Índice General

Prólogo

El desarrollo de la Teoría de la Computación ha estado ligado estrechamente a la búsqueda de modelos que formalicen algunos procesos que se manifiestan en la Naturaleza y que pueden ser considerados como procedimientos *mecánicos* o de *cálculo*.

Un análisis detallado de la evolución en el tiempo de dichos procesos sugieren el uso de dos ingredientes básicos en cualquier modelo que pretenda describir rigurosamente la manera de calcular de la Naturaleza. Por una parte, una *estructura de datos* que proporcione los objetos sobre los que se trabaja; y, por otra, una serie de *operaciones* o transformaciones que actúen sobre dichos objetos. A partir de ahí surgirá de manera natural un *modelo de computación* a través del procesamiento de los datos por medio de las operaciones consideradas.

La *Computación Natural* es una disciplina que trata de modelizar mecanismos computacionales inspirándose en la forma en que la Naturaleza lleva a cabo ciertos procesos que podríamos denominar *evolutivos*. Contiene algunas áreas que se han desarrollado en una doble vertiente: teórica y práctica. Por ejemplo, las *redes neuronales* y los *algoritmos genéticos* están inspirados en la biología, han sido implementados en ordenadores electrónicos y su finalidad ha consistido, básicamente, en obtener nuevos algoritmos que mejoren el rendimiento de los algoritmos clásicos en dichos ordenadores.

La *Computación Molecular* es un área de la Computación Natural cuyo objetivo primordial es el uso de las moléculas orgánicas como soporte básico del dispositivo computacional, de tal manera que permita reemplazar los chips electrónicos de silicio por chips moleculares, a fin de superar las limitaciones de eficiencia inherentes a los ordenadores convencionales, para los que el paralelismo no puede considerarse como una alternativa real debido principalmente a los problemas que se derivan del diseño de arquitecturas con un número elevado

de procesadores.

Desde hace algunas décadas, las moléculas de ADN han sido consideradas como candidatos idóneos para elaborar dispositivos computacionales. El direccionamiento y el principio de complementariedad de Watson-Crick permiten usar dichas moléculas como cadenas de un alfabeto que, además, son capaces de implementar realmente un paralelismo masivo al ser factible procesar de manera simultánea billones de moléculas en un pequeño tubo de ensayo. La *Computación Molecular basada en ADN* nació propiamente en noviembre de 1994 cuando L. Adleman realizó un experimento en el laboratorio que permitió resolver una instancia de un problema computacionalmente intratable, a través de la manipulación de moléculas de ADN.

Generalmente la computación ADN trata de implementar *in vitro* procesos que tienen lugar a nivel genético. Sin embargo, muchos procesos *in vivo* no pueden ser de momento simulados *in vitro*. Por ello, parece interesante usar la propia célula como entorno computacional básico de un nuevo modelo de Computación Natural.

La *Computación Celular con Membranas* nace del supuesto de que los procesos que tienen lugar en el interior de las células vivas pueden ser considerados como procesos de cálculo, como computaciones. Este modelo abre nuevas expectativas a la hora de atacar la *resolubilidad práctica* de problemas **NP**–completos. A diferencia de otros modelos, éste no ha sido aún implementado ni en medios electrónicos ni en medios bioquímicos.

La introducción de un nuevo modelo de computación no convencional obliga a revisar el concepto de *resolubilidad de un problema* en dicho modelo y hace necesario desarrollar una Teoría de la Complejidad que precise las medidas que cuantifiquen la cantidad de recursos utilizados en las computaciones del mismo.

El objetivo de este libro consiste en presentar una primera aproximación a una Teoría de la Complejidad en el modelo de Computación Celular con Membranas. Para ello nos hemos inspirado en unas ideas que Gh. Păun expuso a los autores y que fueron debatidas a principios de este año (véase el capítulo 7 de [39]). Hemos de resaltar que éste es el primer texto que se publica sobre Teoría de la Complejidad en modelos de Computación Celular con Membranas.

El libro está estructurado en capítulos cuyo contenido pasamos a describir brevemente. En el primer capítulo se presenta una sucinta introducción histórica de la Computación Natural y de la Teoría de la Complejidad que, a nuestro juicio, justifican la necesidad de desarrollar una teoría específica de la

Complejidad en P sistemas.

En el segundo capítulo se introducen los P sistemas básicos de transición con símbolos–objetos, a través de una descripción informal de su sintaxis y de su semántica. Además, se estudian las dos variantes de la Computación Celular con Membranas que van a ser utilizadas a lo largo del curso: los *P sistemas de transición con salida externa* y los *P sistemas de decisión*. De ambos modelos se presentan sendas formalizaciones siguiendo la línea general desarrollada en [44].

En la literatura actualmente existente, suele ser usual establecer la universalidad de las diversas variantes del modelo de Computación Celular con Membranas a partir de ciertos resultados de la Teoría de Lenguajes Formales. En [47] se prueba por primera vez la universalidad de los P sistemas de transición con salida externa a través de las máquinas de Turing. El capítulo tercero está dedicado al estudio de la simulación de máquinas de Turing deterministas mediante los P sistemas introducidos en el capítulo anterior. Concretamente se asocia a cada máquina de Turing determinista, TM, un P sistema con salida externa, Π_{TM}, que simula el funcionamiento de la máquina. El P sistema Π_{TM} es descrito de manera exhaustiva y se detalla de manera informal cómo se comporta a fin de simular la máquina TM. Conviene resaltar el hecho de que el P sistema trabaja en dos fases bien diferenciadas: la primera de ellas es no determinista y permite generar, en sus distintas computaciones, todos los posibles datos de entrada de la máquina de Turing; la segunda fase es determinista y simula paso a paso la computación de la máquina con el dato de entrada generado en la fase anterior.

El capítulo cuarto está dedicado a demostrar que el P sistema Π_{TM} diseñado en el capítulo anterior, simula realmente el comportamiento de la máquina de Turing TM. Hay que destacar en primer lugar la complejidad de la prueba de verificación, que es estructurada en cuatro etapas (de generación del dato de entrada, de simulación de un paso de transición, de chequeo del estado final y de producción de la salida). En segundo lugar resaltamos que la prueba de verificación proporciona indirectamente la cantidad de recursos que son necesarios para llevar a cabo la simulación, y que es, en cierto sentido, de tipo polinomial.

Las primeras clases de complejidad en P sistemas se introducen en el capítulo quinto. Se trata de unas clases asociadas a tipos de *P sistemas de aceptación* con o sin entrada. Como caso particular, se destaca la clase de complejidad en Computación Celular que corresponde a la clase **P** de los problemas tratables

en modelos clásicos de computación, y se ilustra con la demostración de que el problema de la satisfactibilidad de la lógica proposicional pertenece a la clase citada.

El último capítulo del libro está dedicado al estudio de la conjetura $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ en el marco de la Computación Celular con Membranas. Es bien conocido un resultado de C. Zandron, C. Ferretti y G. Mauri [54] que proporciona una condición *necesaria* para que se verifique $\mathbf{P} \neq \mathbf{NP}$, a través de la irresolubilidad en tiempo polinomial de problemas **NP**–completos en P sistemas que no admiten división de membranas. En este capítulo se obtiene una *caracterización* de la relación $\mathbf{P} \neq \mathbf{NP}$, en términos de irresolubilidad en tiempo polinomial de problemas **NP**–completos mediante familias de P sistemas de decisión deterministas (introducidos en el capítulo 2).

Desde luego, la caracterización presentada de la relación citada proporciona un atractivo adicional al estudio de la Computación Celular con Membranas al permitir atacar la conjetura $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ en un modelo de computación no convencional. No obstante, no podemos obviar las dificultades que aparecen cuando se trata de establecer la irresolubilidad de un problema en un modelo como el de los P sistemas que es no convencional y, además, no está soportado en un lenguaje de programación.

Confiamos en que este libro pueda servir de ayuda para analizar las dificultades que surgen cuando se trata de formalizar los procesos de verificación y de desarrollar una Teoría de la Complejidad (es decir, de la *computabilidad práctica*) en modelos de computación no convencionales. Estamos convencidos de que el aliciente por superar dichas dificultades estimulará a algunos lectores a adentrarse en el estudio e investigación de la Computación Celular con Membranas que tiene un pasado reciente, un presente prometedor, atractivo y, sobre todo, un futuro fascinante.

M.J. Pérez Jiménez
A. Romero Jiménez
F. Sancho Caparrini
Sevilla, Septiembre de 2002

Capítulo 1

Introducción

Los avances que la Ciencia ha conocido a lo largo de su historia se han producido como consecuencia de una serie de sacudidas y 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 el desarrollo histórico de la *Computación* se pueden observar ciertos puntos de inflexión que han marcado de forma decisiva su evolución, y han propiciado, tanto a nivel teórico como práctico, el desarrollo de nuevas herramientas que han demostrado ser fundamentales. Como factor común a la mayoría de dichos puntos críticos se puede observar el hecho de que su aparición suele estar asociada con la puesta en evidencia de ciertas *limitaciones* inherentes a la potencia de determinados métodos y/o modelos.

El primer hito relevante que se puede 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, obteniéndose como resultado 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.

El estudio de la existencia de problemas abstractos que no puedan ser *efectivamente* resueltos y la necesidad imperiosa de dar respuesta a la tercera cuestión formulada por D. Hilbert, en 1928, relativa a la *decidibilidad* de las Matemáticas, pone de manifiesto la conveniencia de formalizar los conceptos de

algoritmo y de *función computable*. Como resultado de dicho estudio aparecen, en la década de los treinta, los primeros modelos teóricos de computación y se inicia el estudio formal del concepto de *procedimiento efectivo*. El establecimiento 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*) proporcionó una nueva e importante *limitación* al método de formalización del concepto de algoritmo.

Posteriormente, en la década de los cincuenta, surgen los primeros ordenadores electrónicos y, con ellos, lo que se puede denominar informalmente como *modelos de computación práctica*. En éstos, mediante un proceso de traducción a un lenguaje que puede ser interpretado por la máquina, se pueden implementar los algoritmos diseñados, surgiendo una nueva *limitación*, en este caso, de la potencia de los modelos de computación práctica, al establecer la existencia de problemas resolubles algorítmicamente que necesitan una cantidad de *recursos* que excede las posibilidades reales de cualquier ordenador electrónico.

La necesidad de estudiar la mínima cantidad de recursos necesarios para resolver un problema abstracto da origen a la *Teoría de la Complejidad*, que tiene como objetivo fundamental el proporcionar una clasificación de los problemas en función de su resolubilidad en modelos de computación prácticos.

