191.

[68] PÉREZ-JIMÉNEZ, M.J.; SANCHO, F. Verification of non Deterministic Transition P Systems Solving SAT Problem. En fase de aceptación en el *8th International Meeting on DNA Based Computers*, Sapporo, Japón, 2002.

[69] PÉREZ-JIMÉNEZ, M.J.; SANCHO, F.; GRACIANI, M.C.; ROMERO, A. NP completitud y Computación ADN. *Actas del Encuentro de Matemáticos Andaluces*, vol. 2 (2000), 539–548.

[70] PÉREZ-JIMÉNEZ, M.J.; SANCHO, F.; GRACIANI, M.C.; ROMERO, A. Soluciones moleculares del problema SAT de la Lógica Proposicional. En *Lógica, Lenguaje e Información, JOLL'2000* (A. Nepomuceno y otros, eds.), Ed. Kronos, 2000, 243–252.

[71] PÉREZ-JIMÉNEZ, M.J.; GRACIANI, M.C.; ROMERO, A.; SANCHO, F. Formalización computacional del experimento de Lipton sobre el problema SAT. *Actas del Primer Congreso Español de Algoritmos Evolutivos y Bio-inspirados.* ISBN 84-607-3913-9 (2002), 326-332.

[72] PEVZNER, P.A. *Computational Molecular Biology. An Algorithmic Approach.* The MIT Press, 2000.

[73] PISANTI, N.*A survey on DNA computing.* Technical report, TR–97–07. Università di Pisa, 1997.

[74] ROMERO-JIMÉNEZ, A.; PÉREZ-JIMÉNEZ, M.J. Simulating Turing Machines by P systems with External Output. *Fundamenta Informaticae*, vol. 49 (2002), 273–287.

[75] ROMERO-JIMÉNEZ, A.; PÉREZ-JIMÉNEZ, M.J. Generation of Diophantine Sets by Computing P Systems with External Output. En fase de aceptación en el *3rd International Conference on Unconventional Models of Computation*, Himeji, Japón, 2002.

[76] ROSENBLATT, F. The perception: a probabilistic model for information storage and organization in the brain. *Psychological Review*, vol. 65 (1959), 368–408.

[77] ROWEIS, S.; WINFREE, E.; BURGOYNE, R.; CHELYAPOV, N.V.; GOODMAN, M.F.; ROTHEMUND, P.W.K.; ADLEMAN, L. A Sticker-Based Model for DNA Computation. *Journal of Computational Biology*, vol. 5, num. 4 (1998), 615–629.

[78] ROZENBERG, G.; SALOMAA, A. *The Mathematical Theory of L systems*, Academic Press, New York, 1980.

[79] RUBIN, H.; WOOD, D.H., eds. *DNA Based Computers III.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, vol. 48, 1999.

[80] SALOMAA, A. *Formal Languages.* Academic Press, New York, 1973.

[81] SALOMAA, A. *Turing, Watson-Crick and Lindenmayer. Aspects of DNA complementarity.* Turku Centre for Computer Science. Technical report, 128, 1997.

[82] SINDEN, R.R. *DNA Structure and Function.* Academic Press, 1994.

[83] TURING A.M., On computable numbers, with an application to the Enstcheidungsproblem. *Proceedings of the London Mathematical Society*, 42 (1936), 230–265; correciones ibid. 43 (1936), 544–546.

[84] TURING A.M., Computability and λ-definability. *Journal of Symbolic Logic*, 2 (1937), 153–163.

[85] ZANDRON, C.; MAURI, G.; FERRETI, C. Solving NP-complete problems using P systems with active membranes. En *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinnen, eds.), Springer-Verlag, London, 2000, 289–301.

[86] P systems web page: `http://dna.bio.disco.unimib.it/psystems`

[87] Página Web del Grupo de Investigación en Computación Natural de la Universidad de Sevilla: `http://www.cs.us.es/gcn`

UNIVERSIDAD DE SEVILLA

Reunido el Tribunal integrado por los abajo firmantes en el día de la fecha, para juzgar la Tesis Doctoral de D. Fernando Sancho Caparrini titulada Verificación de programas en modelos de computación no convencionales.

acordó otorgarle la calificación de Sobresaliente cum laude por unanimidad

Sevilla, 27 de Junio 2002

El Vocal, El Vocal, El Vocal,

José A. Alonso

El Presidente El Secretario, El Doctorando,

Alfonso R. Pérez