En este marco se plantea la necesidad de mejorar la velocidad de cálculo (parámetro temporal) y la densidad de almacenamiento de información (parámetro espacial) 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. Pero en la década de los ochenta, R. Churchhouse establece los limites físicos en la velocidad de cálculo de un ordenador convencional y, en consecuencia, proporciona una nueva *limitación*, esta vez para los modelos convencionales de computación práctica.

Ante la imposibilidad de resolver dichos problemas por medio de dispositivos clásicos de computación surge la necesidad de buscar modelos de computación alternativos que permitan implementar máquinas que mejoren cuantitativamente el rendimiento de las clásicas. La posibilidad de desarrollar modelos de computación no convencionales abre un nuevo horizonte a la hora de amorti-

guar los efectos propios de las limitaciones inherentes a la computación práctica en modelos clásicos.

1.1 Computación Natural

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 provoque 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 alto coste en una de las medidas sea *asimilado*, en cierto sentido, por el propio modelo en beneficio de una reducción considerable de la otra.

Es por ello que la búsqueda de nuevos modelos alternativos de computación debe ir encaminada a la consecución de una mejora cuantitativa en los resultados que proporciona la *Teoría de la Complejidad* que podríamos denominar clásica.

Esta búsqueda ha dado como resultado la introducción, en los últimos años, de nuevos modelos de computación sustancialmente distintos de los clásicos o convencionales (como pueden ser las 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 clásica, en la búsqueda de nuevos paradigmas que puedan proporcionar una solución *efectiva* a las limitaciones que poseen los modelos convencionales. En la actualidad, 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. Sintetizando, se puede decir que 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 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).

Ahora bien, este enfoque que se produce desde la óptica de la Computación Natural puede tener diversas interpretaciones a la hora de describir los nuevos modelos: ya sea porque se utilice para el diseño de nuevos esquemas algorítmicos usando técnicas inspiradas en la naturaleza, o bien porque 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 [12] 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. De esta manera, los complejos moleculares empiezan a ser considerados como componentes virtuales de un dispositivo de procesamiento de información. En 1987, T. Head [17] propone explícitamente el primer modelo teórico de computación 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 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 déca-

da 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.

A continuación pasamos a estudiar con un poco más de detalle algunas cuestiones relativas a los modelos de computación celular con membranas.

1.2 Computación Celular con Membranas

Como hemos indicado anteriormente, 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 que sea imposible modelizarlos completamente, ya que un modelo de computación que intente simular *literalmente* esos procesos dejaría de ser práctico, salvo desde un punto de vista biológico. Lo que se pretende es 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, 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, al-

gunas veces de forma selectiva e incluso en una sola de las direcciones. Estas ideas fueron consideradas con anterioridad por G. Berry y V. Manca en [5] y [28], respectivamente.

En las células tienen lugar una serie de reacciones químicas que provocan una 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.

El modelo de *Computación Celular con Membranas* fue introducido por Gh. Păun en octubre de 1998 [36] como un modelo de tipo distribuido y paralelo, y está inspirado en el funcionamiento de la célula como organismo vivo capaz de generar y procesar información.

Los ingredientes básicos de un *P sistema* (que es como también se conocen los dispositivos 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 desplazarlos entre 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.

A pesar de que el modelo de computación celular tiene inspiración biológica, debe 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.

Desde la aparición de los primeros P sistemas han sido muchas las variantes introducidas buscando unas veces mayor eficiencia en la solución de problemas complejos, y otras una mayor aproximación al modelo biológico real que trata de simular [38]. Entre las diversas variantes que se consideran destacan:

- Los P sistemas que hacen 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.

- Los P sistemas que admiten 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 inalterados tras la ejecución de la misma).

- Los 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 otros modelos de Computación Natural, 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 basada en ADN) o bien a través de una adaptación en ordenadores electrónicos (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 [18], y a una simple simulación de computaciones y/o ejecuciones de los P sistemas a través de lenguajes de programación convencionales ([2], [3], [4]).

Hay que resaltar 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).

1.3 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 [16][27]), mediante el análisis 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 la resolubilidad mecánica de 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 de dichas teorías

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 genérico 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 asintótico 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 toda una clase de algoritmos (en este caso, a la clase de todos los algoritmos que resuelven dicho problema).

Como es fácilmente imaginable, la tarea de calcular un *algoritmo óptimo* que resuelva un problema suele ser ardua, 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* [6]), 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 sencillo 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 adicional en tiempo 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 $\mathbf{P}\stackrel{?}{=}\mathbf{NP}$. 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 [10] 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 [22] 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 [14] de M.R. Garey y D.S. Johnson constituye un catálogo exhaustivo de problemas **NP**-completos.

En relación a los modelos de Computación Natural, y en particular a los modelos de Computación Celular con Membranas, surge de manera natural una cuestión de vital importancia para la Teoría de la Complejidad: ¿pueden estos modelos no convencionales aportar alguna luz acerca de la conjetura anteriormente expuesta?

Uno de los objetivos fundamentales de este libro consiste en demostrar que los modelos de Computación Celular con Membranas proporcionan una nueva herramienta de carácter teórico que permite abordar dicha conjetura, abriendo un nuevo frente que puede proporcionar una respuesta (positiva o negativa) a la misma. Para ello, se prueba que la respuesta a dicha conjetura se puede obtener a partir de la irresolubilidad de algún problema **NP**–completo en dicho modelo no convencional.

Capítulo 2

Variantes de P sistemas

En este capítulo presentamos una descripción informal de los P sistemas básicos de transición (sección 1), así como todos los preliminares (sección 2) que son necesarios para una formalización de las dos variantes del modelo básico que se van a utilizar a lo largo del texto: la primera de ellas no tiene membrana de entrada y es un sistema generador de lenguajes (sección 3); y la segunda variante sí tiene membrana de entrada y es un sistema de decisión (sección 4). Debemos destacar que un rasgo común a ambas es que recogen la salida en el entorno externo de la estructura de membranas, y no en una membrana en particular.

2.1 P sistemas de transición: descripción informal

En líneas generales, los modelos de Computación Celular con Membranas están basados 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) y, además, desaparecen del sistema las reglas de evolución asociadas a la membrana que se disuelve.

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 han 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 recordar que el paralelismo presente en los P sistemas se produce en 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.

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 $\{,\}$ (llaves *simples*), cuando hablamos de multiconjuntos usaremos como símbolos auxiliares $\{\{,\}\}$ (llaves *dobles*).

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).

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 quiere 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*.

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 2.1).

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.

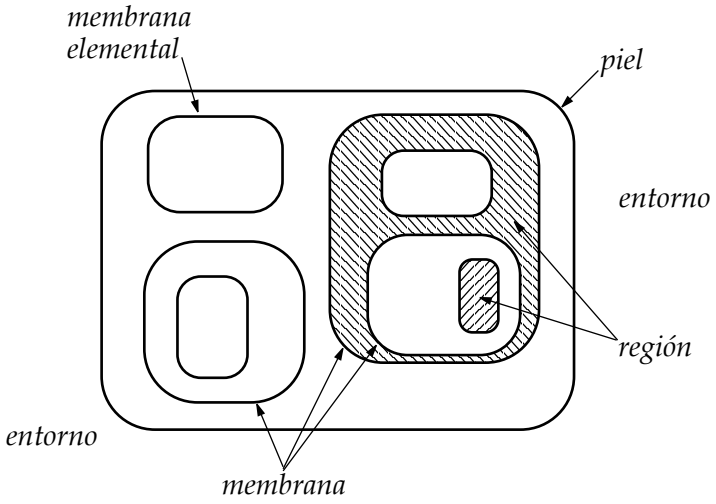

Figura 2.1: Representación gráfica de una estructura de membranas.

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*.

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*.

Evolución de los P sistemas de transición Hasta ahora se han introducido los elementos necesarios para poder definir sintácticamente 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 $u \to v$), 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 $u \to v$ 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 $u \to v$. 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 de mayor a menor prioridad). Todos los objetos de u son *agotados* en el uso de la regla $u \to v$; es decir, el multiconjunto definido por u es *eliminado* del multiconjunto m'_{i_t}.
- El resultado de aplicar la regla $u \to v$ viene determinado por v:
  - Si un objeto aparece en v en un par de la forma $(a, here)$, entonces los objetos de a se añadirán al contenido de la 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 los objetos de a *saldrán* de la membrana i_t, pasando a formar parte de la membrana inmediatamente superior: si los elementos de a estuvieran en la membrana más exterior (es decir, en la piel), entonces simplemente abandonan la célula y pasan al entorno.

- Si un objeto aparece en v en un par de la forma (a, in_q), entonces los objetos de a serán añadidos 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; es decir, no será aplicable una regla del tipo $u \to v\delta$ que aparezca en la piel.

Todas las operaciones antes descritas se realizan en paralelo, para todas las posibles reglas $u \to v$ 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.

Nosotros consideraremos la versión fuerte en los P sistemas de transición.

Una *computación*, $\mathcal{C}$, de un P sistema de transición Π, es una sucesión (finita o infinita) de configuraciones $\mathcal{C}^0 \Rightarrow_\Pi \mathcal{C}^1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi \mathcal{C}^r$, con $r \geq 0$, de tal

manera que $\mathcal{C}^0$ es la configuración inicial de Π y para cada $i \geq 0$ se verifica que $\mathcal{C}^{i+1}$ se obtiene de $\mathcal{C}^i$ por un paso de Π. Diremos que una computación, $\mathcal{C}^0 \Rightarrow_\Pi \mathcal{C}^1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi \mathcal{C}^r$ es de *parada* si $r \in \mathbf{N}$ y en la configuración $\mathcal{C}^r$ (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 $\mathcal{C}^r$. Diremos que una computación, $\mathcal{C}^0 \Rightarrow_\Pi \mathcal{C}^1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi \mathcal{C}^r$ es *exitosa* si es de parada y, además, la membrana i_0 aparece en $\mathcal{C}^r$ como membrana elemental; es decir, i_0 no debe contener otras membranas en su interior, en la configuración $\mathcal{C}^r$. En este caso, a la configuración $\mathcal{C}^r$ se le denomina una *configuración exitosa* de dicha computación.

Variantes de P sistemas Hemos descrito informalmente el modelo básico de P sistemas de transición, que fue el primer modelo de computación celular con membranas desarrollado. Desde que G. Păun lo introdujo en 1998 se han considerado gran cantidad de variantes, que pueden diferir del modelo básico de múltiples maneras.

Una diferencia radical consistiría en usar otros tipos de reglas o estructuras no jerarquizadas de membranas interconectadas entre sí (grafos de membranas).

Otras posibilidades más suaves tienen que ver con la salida, e incluso con la entrada, de los P sistemas: podemos pensar en modelos que tengan una membrana de entrada, en la que se introduzcan objetos antes de iniciar las computaciones.

Respecto a la salida, las diferencias se pueden encontrar bien en dónde se recoge la salida, ya sea en una membrana o en el entorno externo del sistema, bien qué se considera como salida, lo que puede dar lugar a sistemas generadores, reconocedores, de cálculo, de decisión, y cualquier otro que se nos ocurra.

2.2 Preliminares para la formalización

En esta sección vamos a introducir todos los conceptos que son necesarios para formalizar las dos variantes del modelo de Computación Celular con Membranas que van a ser objeto de estudio en este texto: los P *sistemas con salida externa* y los P *sistemas de decisión*. Concretamente, conceptos relativos a multiconjuntos, grafos y árboles. Además, se formalizará en esta sección el concepto de estructura de membranas que es común a las dos variantes.

2.2.1 Multiconjuntos

Definición 2.1 *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 2.2 *Dados $m_1, m_2 \in \mathbf{M}(A)$ y $n \in \mathbf{N}$, 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:* $(m_1 + m_2)(j) = m_1(j) + m_2(j)$, *para cada* $j \in A$.
- *Diferencia:* $(m_1 - m_2)(j) = \max\{m_1(j) - m_2(j), 0\}$, *para cada* $j \in A$.
- *Amplificación:* $(n \cdot m_1)(j) = n \cdot m_1(j)$, *para cada* $j \in A$.

El *tamaño* de un multiconjunto finito, m, que se notará por $|m|$, se define como $|m| = \sum_{a \in A} m(a)$. 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).

2.2.2 Grafos y árboles

Un *grafo* es una construcción matemática 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 $E \subseteq \big\{\{u, v\} : u \in V \wedge v \in V \wedge u \neq v\big\}$ (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*), cuyos elementos se denominan *aristas*.

Sean $G = (V, E)$ 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 no dirigido) o bien $\langle u, v\rangle \in E$ (en el caso de un grafo dirigido). Para englobar ambos casos, notaremos genéricamente $[u, v] \in E$.

Un *camino de longitud* k desde un vértice u de G hasta un vértice v es una sucesión finita $x_0, x_1, \ldots, x_k$ de vértices de G tal que $u = x_0, v = x_k$ y $[x_j, x_{j+1}] \in E$, para todo $j = 0, \ldots, k-1$. Obsérvese que la longitud de un 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 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. Obsérvese que en un grafo no dirigido, un ciclo debe tener al menos longitud 3. Un grafo no dirigido sin ciclos se denomina *acíclico*.

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 se le llama *raíz* del árbol.

Sea T un árbol enraizado de raíz r. Dado un vértice u, cualquier otro vértice, v, 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.

Para cada vértice, u, del árbol que no sea la raíz, notaremos por $ft_T(u)$ (el *padre* de u) el único antecesor propio de u adyacente a él. El padre de la raíz no está definido. Para cualquier vértice, u, del árbol, notaremos por $ch_T(u)$ (los *hijos* de u) el conjunto de descendientes propios de u tales que u es su padre.

Dado un árbol enraizado, T, con raíz r, podemos definir una relación binaria, $E^*(T)$, sobre sus nodos, de forma natural, como sigue: $\langle x, y\rangle \in E^*(T) \Leftrightarrow y \in ch_T(x)$. Esta relación establece, de alguna forma, un direccionamiento sobre las aristas del árbol enraizado. Además, es fácil comprobar que determina unívocamente las aristas de T.

2.2.3 Estructuras de membranas

A continuación formalizamos el concepto de estructura de membranas, que es el marco dentro del cual trabajan los modelos de computación con membranas.

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

El grado *de una estructura de membranas es el número de membranas que contiene.*

Los elementos de una estructura de membranas reciben una etiqueta para individualizarlos. Para facilitar las notaciones y definiciones supondremos por convenio que las membranas de una estructura de grado p están etiquetadas con los números de 1 a p. Además, como estas etiquetas no cambiarán, podemos identificar las membranas con sus etiquetas; esto es, las membranas de una estructura de grado p *serán* los números de 1 a p.

La membrana piel de una estructura de membranas, cuya etiqueta denominaremos genéricamente por $skin$, aísla a ésta de lo que se conoce como entorno externo de la estructura. En las variantes de P sistemas que vamos a considerar es en este entorno donde se recogerá la salida de las computaciones. Es por ello que debemos asociarlo de alguna manera a la estructura de membranas.

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

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

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

Obsérvese que lo único que hacemos es incluir un nuevo nodo que representa al entorno externo, y que por tanto sólo está conectado con la piel, mientras que la estructura de membranas original permanece inalterada.

2.3 P sistemas con salida externa

A continuación presentamos una formalización del modelo básico de computación con membranas que vamos a considerar.

En primer lugar, establecemos qué es una regla de evolución, concepto que sólo depende del alfabeto a partir del cual se construyen los objetos a manejar por el sistema, y del conjunto de membranas con el que trabajaremos.

Definición 2.5 *Sean Γ un alfabeto finito no vacío y μ una estructura de membranas. El conjunto de* reglas de evolución de transición *asociado a Γ y μ, $TER(\Gamma, \mu)$, es el conjunto de todas las posibles tuplas (u, v, δ), donde*

- *u es un multiconjunto sobre Γ.*
- *v es una aplicación de dominio $V(\mu) \cup \{here, out\}$, cuyo rango está contenido en $\mathbf{M}(\Gamma)$.*
- *$\delta \in \{0, 1\}$.*

En segundo lugar, describimos la sintaxis de los P sistemas de transición con salida externa.

Definición 2.6 *Un* P sistema de transición con salida externa *de grado p es una tupla*

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

donde

- *Γ es un alfabeto finito no vacío cuyos elementos se llaman* objetos.
- *μ_{Π} es una estructura de membranas de grado p.*
- *$\mathcal{M}_i$ es un multiconjunto sobre Γ asociado a la membrana i, para cada $i = 1, \ldots, p$.*
- *R_i es un subconjunto finito de $TER(\Gamma, \mu_{\Pi})$ asociado a la membrana i y ρ_i es un orden parcial estricto sobre R_i, para cada $i = 1, \ldots, p$.*

A continuación, detallamos la semántica de este modelo de computación.

Para estudiar cómo evoluciona un P sistema con salida externa debemos conocer en qué estado se encuentra dicho sistema en un momento determinado (es decir, su configuración), para luego analizar las posibles transiciones entre estos estados.

Definición 2.7 *Sea Π un P sistema de transición con salida externa.*

- *Una* configuración *de Π es un par $C = \big(Ext(\mu), M\big)$ tal que:*
  - *$V(\mu) \subseteq V(\mu_{\Pi})$ y la raíz de ambas estructuras de membranas es la misma.*
  - *M es una aplicación de $V(Ext(\mu))$ en $\mathbf{M}(\Gamma)$. Para cada nodo $nd \in V(Ext(\mu))$ notaremos $M_{nd} = M(nd)$.*
- *La* configuración inicial *de Π es el par $C_0 = \big(Ext(\mu), M\big)$ donde:*
  - *$\mu = \mu_{\Pi}$.*
  - *$M_i = \mathcal{M}_i$ para cada $i = 1, \ldots, p$.*
  - *$M_{env} = \emptyset$.*

  En lo que sigue, fijaremos

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

un P sistema de transición con salida externa y $C = \big(Ext(\mu), M\big)$ una configuración de Π.

Definición 2.8 *Diremos que la regla $r = (u_r, v_r, \delta_r) \in TER(\Gamma, \mu_{\Pi}$ es* aplicable a C en la membrana i, *y lo notaremos $r \in Ap(C, i)$, si se cumple lo siguiente:*

- *La membrana existe en la configuración: $i \in V(\mu)$.*
- *La regla está asociada a la membrana: $r \in R_i$.*
- *La regla no intenta disolver el nodo raíz: $i = skin \rightarrow \delta_r = 0$.*
- *La membrana tiene los objetos necesarios para aplicar la regla: $u_r \leq M_i$.*
- *Las membranas a las que la regla intenta enviar objetos existen en la configuración y son inmediatamente interiores a la membrana i: $\forall j \in V(\mu_{\Pi})\big(v_r(j) \neq \emptyset \rightarrow j \in V(\mu) \wedge j \in ch_{\mu}(i)\big)$.*
- *No hay otras reglas aplicables a C en la membrana i con mayor prioridad: $\neg\exists r'\big(\rho_i(r', r) \wedge r' \in Ap(C, i)\big)$.*

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 reglas aplicable a una configuración, que va a representar una colección de reglas de evolución que se puede seleccionar para realizar un paso de transición en el P sistema. A fin de que éste sea correcto, es necesario que las reglas contenidas en el multiconjunto, así como su multiplicidad, cumplan ciertos requisitos.

Definición 2.9 *Diremos que* $amr \in \mathbf{M}(TER(\Gamma, \mu_\Pi) \times V(\mu_\Pi))$ *es un* multiconjunto de reglas aplicable a C, *y lo notamos* $amr \in \mathbf{M_{Ap}}(C)$, *si verifica:*

- *Contiene al menos una regla:* $amr \neq \emptyset$.
- *Las reglas contenidas en el multiconjunto son aplicables a* C *en la membrana asociada:*

$$\forall \langle r, i \rangle \in amr \big(amr(r,i) \neq 0 \rightarrow r \in Ap(C,i) \big)$$

- *Todas las reglas se pueden aplicar a la vez:*

$$\forall i \in V(\mu) \big(\sum_{r \in R_i} amr(r,i) \cdot u_r \leq M_i \big)$$

- *Las reglas se aplican de forma maximal:*

$$\neg \exists amr' \big(amr < amr' \wedge amr' \in \mathbf{M_{Ap}}(C) \big)$$

La acción de un multiconjunto de reglas aplicable sobre una configuración da lugar a dos tipos de procesos claramente diferenciados: por una parte, los objetos contenidos en las membranas evolucionan y son desplazados entre ellas de acuerdo con las reglas del multiconjunto; por otra parte, la ejecución de reglas que supongan la disolución de membranas implica un cambio en la estructura de membranas dentro de la cual estamos trabajando.

A continuación mostramos cómo caracterizar, de una manera formal, la nueva estructura de membranas obtenida por este segundo proceso.

Primero caracterizamos las membranas que deben disolverse por la acción de un multiconjunto de reglas aplicable.

Definición 2.10 *Sea $amr \in \mathbf{M_{Ap}}(C)$ un multiconjunto de reglas aplicable a C. Se define el conjunto:*

$$\Delta(amr, C) = \{i \in V(\mu) : \exists r \in TER(\Gamma, \mu_{\text{п}})\big(amr(r, i) > 0 \wedge \delta_r = 1\big)\}$$

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, i, que permanezca en la próxima configuración, cuál es el conjunto de membranas que se disuelven añadiendo su contenido a i (tales membranas se llamarán *donantes* de i).

Definición 2.11 *Sea $amr \in \mathbf{M_{Ap}}(C)$ un multiconjunto de reglas aplicable a C. Para cada nodo $i \in V(\mu)$, se define el conjunto de* donantes *de i para C por la aplicación de amr, como sigue:*

$$Don(i, amr, C) = \begin{cases} \emptyset, & \text{si } i \in \Delta(amr, C) \\ \{j \in V(\mu) : j \in \Delta(amr, C) \wedge & \\ \quad \wedge\ i \rightsquigarrow_\mu j\ \wedge\ \forall k \in V(\mu) & , \quad \text{si } i \notin \Delta(amr, C) \\ \quad \big(i \rightsquigarrow_\mu k \rightsquigarrow_\mu j \rightarrow k \in \Delta(amr, C)\big)\} & \end{cases}$$

Es decir, si un nodo i no se disuelve, entonces el nodo j es donante de i si j se disuelve por la aplicación de amr, así como todo nodo que sea descendiente propio de i y antecesor propio de j.

Finalmente, calculamos la nueva estructura de membranas obtenida.

Definición 2.12 *Sea $amr \in \mathbf{M_{Ap}}(C)$ un multiconjunto de reglas aplicable a C. Entonces la estructura de membranas que resulta de aplicar amr sobre C, que notamos $amr(\mu)$, es el árbol enraizado dado por:*

- $V(amr(\mu)) = V(\mu) \setminus \Delta(amr, C)$.
- $E^*(amr(\mu)) = \big\{\langle i, j\rangle \in V(amr(\mu))^2 : \exists i_0, \ldots, i_n \in V(\mu)\big(i_1, \ldots, i_{n-1} \in \Delta(amr, C) \wedge i_0 = i \wedge i_n = j \wedge \forall l\big(0 \leq l < n \rightarrow i_l = ft_\mu(i_{l+1})\big)\big)\big\}$.
- *La raíz de $amr(\mu)$ coincide con la raíz de μ.*

Es decir, $amr(\mu)$ es el árbol enraizado obtenido de μ eliminando los nodos que se disuelven por la aplicación del multiconjunto amr, y restaurando las conexiones en la forma correcta. Si un nodo j es disuelto, entonces el nodo se

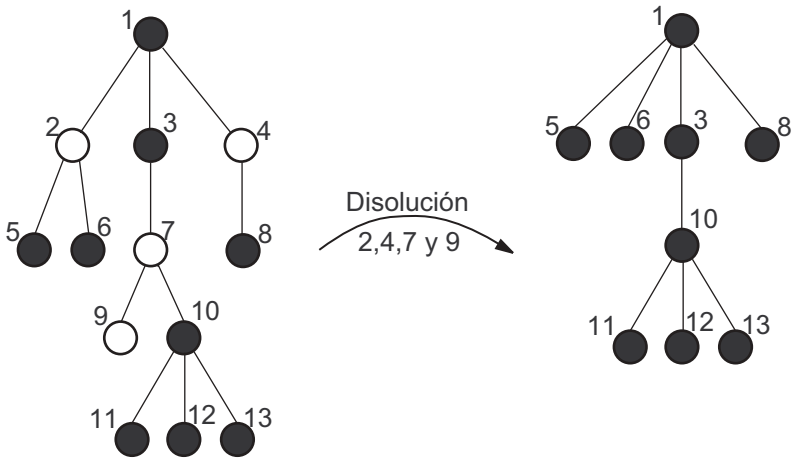

Figura 2.2: Ejemplo de disolución.

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 j no disueltos.

En la figura 2.2 se muestra un ejemplo del árbol de membranas resultante tras haber disuelto un conjunto de nodos (en este caso, $\Delta(amr, C) = \{2, 4, 7, 9\}$).

La evolución y desplazamiento de los objetos por las membranas se calcula en dos pasos: primero se aplican las reglas del multiconjunto de reglas, amr, sin tener en cuenta las disoluciones y, seguidamente, se recolocan los objetos de las membranas disueltas.

Definición 2.13 *La función $amr(M)$ que a cada membrana de $Ext(amr(\mu))$ le asocia los objetos que contiene como resultado de aplicar amr sobre C, se define como sigue:*

- *Dado $i \in V(\mu)$,*

$$M_i' = M_i - \sum_{r \in R_i} amr(r,i) \cdot u_r + \sum_{r \in R_i} amr(r,i) \cdot v_r(here) + \\ + \sum_{r \in R_{ft_\mu(i)}} amr(r, ft_\mu(i)) \cdot v_r(i) + \sum_{j \in ch_\mu(i)} \sum_{r \in R_j} amr(r,j) \cdot v_r(out)$$

- *Dado* $i \in V(amr(\mu))$,

$$(amr(M))_i = M'_i \cup \bigcup_{j \in Don(i,amr,C)} M'_j$$

y

$$M'_{env} = M_{env} + \sum_{r \in R_{skin}} amr(r, skin) \cdot v_r(out)$$

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.

Definición 2.14 *Se dice que una configuración* $C_2 = \big(Ext(\mu_2), M_2\big)$ *de* Π *deriva de una configuración* $C_1 = \big(Ext(\mu_1), M_1\big)$ *por una* transición en un paso *(o tras la ejecución de un paso), y se nota por* $C_1 \Rightarrow_{\Pi} C_2$*, si existe un multiconjunto de reglas, amr, aplicable a* C_1 *tal que* $\mu_2 = amr(\mu_1)$ *y* $M_2 = amr(M_1)$.

Una vez que sabemos realizar transiciones de un paso, reiterándolas obtenemos las computaciones del P sistema.

Definición 2.15 *Una computación,* $\mathcal{C}$*, de* Π *es una sucesión, posiblemente infinita, de configuraciones* $\mathcal{C}^0, \mathcal{C}^1, \ldots, \mathcal{C}^q, q \geq 0$*, tal que*

- $\mathcal{C}^0$ *es la configuración inicial de* Π.
- $\mathcal{C}^i \Rightarrow_{\Pi} \mathcal{C}^{i+1}$*, para todo* $i < q$.
- $q \in \mathbf{N}$ *y no existe ningún multiconjunto de reglas aplicable a* $\mathcal{C}^q$ *(en cuyo caso se dice que la computación* $\mathcal{C}$ *es de parada de longitud* q*, y que* $\mathcal{C}^q$ *es la configuración de parada de* $\mathcal{C}$*) o bien* $q = \infty$ *(en cuyo caso se dice que la computación* $\mathcal{C}$ *no es de parada).*

Diremos que Π *es un P sistema determinista si existe una única computación de* Π*. En caso contrario, diremos que es no determinista.*

Notación: Dada una computación $\mathcal{C}$ de Π, notaremos las configuraciones que la forman por $\mathcal{C}^q = \big(Ext(\mu^q), M^q\big)$, para cada $q \geq 0$.

La idea de los P sistemas con salida externa es que no conocemos lo que está ocurriendo dentro de la estructura de membranas, sino que sólo recogemos la información que expulsa al entorno externo. Ahora bien, esta información *sale* del sistema en un determinado orden, lo que nos permite obtener cadenas, en lugar de sólo conjuntos numéricos.

Definición 2.16 *Sea $\mathcal{C}$ una computación de parada de Π de longitud q. La salida de $\mathcal{C}$ se define como sigue:*

$$Output(\mathcal{C}) = \{w \in \Gamma^* : \exists w_1, \ldots, w_q \in \Gamma^* (w = w_1 \ldots w_q \wedge \forall l (1 \leq l \leq q \rightarrow w_l = M_{env}^l))\}$$

En el capítulo siguiente probaremos que toda máquina de Turing determinista puede ser simulada por un P sistema de transición con salida externa. Es interesante observar que un dispositivo computacional determinista (máquina de Turing) *con entrada* puede ser simulado por otro dispositivo computacional no determinista (P sistema) *sin entrada*; de tal manera que en la simulación el sistema sólo se utiliza el no determinismo durante la fase de generación del dato de entrada de la máquina.

2.4 P sistemas de decision

En esta sección presentamos otra variante de P sistemas con salida externa: los P sistemas de decisión. Éstos serán dispositivos de decisión de multiconjuntos: dispondrán de una membrana de entrada en la cual, al comienzo de cada computación, se introducirá un multiconjunto; la computación parará siempre, bien aceptando el multiconjunto, bien rechazándolo.

Definición 2.17 *Un* P sistema de decisión *de grado p es una tupla*

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

donde

- Σ *es un alfabeto finito no vacío, llamado* alfabeto de entrada.
- Γ *es un alfabeto finito no vacío tal que* $\Sigma \subseteq \Gamma$ *y sus elementos se llaman* objetos.

- μ_{Π} *es una estructura de membranas de grado* p.
- *La membrana de entrada de* Π *es* $i_{\Pi} \in V(\mu_{\Pi})$.
- $\mathcal{M}_i$ *es un multiconjunto sobre* $\Gamma \setminus \Sigma$ *asociada a la membrana* i, *para cada* $i = 1, \ldots, p$.
- R_i *es un subconjunto finito de* $TER(\Gamma, \mu_{\Pi})$ *asociado a la membrana* i *y* ρ_i *es un orden parcial estricto sobre* R_i, *para cada* $i = 1, \ldots, p$.
- *Existen dos objetos distinguidos* $Yes, No \in \Gamma \setminus \Sigma$.

Para formalizar la semántica de este modelo tenemos que establecer primero cuáles son sus configuraciones.

Definición 2.18 *Sea* Π *un P sistema de decisión.*

- *Una* configuración *de* Π *es un par* $C = \big(Ext(\mu), M\big)$ *tal que:*
  - $V(\mu) \subseteq V(\mu_{\Pi})$ *y la raíz de ambas estructuras de membranas es la misma.*
  - M *es una aplicación de* $V(Ext(\mu))$ *en* $\mathbf{M}(\Gamma)$. *Para cada nodo* $nd \in V(Ext(\mu))$ *notaremos* $M_{nd} = M(nd)$.
- *La* configuración inicial de Π para un multiconjunto $m \in \mathbf{M}(\Sigma)$ *es el par* $C_0 = \big(Ext(\mu), M\big)$ *donde:*
  - $\mu = \mu_{\Pi}$.
  - $M_{i_{\Pi}} = \mathcal{M}_{i_{\Pi}} + m$.
  - $M_i = \mathcal{M}_i$, *para cada* $i \neq i_{\Pi}$.
  - $M_{env} = \emptyset$.

Puesto que las reglas de los P sistemas de decisión son reglas de evolución de transición, los conceptos de regla aplicable a una configuración en una membrana, multiconjunto de reglas aplicable a una configuración y paso de transición de una configuración a otra coinciden con los ya definidos para los P sistemas de transición con salida externa.

Ahora bien, puesto que ahora tenemos una configuración inicial del P sistema para cada multiconjunto sobre el alfabeto de entrada, las computaciones de los P sistemas de decisión tendrán asociada una entrada.

Definición 2.19 *Sea Π un P sistema de decisión. Una computación, $\mathcal{C}$, de Π con entrada $m \in \mathbf{M}(\Sigma)$ es una sucesión, posiblemente infinita, de configuraciones $\mathcal{C}^0, \mathcal{C}^1, \ldots, \mathcal{C}^q, q \geq 0$, tal que*

- *$\mathcal{C}^0$ es la configuración inicial de Π para el multiconjunto m.*
- *$\mathcal{C}^i \Rightarrow_{\Pi} \mathcal{C}^{i+1}$, para todo $i < q$.*
- *$q \in \mathbf{N}$ y no existe ningún multiconjunto de reglas aplicable a $\mathcal{C}^q$ (en cuyo caso se dice que la computación $\mathcal{C}$ es de parada de longitud q, y que $\mathcal{C}^q$ es la configuración de parada de $\mathcal{C}$) o bien $q = \infty$ (en cuyo caso se dice que la computación $\mathcal{C}$ no es de parada).*

Diremos que Π es un P sistema determinista si para cada multiconjunto $m \in \mathbf{M}(\Sigma)$ existe una única computación de Π con entrada m. En caso contrario, diremos que es no determinista.

Notación: Dada una computación $\mathcal{C}$ de Π, notaremos las configuraciones que la forman por $\mathcal{C}^q = \left(Ext(\mu^q), M^q\right)$, para cada $q \geq 0$.

Pretendemos usar estos sistemas como dispositivos que nos permitan decidir sobre $\mathbf{M}(\Sigma)$, lo que realizaremos por medio de los objetos especiales Yes y No. Cuando uno de estos objetos sea expulsado al entorno externo, sabremos si debemos aceptar o no el multiconjunto que se ha introducido en la membrana de entrada. Es necesario entonces que todas las computaciones de un P sistema de decisión paren y, además, que sólamente sea en sus configuraciones de parada cuando el sistema expulse al entorno externo o bien el objeto Yes, o bien el objeto No.

Definición 2.20 *Un P sistema de decisión determinista, Π, se dice que es* válido *cuando se verifica lo siguiente:*

- *Todas sus computaciones son de parada.*
- *Para cada computación de Π sólo una regla de la forma $u \to v(ob, out)$, donde $ob = Yes$ u $ob = No$, se debe haber aplicado en la membrana piel de μ_{Π}, y además sólo en el último paso de la computación.*

En estas condiciones ya podemos definir el concepto de *aceptación* o *rechazo* de un multiconjunto por parte del sistema.

Definición 2.21 *Sea* Π *un P sistema de decisión determinista válido. Diremos que una configuración* $C = \left(Ext(\mu), M\right)$ *de* Π *es una* configuración de aceptación *(respectivamente,* configuración de rechazo*) si* $Yes \in M_{env}$ *(respectivamente,* $No \in M_{env}$*). Diremos que una computación* $\mathcal{C}$ *es de* aceptación *(respectivamente,* computación de rechazo*) de* Π *si su configuración de parada es una configuración de aceptación (respectivamente, configuración de rechazo).*

Definición 2.22 *Diremos que un P sistema de decisión determinista válido,* Π, acepta *(respectivamente,* rechaza*) un multiconjunto* $m \in \mathbf{M}(\Sigma)$ *si la computación de* Π *con entrada* m *es una computación de aceptación (respectivamente, de rechazo).*

En el capítulo seís estudiaremos cómo es posible caracterizar la conjetura $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ a través de la irresolubilidad de problemas **NP**–completos en familias de P sistemas de decisión deterministas y válidos.

Capítulo 3

Simulación de máquinas de Turing por P sistemas: diseño

En este capítulo mostramos cómo es posible simular una máquina de Turing determinista mediante un P sistema de transición con salida externa, de forma tal que el lenguaje generado por el P sistema coincide con el lenguaje reconocido por la máquina de Turing.

Primeramente fijamos qué variante de máquina de Turing determinista vamos a considerar. A continuación, dada una máquina de Turing, presentamos un P sistema que la simula, para terminar con una descripción detallada del funcionamiento de dicho P sistema.

Obsérvese que, como pretendemos simular un dispositivo reconocedor de lenguajes mediante un dispositivo generador de lenguajes, el P sistema debe generar no determinísticamente todas las posibles cadenas del lenguaje. No obstante, una vez generada una de dichas cadenas, la simulación del funcionamiento de la máquina de Turing sobre ella es totalmente determinista.

3.1 Máquinas de Turing

Una máquina de Turing determinista es una tupla

$$TM = (Q, q_0, q_N, q_Y, \Gamma, \Sigma, B, \triangleright, \delta)$$

donde

- $Q \cap \Gamma = \emptyset$.
- $Q = \{q_N, q_Y, q_0, \ldots, q_n\}$ es un conjunto finito de *estados*.
- $q_0 \in Q$ es el *estado inicial*; $q_N, q_Y \in Q$ son los *estados finales*: el primero es el *estado de rechazo*, el último es el *estado de aceptación*.
- $\Gamma = \{B, \triangleright, a_1, \ldots, a_m\}$ es el *alfabeto de trabajo*.
- $\Sigma = \{a_1, \ldots, a_p\}$, con $p \leq m$, es el *alfabeto de entrada*. Obsérvese que $\Sigma \subsetneq \Gamma$.
- $B \in \Gamma \setminus \Sigma$ es el *símbolo blanco*; $\triangleright \in \Gamma \setminus \Sigma$ es el *símbolo más a la izquierda*. Denotamos $a_B = B$ y $a_0 = \triangleright$.
- $\delta : \big(Q \setminus \{q_N, q_Y\}\big) \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$ es la *función de transición*.

TM tiene una sóla cinta divida en celdas, infinita hacia la derecha, pero con una primera celda. Cada celda contendrá o el símbolo blanco, indicando que la celda está vacía, u otro símbolo del alfabeto de trabajo. El símbolo $\triangleright$ jugará un papel especial en el sentido de que siempre marcará la celda más a la izquierda y no aparecerá en ningún otro lugar.

Los símbolos en la cinta se leen y escriben a través de una cabeza, que en cada momento analiza una y sólo una de las celdas. La cabeza se puede mover a lo largo de la cinta a la izquierda y a la derecha, o permanecer quieta.

La siguiente acción de la máquina está totalmente determinada, a través de la función de transición, δ, por su estado actual y por el símbolo leído por la cabeza. Denotemos, para $q_i \in Q \setminus \{q_N, q_Y\}$ y $a_j \in \Gamma$, $\delta(q_i, a_j) = (q_{Q(i,j)}, a_{A(i,j)}, D(i,j))$. Entonces, la función de transición debe satisfacer las siguientes condiciones:

$$(A(i,0) = 0 \wedge D(i,0) = 1) \wedge (A(i,j) \neq 0, \forall j \neq 0)$$

Esto es, el símbolo $\triangleright$ siempre dirige la cabeza hacia la derecha, y nunca es borrada de la celda más a la izquierda ni escrita en una celda distinta de ésa.

La función de transición, δ, es el «programa» de la máquina de Turing.

Una *descripción instantánea* de una máquina de Turing es una cadena uqv, donde $u \in \Gamma^*$, $q \in Q$ y $v \in \Gamma^*(\Gamma \setminus \{B\}) \cup \{B\}$. De esta forma identificamos el contenido de la cinta, el estado y la posición de la cabeza: analiza el primer símbolo de v.

Decimos que uqv es una descripción instantánea de *parada* si $q = q_Y$ (descripción instantánea de aceptación) o $q = q_N$ (descripción instantánea de rechazo). Dada $x \in \Sigma^*$, la descripción instantánea *inicial* sobre x, denotada por I_x, es $q_0 \triangleright x$.

Podemos definir, sobre el conjunto de las descripciones instantáneas de TM, la relación $\vdash_{TM}$ como sigue:

$$uaqbv \vdash_{TM} upacv \Leftrightarrow \delta(q,b) = (p,c,-1) \quad uaqb \vdash_{TM} upac \Leftrightarrow \delta(q,b) = (p,c,-1)$$
$$uaqbv \vdash_{TM} uapcb \Leftrightarrow \delta(q,b) = (p,c,0) \quad uaqb \vdash_{TM} uapc \Leftrightarrow \delta(q,b) = (p,c,0)$$
$$uaqbv \vdash_{TM} uacpv \Leftrightarrow \delta(q,b) = (p,c,1) \quad uaqb \vdash_{TM} uacpB \Leftrightarrow \delta(q,b) = (p,c,1)$$

donde $p, q \in Q$ y $a, b, c \in \Gamma$.

Sea $\vdash^*_{TM}$ la clausura reflexiva y transitiva de $\vdash_{TM}$.

Dado $x \in \Sigma^*$, decimos que TM para sobre x, denotado por $TM \downarrow x$, si existe una descripción instantánea de parada, I^f_x, tal que $I_x \vdash^*_{TM} I^f_x$ (I^f_x es entonces única). El lenguaje reconocido por TM se define por

$$L(TM) = \{x \in \Sigma^* : TM \downarrow x \wedge I^f_x \text{ es una descripción instantánea de aceptación}\}$$

3.2 Simulación mediante P sistemas

Sea TM una máquina de Turing. Podemos asociar a esta máquina un P sistema, Π_{TM}, de forma que el lenguaje generado por éste coincida con el lenguaje reconocido por aquélla.

Para generar este lenguaje, el P sistema Π_{TM} realizará, sobre todas las posibles cadenas de entrada para TM, que se generarán de una forma no determinista, una simulación completa del funcionamiento de la máquina de Turing, y devolverá como salida sólo aquellas cadenas aceptadas por TM. Así, las reglas de Π_{TM} se elegirán cuidadosamente de forma tal que su operación transcurrirá a través de ciertas fases principales:

I. Calcular, no determinísticamente, la cadena de entrada.

1. Leer el elemento en la celda analizada por la cabeza.
2. Calcular el nuevo elemento a escribir en la celda, mover la cabeza y cambiar de estado.

3. Borrar el viejo elemento y escribir el nuevo.

4. Comprobar si se ha alcanzado un estado final: si no, repetir desde la fase 1; si q_N es el estado final alcanzado, entonces entrar en un bucle infinito; si q_Y es el estado final alcanzado, entonces finalizar con la fase O.

O. En caso de que TM acepte a la cadena de entrada, devolver dicha cadena como salida.

Estas fases se llevarán a cabo en varios pasos pequeños, cada uno de ellos controlado por un grupo de reglas. Para evitar que reglas de distintos pasos se apliquen juntas, usaremos objetos de tipo s^- como *objetos prohibidores* y objetos de tipo s^+ como *objetos permitidores*; si s_j^- está presente en el P sistema, entonces las reglas para el paso j **no se pueden** ejecutar; si, en cambio, s_j^+ es el objeto presente, entonces las reglas para el paso j **deben** ser ejecutadas (si es posible). Por supuesto, en cualquier momento, para cada paso sólo el correspondiente objeto prohibidor o permitidor estará presente. También habrá siempre sólamente un símbolo permitidor al mismo tiempo.

Para indicar que queremos realizar un paso, usaremos objetos de tipo s como *objetos promotores*. Cuando s_j aparezca en el P sistema, transformará su correspondiente objeto prohibidor s_j^- en el permitidor s_j^+, dando lugar así a que las reglas para el paso j se apliquen. Entonces el objeto permitidor se transformará a sí mismo de nuevo en el objeto prohibidor, y en un objeto promotor adecuado.

Supongamos que tenemos $Q_{TM} = \{q_N, q_Y, q_0, \ldots, q_n\}$, $\Gamma_{TM} = \{B, \triangleright, a_1, \ldots, a_m\}$, $\Sigma_{TM} = \{a_1, \ldots, a_p\}$, con $p \leq m$, y $\delta_{TM}(q_i, a_j) = (q_{Q(i,j)}, a_{A(i,j)}, D(i,j))$ como conjunto de estados, alfabeto de trabajo, alfabeto de entrada y función de transición para TM, respectivamente. Recuérdese que denotamos $a_B = B$ and $a_0 = \triangleright$.

Construimos el P sistema con salida externa $\Pi_{TM} = (\Gamma, \mu_\pi, \mathcal{M}, (R, \rho))$ asociado a TM, donde

$$\begin{aligned}
\Gamma = {} & \{q_N, q_Y, h, h', s, s_I^-, s_I^+, s_I, s_9^-, s_9, s_O^-\} \cup \{q_i : 0 \leq i \leq n\} \cup \\
& \{a_i, d_i : 1 \leq i \leq p\} \cup \{b_i, b_i', b_i'', c_i : 0 \leq i \leq m\} \cup \\
& \{s_{a_i}, s_{a_i}^-, s_{a_i}^+ : 1 \leq i \leq p\} \cup \{s_i, s_i^-, s_i^+ : 1 \leq i \leq 8\} \\
\mu_\pi = {} & [_1]_1 \\
\mathcal{M} = {} & \{\{q_0, h, h, b_0, s_{a_1}^-, \ldots, s_{a_p}^-, s_I^-, s_1^-, \ldots, s_9^-, s_O^-, s\}\}
\end{aligned}$$

$$R = R_I \cup R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_O$$

y

$R_I = R_{I,1} \cup R_{I,2}$, con

$$R_{I,1} \equiv \begin{cases} s \to s_{a_i} \quad (1 \leq i \leq p) \\ s \to s_I \\ s_{a_i} s_{a_i}^- \to s_{a_i}^+ > s_{a_i}^- \to s_{a_i}^- > \begin{cases} h \to h^2 a_i b_i \\ s_{a_i}^+ \to s_{a_1}^- s \end{cases} \quad (1 \leq i \leq p) \end{cases}$$

$$R_{I,2} \equiv \quad s_I s_I^- \to s_I^+ > s_I^- \to s_I^- > \begin{cases} h \to \lambda \\ s_I^+ \to s_I^- s_1 \end{cases}$$

$R_1 = R_{1,1} \cup R_{1,2} \cup R_{1,3}$, con

$$R_{1,1} \equiv \quad s_1 s_1^- \to s_1^+ > s_1^- \to s_1^- > \begin{cases} h \to hh' \\ b_i \to b_i b_i' \quad (0 \leq i \leq m) \\ s_1^+ \to s_1^- s_2 \end{cases}$$

$$R_{1,2} \equiv \begin{cases} h' s_2 s_2^- \to s_2^+ > s_2 \to s_3 > s_2^- \to s_2^- > \\ \qquad > b_i'^2 \to b_i'' \quad (0 \leq i \leq m) > \begin{cases} b_i' \to \lambda \quad (0 \leq i \leq m) \\ b_i'' \to b_i' \quad (0 \leq i \leq m) \\ s_2^+ \to s_2^- s_2 \end{cases} \end{cases}$$

$$R_{1,3} \equiv \quad s_3 s_3^- \to s_3^+ > s_3^- \to s_3^- > \begin{cases} b_i'^2 \to \lambda \quad (0 \leq i \leq m) \\ s_3^+ \to s_3^- s_4 \end{cases}$$

$R_2 = R_{2,1} \cup R_{2,2}$, con

$$R_{2,1} \equiv \quad s_4 s_4^- \to s_4^+ > s_4^- \to s_4^- > \begin{cases} h \to hh' \\ s_4^+ \to s_4^- s_5 \end{cases}$$

$$R_{2,2} \equiv \quad s_5 s_5^- \to s_5^+ > s_5^- \to s_5^- > \begin{cases} \text{Reglas para la función} \\ \text{de transición} \\ s_5^+ \to s_5^- s_6 \end{cases}$$

Las reglas para la función de transición, δ_{TM}, son las siguientes:

Caso 1: Estado q_r, elemento $a_s \neq B$

Movimiento	*Reglas*
izquierda	$q_r b'_s h \to q_{Q(r,s)} b'_s c_{A(r,s)}$, si $A(r,s) \neq B$ $q_r b'_s h \to q_{Q(r,s)} b'_s$, si $A(r,s) = B$
igual	$q_r b'_s \to q_{Q(r,s)} b'_s c_{A(r,s)}$, si $A(r,s) \neq B$ $q_r b'_s \to q_{Q(r,s)} b'_s$, si $A(r,s) = B$
derecha	$q_r b'_s \to q_{Q(r,s)} b'_s c_{A(r,s)} h$, si $A(r,s) \neq B$ $q_r b'_s \to q_{Q(r,s)} b'_s h$, si $A(r,s) = B$

Caso 2: Estado q_r, sin elemento

Movimiento	*Reglas*
izquierda	$q_r h \to q_{Q(r,s)} c_{A(r,s)}$, si $A(r,s) \neq B$ $q_r h \to q_{Q(r,s)}$, si $A(r,s) = B$
igual	$q_r \to q_{Q(r,s)} c_{A(r,s)}$, si $A(r,s) \neq B$ $q_r \to q_{Q(r,s)}$, si $A(r,s) = B$
derecha	$q_r \to q_{Q(r,s)} c_{A(r,s)} h$, si $A(r,s) \neq B$ $q_r \to q_{Q(r,s)} h$, si $A(r,s) = B$

Para evitar conflictos, cada regla del caso 1 tiene una prioridad mayor que cada regla del caso 2.

$R_3 = R_{3,1} \cup R_{3,2}$, con

$$R_{3,1} \equiv \quad h' s_6 s_6^- \to s_6^+ > s_6 \to s_7 > s_6^- \to s_6^- > \begin{cases} b'_i \to b'^2_i \quad (0 \leq i \leq m) \\ c_i \to c_i^2 \quad (0 \leq i \leq m) \\ s_6^+ \to s_6^- s_6 \end{cases}$$

$$R_{3,2} \equiv \quad s_7 s_7^- \to s_7^+ > s_7^- \to s_7^- > \begin{cases} b_i b'_i \to \lambda \quad (0 \leq i \leq m) \\ c_i \to b_i \quad (0 \leq i \leq m) \\ s_7^+ \to s_7^- s_8 \end{cases}$$

$R_4 = R_{4,1} \cup R_{4,2}$, con

$$R_{4,1} \equiv \quad s_8 s_8^- \to s_8^+ > s_8^- \to s_8^- > \begin{cases} q_Y \to q_Y s_9, \quad q_N \to q_N \\ q_i \to q_i s_1 \quad (0 \leq i \leq n) \\ s_8^+ \to s_8^- \end{cases}$$

$$R_{4,2} \equiv \quad s_9 s_9^- \to \lambda > s_9^- \to s_9^- > \begin{cases} s_{a_1}^- \ldots s_{a_p}^- s_I^- s_1^- \ldots s_8^- s_O^- \to \lambda \\ q_Y \to \lambda \\ h \to \lambda \\ b_i \to \lambda \quad (0 \leq i \leq m) \end{cases}$$

$R_O = R_{O,1}$

$$R_{O,1} \equiv \quad \left\{ s_O^- \to s_O^- > a_i^2 \to d_i \quad (1 \leq i \leq p) \right. > \begin{cases} a_i \to (a_i, out) \quad (1 \leq i \leq p) \\ d_i \to a_i \quad (1 \leq i \leq p) \end{cases}$$

Para simular el funcionamiento de la máquina de Turing, es obvio que tenemos que fijar una representación para varios de sus elementos: cuál es el estado actual de la máquina de Turing, cuáles son los elementos en las celdas de la cinta, y qué celda está siendo analizada por la cabeza.

Para representar los estados usaremos los objetos $q_N, q_Y, q_0, \ldots, q_n$.

A lo largo de la simulación, los objetos $b_0, \ldots, b_m$ representarán, en base 2, qué celdas contienen los símbolos $a_0, \ldots, a_m$, respectivamente (nótese que, en cualquier momento, el número de celdas no vacías es finito); al generar la cadena de entrada, mantendremos una copia suya usando los objetos $a_1, \ldots, a_p$, de forma que podamos devolverla en el caso de que sea aceptada; estos objetos no evolucionarán a lo largo de la simulación del funcionamiento de la máquina de Turing; los objetos $b'_0, \ldots, b'_m, b''_0, \ldots, b''_m$ se usarán como copias de trabajo de los objetos anteriores; los objetos prima también servirán para guardar los símbolos leídos de las celdas, y los objetos $c_0, \ldots, c_m$ servirán para indicar los nuevos símbolos a escribir en ellas. Finalmente, los objetos $d_1, \ldots, d_p$ se usarán como copias de trabajo cuando se devuelva la salida de una computación exitosa de Π_{TM}.

La celda analizada por la cabeza, numerada a partir de cero, se representará

por el objeto h, en base uno. No obstante, para reducir el número de cálculos a realizar cuando se genera la cadena de entrada, usaremos h para representar en base 2, sólo en la primera fase, la celda de la cinta a escribir. El objeto h' se usará, cuando se necesite, como copia de trabajo del objeto h; se utilizará como contador.

Entre los objetos promotores, hay varios que merece la pena destacar: s sirve como mecanismo de elección; determina, en la fase de generación de la cadena de entrada, qué elemento del alfabeto de entrada escribir; por otro lado, si s_I está presente, entonces esa fase debe finalizar; s_1 comienza la simulación de un paso del funcionamiento de la máquina de Turing, mientras que s_8 la termina; finalmente, la presencia de s_9 marca el comienzo de la fase de salida.

3.3 Descripción de la simulación

Veamos con mayor detalle, de una manera informal, cómo trabaja este P sistema.

Debido a la presencia del objeto s en la configuración inicial del P sistema, la primera fase en la simulación en Π_{TM} de una computación de TM comienza con la ejecución de una regla del tipo $s \to s_{a_j}$, para algún a_j, (o la regla $s \to s_I$), y, simultáneamente, con la ejecución de reglas del tipo $s_j^- \to s_j^-$ que no alteran el contenido de la membrana piel.

Cada vez que el objeto s aparece en Π_{TM}, se activa el mecanismo de elección, produciendo la selección de algunos objetos que codifican símbolos del dato de entrada para la máquina de Turing, junto con la celda que ocupan en la cinta de TM.

La presencia del objeto s_I en el P sistema vendrá dado por la aplicación de la regla $s \to s_I$. Este objeto causará la eliminación de cualquier objeto h presente en la membrana (que a partir de ahora codificará, en base 1, la posición de la cabeza lectora de la máquina de Turing), indicando así que la cabeza está en la celda número cero, y permitirá la inserción del objeto s_1 en el P sistema.

En ese momento, la primera fase, que es la única no determinista, del funcionamiento del P sistema, encargada de la generación de la cadena de entrada de la máquina de Turing, terminará, comenzando entonces la simulación de la computación de TM sobre ese dato de entrada.

Fase I: Generación del dato de entrada Esta fase se lleva a cabo mediante las reglas del conjunto R_I.

Nótese que empezamos con dos copias de h, es decir, la primera celda a considerar para introducir el primer símbolo de la cadena de entrada es la celda número uno. Esto es porque la celda número cero está reservada para el símbolo $\triangleright$, que se codifica mediante el objeto b_0, el cual aparece en el P sistema desde el comienzo de la simulación.

A causa de las reglas de prohibición, ningún elemento se puede escribir y la fase I no puede concluirse. No obstante, el objeto s se transforma a sí mismo, no determinísticamente, o en un objeto promotor de tipo s_{a_j}, para algún j, $1 \leq j \leq p$, o en el objeto promotor s_I. Por tanto, el correspondiente objeto prohibidor cambiará en el permitidor, haciendo así posible «escribir el símbolo a_j» o finalizar la fase de generación de la cadena.

En el primer caso copiamos tantos a_j (para mantenerlos hasta la fase O) y b_j (para trabajar con ellos en las fases siguientes) como h estén presentes. También doblamos el número de objetos h para considerar la siguiente celda en la cinta (recuérdese que en esta fase representamos las celdas en base 2). Finalmente, el objeto permitidor $s^+_{a_j}$ se transforma a sí mismo en el prohibidor, $s^-_{a_j}$, y aparece un nuevo símbolo s, para elegir de nuevo la siguiente acción.

En el segundo caso, todos los objetos h se borran, comenzando la simulación de un paso de la máquina de Turing con la cabeza lectora empezando a partir de la celda número cero. Además, s^+_I se transforma a sí mismo en s^-_I y s_1, para inducir la ejecución de la fase 1.

La presencia del objeto s_1 en el P sistema ocasiona el comienzo de la simulación de un paso de la computación de la máquina de Turing sobre el dato de entrada, codificado apropiadamente en Π_{TM}, y, además, preservado por determinados objetos de tipo $a_1, \ldots, a_p$. De esta forma, se empieza la fase encargada de leer el elemento que la cabeza de la máquina de Turing está analizando en ese momento. Esta fase acabará cuando el objeto s_4 aparezca dentro de la membrana.

Fase 1: Leyendo el elemento Esta fase se lleva a cabo mediante las reglas del conjunto R_1.

Para decodificar el elemento leído por la cabeza a partir de los objetos dentro del P sistema, nos basamos en la siguiente propiedad:

El número x tiene un 1 como y-ésimo dígito en su representación en base 2 $\iff$ $quotient(x, 2^y) \equiv 1 \pmod 2$

Recuérdese que la celda que está siendo analizada por la cabeza se representa mediante el número de objetos h y las celdas no vacías con elementos a_0 a a_m se representan, en base 2, mediante el número de objetos b_0 a b_m, respectivamente. Supongamos que esos números son y, para la celda analizada, y $x_0, \ldots, x_m$, para las celdas no vacías. Entonces o sólo uno, si la celda número y es no vacía, o ninguno, si la celda número y está vacía, de estos últimos tiene un 1 como y-ésimo dígito en base 2.

Así, calculamos el cociente de x_0 a x_m cuando se dividen por 2^y y tomamos módulo 2. De esta forma tendremos el elemento en la celda número y si alguno de los resultados es 1 (y sólo puede ser como mucho uno de ellos) o una celda número y vacía (y, por tanto, ningún elemento) si no. Esto se realiza en tres pasos.

Primero, mediante las reglas de $R_{1,1}$, obtenemos copias de trabajo, h', $b'_0, \ldots, b'_m$, de los objetos $h, b_0, \ldots, b_m$, para hacer los cálculos sin cambiar el estatus de la (representación de la) máquina de Turing.

Las reglas de $R_{1,2}$ son las encargadas de la división por 2^y. Esto se hace dividiendo por 2 cuantas veces sea necesario, deshaciéndonos de los restos obtenidos cada vez. Tenemos que dividir tantas veces como indique el número de objetos h'. Por tanto, para transformar el objeto prohibidor en el permitidor, de forma que podamos realizar la división, no sólo necesitamos que el objeto promotor s_2 esté presente, sino también al menos algún h'. Si este no es el caso, entonces hemos terminado con el segundo paso y podemos continuar con el tercer paso. Así, s_2 se transforma a sí mismo en el objeto promotor para el siguiente paso, s_3.

En otro caso, llevamos a cabo la división cambiando primero pares de b'_j por un sólo b''_j. Entonces eliminamos los b'_j que quedan (los restos) y recuperamos b'_j a partir de b''_j. Finalmente, s_2^+ se cambia en s_2^- y en s_2, indicando de esta forma que queremos realizar otra división, si es posible (si quedan h').

El último paso es relativamente simple. Las reglas de $R_{1,3}$ sólo tienen que tomar módulo dos, y esto se hace eliminando tantos pares de b'_j como sea posible. Al final, si la celda número y contiene el símbolo a_k, $a_k \neq B$, entonces obtenemos un único b'_k en el P sistema, y ningún b'_j para todo $j \neq k$. En otro caso, si la celda y está vacía, entonces no obtenemos ningún b'_j, para cualquier $j = 0, \ldots, m$.

La presencia del objeto s_4 da lugar al comienzo de la siguiente fase, que consite en calcular el nuevo elemento que se va a escribir, mover la cabeza y cambiar el estado, de acuerdo con la función de transición, δ_{TM}. Esta fase finalizará cuando el objeto s_6 aparezca dentro de la membrana.

Fase 2: Calculando el nuevo elemento, moviendo la cabeza y cambiando el estado Esta fase se lleva a cabo mediante las reglas del conjunto R_2 y traduce la función de transición de la máquina de Turing.

Primeramente, preservamos qué celda estaba analizando la cabeza antes de aplicar la función de transición, de forma que la siguiente fase pueda realizarse correctamente. Esto se lleva a cabo mediante las reglas de $R_{2,1}$ simplemente haciendo una copia de los objetos h usando los objetos h'.

Entonces usamos las reglas de $R_{2,2}$, que se dividen en dos clases, dependiendo de si la celda número y está o no vacía. Las reglas para el último caso tienen mayor prioridad que las reglas para el primero, por lo que no hay ambigüedad; esto es, se aplicará una única regla asociada a la función de transición en cualquier caso.

Supongamos que el estado de la máquina de Turing es q_r y el elemento en la celda número y es a_s. Por lo tanto tenemos un objeto q_r y un objeto b'_s en el P sistema. Entonces existe una sóla regla que implica a estos dos objetos, la cual transforma q_r en $q_{Q(r,s)}$ (el nuevo estado), introduce un nuevo objeto $c_{A(r,s)}$ (el nuevo elemento) o ninguno, si la celda tiene que borrarse, pero conservando el antiguo b'_s y, finalmente: si la cabeza tiene que moverse a la izquierda, entonces borra un h; si tiene que moverse a la derecha, añade un h; si tiene que permanecer quieta, no hace nada.

Supongamos que el estado de la máquina de Turing es q_r y la celda número y está vacía. Por lo tanto tenemos un objeto q_r pero ningún objeto b'_s en el P sistema. Entonces existe una única regla que implica el primer objeto pero ninguno de los últimos, la cual transforma q_r en $q_{Q(r,s)}$ (el nuevo estado), introduce un nuevo objeto $c_{A(r,s)}$ (el nuevo elemento) o ninguno, si la celda tiene que estar aún vacía, y, finalmente: si la cabeza tiene que moverse a la izquierda, entonces borra un h; si tiene que moverse a la derecha, añade un h; y si tiene que permanecer quieta, no hace nada.

La presencia del objeto s_6 comienza la fase encargada de borrar el elemento leído por la cabeza y de escribir en su lugar el nuevo elemento calculado por

medio de las reglas asociadas a la función de transición. Esta fase terminará cuando el objeto s_8 aparezca dentro de la membrana.

Fase 3: Borrando el viejo elemento, escribiendo el nuevo Esta fase se lleva a cabo mediante las reglas del conjunto R_3.

En esta situación, el P sistema tiene estas clases de objetos: objetos h' que representan la celda que la cabeza **analizó**; objetos h que representan la celda que la cabeza **va a analizar**, posiblemente diferente de la anterior; un objeto b'_s, o ningún objeto de este tipo, que representa el elemento **leído** por la cabeza o blanco, respectivamente; un objeto c_t, o ningún objeto de este tipo, que representa el nuevo elemento **a escribir** en la celda antigua.

Como la representación de las celdas no vacías es en base 2, tenemos que hacer 2^y copias de los dos últimos objetos, b'_s y c_t. Nótese que no conocemos en principio qué objetos son, pero podemos sacar provecho del hecho de que existe como mucho sólo uno de cada tipo. Por tanto, podemos poner una regla para cada b'_j teniendo la seguridad de que sólo una de ellas, la que queremos, se ejecutará. Lo mismo se aplica a los objetos c_k.

El mecanismo para calcular las potencias es el mismo que el usado para calcular las divisiones sucesivas en la fase 1, y se realiza mediante las reglas de $R_{3,1}$. Tenemos que doblar el número de b'_j y c_k tantas veces como indique el número de objetos h'. En consecuencia, para que el objeto prohibidor s_6^- pueda cambiar en el correspondiente permitidor es necesario no sólo que el objeto promotor s_6 esté presente, sino también al menos un h'. Si este no es el caso, entonces hemos terminado con este paso y s_6 se transforma a sí mismo en s_7, para continuar con el siguiente paso. En otro caso, sustituimos cada b'_j y cada c_k por dos copias suyas y cambiamos s_6^+ por s_6^- y s_6 para indicar que tiene que hacerse otra multiplicación, si es posible.

Finalmente, las reglas de $R_{3,2}$ se usan para borrar el viejo símbolo de la celda número y y escribir en ella el nuevo. Esto se hace cancelando cada copia de b'_j con un b_j y transformando cada copia de c_k en b_k.

La siguiente fase se ocupa del problema de comprobar si se ha alcanzado un estado final, realizando una de las siguientes acciones:

(a) Si no se ha alcanzado un estado final, entonces comienza de nuevo a partir de la fase 1, para simular otro paso de la computación de la máquina de

Turing. Esto se hace introduciendo el objeto promotor s_1 dentro de la membrana.

(b) Si se ha alcanzado el estado q_N, entonces entra en un bucle infinito.

(c) Si se ha alcanzado el estado q_Y, entonces inserta el objeto promotor s_9 para continuar con la fase O final.

Esta fase se inicia con la presencia del objeto promotor s_8 y se termina con la ejecución de la regla correspondiente al caso, de los tres anteriores, que ha tenido lugar y de la regla $s_8^+ \to s_8^-$.

Fase 4: Comprobando si el estado es final Esta fase se lleva a cabo mediante las reglas del conjunto R_4. Esta es la última fase de la simulación de la máquina de Turing. Detecta si se ha alcanzado o no un estado final. En el primer caso devuelve la cadena de entrada como salida externa si fue el estado de aceptación, o entra en un bucle infinito si fue el estado de rechazo. En el último caso, continuamos la simulación a partir de la fase 1.

Las reglas de $R_{4,1}$ se usan para distinguir entre todas las posibilidades: si q_Y está presente, lo que significa que hemos alcanzado el estado final de aceptación, entonces continuamos con la siguiente fase, en lugar de con la fase 1; si q_N está presente, lo que significa que hemos alcanzado el estado final de rechazo, entonces no introducimos ningún objeto promotor. El efecto de esto es que la simulación de la máquina de Turing se detiene, pero **el funcionamiento del P sistema no**. Esto es así porque todas las reglas de prohibición, y ninguna otra, se pueden aplicar y son aplicadas. Así, no hay salida del P sistema en este caso; si cualquiera de $q_0, \ldots, q_n$ está presente, entonces la simulación de la máquina de Turing no ha concluido. Por tanto, introducimos el objeto promotor s_1 para comenzar de nuevo a partir de la fase 1. En todos los casos el objeto permitidor s_8^+ se transforma a sí mismo en el objeto prohibidor s_8^-.

Las reglas de $R_{4,2}$ sólo se aplican cuando se ha alcanzado el estado final de aceptación. Su única utilidad es eliminar todos los objetos del P sistema excepto aquellos que codifican la cadena de entrada. Esto se hace para que ninguna regla, excepto las que devuelven la salida, se pueda aplicar y el P sistema pare. A causa del funcionamiento de estas reglas, no es necesario un objeto permitidor s_9^+, sino que el promotor y el prohibidor se cancelan uno con el otro.

La presencia del objeto s_9 en el P sistema significa que se ha alcanzado el estado de aceptación en la computación de TM que se está simulando. Ello comenzará la fase final en la que el dato de entrada para la máquina de Turing se devolverá como cadena de salida para la computación de Π_{TM} considerada.

Fase O: Produciendo el dato de salida Esta fase se lleva a cabo mediante las reglas del conjunto R_O, las cuales se usan para decodificar la cadena de entrada y expulsarla, en orden, de la membrana piel.

Como la cadena está codificada en base 2, sólamente tenemos que dividir por 2, repetidamente, el número de símbolos a_j presentes. Cada vez, excepto la primera (porque no expulsamos el símbolo $\triangleright$, que siempre ocupa la celda número cero), uno y sólo uno de estos números es impar, lo que indica la siguiente letra de la cadena de entrada a expulsar. Terminamos cuando no queden más objetos.

Nótese que no necesitamos ningún objeto promotor ni permitidor para esta fase, porque si la alcanzamos todos los objetos excepto los a_j habrán desaparecido.

Primero aplicamos las reglas que sustituyen todos los posibles pares de a_j con un sólo d_j, dividiendo así por dos el número de objetos presentes. Entonces sólo un a_k permanece (excepto la primera vez), para algún k, el cual se expulsa fuera de la membrana. Los d_j se transforman en los correspondientes a_j para repetir la misma operación una y otra vez hasta que no quede ningún objeto en la membrana piel.

Capítulo 4

Simulación de máquinas de Turing por P sistemas: verificación

En el capítulo anterior, dada una máquina de Turing determinista arbitraria (con una cinta) se ha diseñado un P sistema con salida externa que *simula* dicha máquina. Es importante resaltar que basta *un* P sistema con salida externa (y, por tanto, un P sistema sin entrada) para simular *una* máquina de Turing determinista. Más aún, dicho P sistema actúa en dos fases bien diferenciadas: en la primera fase, el P sistema genera de manera no determinista un dato de entrada arbitrario de la máquina de Turing; en la segunda fase, el P sistema trabaja de manera determinista y simula todos y cada uno de los pasos que realiza la máquina de Turing sobre el dato de entrada generado por el P sistema en la fase no determinista. Para ello se ha descrito de manera exhaustiva el P sistema asociado a cada máquina de Turing determinista, así como el comportamiento del mismo que justifica informalmente la simulación.

El objetivo de este capítulo es demostrar que el P sistema con salida externa, Π_{TM}, asociado a cada máquina de Turing determinista, TM, simula realmente el funcionamiento de dicha máquina. Es decir, hemos de probar que para cada $w \in \Sigma^*_{TM}$ se verifica que el dato w es aceptado por TM si y sólo si existe una computación de Π_{TM} cuya salida en el entorno codifica la cadena w.

Para ello, hemos de demostrar en primer lugar que dada una máquina de Turing cada dato de entrada de la máquina (y sólo esos) es generado por el P sistema en la fase no determinista. Una vez seleccionado un dato de entrada de la máquina por el P sistema, se ha de probar que cada paso de transición de la computación de la máquina de Turing con dicho dato de entrada es simulado de manera determinista por el P sistema, usando las codificaciones adecuadas. Seguidamente se demuestra la simulación por el P sistema del chequeo de cada configuración de la máquina a fin de detectar si corresponde o no a un estado de aceptación. Finalmente se demuestra que tras llegar a una configuración de aceptación en la máquina de Turing, el P sistema devuelve el dato de entrada de la máquina.

Hay que hacer notar que en el proceso de verificación de la simulación de una máquina de Turing determinista por un P sistema con salida externa, se va calculando el número de pasos que realiza el P sistema para efectuar cada etapa de la simulación. Esto nos proporcionará el coste en tiempo de la simulación realizada.

4.1 Etapa de generación del dato de entrada

Recordemos que si $w = a_{i_1} \dots a_{i_k}$ es un dato de entrada de la máquina de Turing TM, entonces el P sistema asociado debe generar el multiconjunto que codifica en base 2 dicha cadena; es decir, el multiconjunto $A_k = \{\{a_{i_1}^2 a_{i_2}^{2^2} \dots a_{i_k}^{2^k}\}\}$. Ahora bien, por cuestiones técnicas se necesita una copia de trabajo y, por tanto, hemos de añadir el multiconjunto $B_k = \{\{b_{i_1}^2 b_{i_2}^{2^2} \dots b_{i_k}^{2^k}\}\}$. El símbolo $\triangleright$ de la primera casilla de la cinta está codificado por b_0. El símbolo h jugará un doble papel a lo largo de la ejecución del P sistema. En la fase no determinista de generación del dato de entrada, dicho símbolo codificará en base 2 la casilla donde se va a escribir un símbolo (para ello, utilizaremos un símbolo auxiliar h' cuando sea necesario para utilizarlo como contador). En la fase determinista, h codificará en base 1 la casilla que está siendo analizada por la cabeza de trabajo. Las casillas de la cinta están enumeradas a partir del cero.

En los multiconjuntos que se van generando en la única membrana del P sistema durante esta primera fase, deben aparecer símbolos que actúan de prohibidores (s^-), permitidores (s^+) y promotores (s). La fase no determinista de generación del dato de entrada finalizará con la aparición de un objeto

promotor s_1. Para facilitar la notación, convendremos que:

$$S = \{\{s_{a_1}^- \dots s_{a_p}^- s_I^- s_1^- \dots s_9^- s_O^-\}\}$$

y además,

$$S(a_i) = S - \{s_{a_i}^-\};\ S(I) = S - \{s_I^-\};\ S(j) = S - \{s_j^-\};\ S(O) = S - \{s_O^-\}$$

Nota: Usaremos la notación de cadena para los multiconjuntos. Además, cuando describamos una configuración arbitraria, salvo que el entorno externo no sea vacío explicitaremos sólamente el contenido de la única membrana del P sistema.

En primer lugar, veamos que todo dato de entrada de la máquina de Turing puede ser debidamente codificado en la membrana del P sistema asociado. Más aún, veamos que un dato de entrada de tamaño k de la máquina de Turing es generado al cabo de $3k+3$ pasos de una computación del P sistema (instante en el que finaliza la fase de generación del dato de entrada).

Recordemos que M_i^j es el contenido de la membrana i en la configuración j-ésima de la computación.

Teorema 4.1 *Para cada $w = a_{i_1} \dots a_{i_k} \in \Sigma_{TM}^*$ existe, al menos, una computación, $\mathcal{C}$, de Π_{TM} tal que:*

$$\begin{aligned} M_1^{3k} &= q_0 h^{2^{k+1}} b_0 s + B_k + A_k + S \\ M_1^{3k+1} &= q_0 h^{2^{k+1}} b_0 s_I + B_k + A_k + S \\ M_1^{3k+2} &= q_0 h^{2^{k+1}} b_0 s_I^+ + B_k + A_k + S(I) \\ M_1^{3k+3} &= q_0 b_0 s_1 + B_k + A_k + S \end{aligned}$$

Demostración:
Por inducción en la longitud, k, de la palabra w.

j	M_1^j	Multiconjunto de reglas aplicable
0	$q_0 h^2 b_0 s + S$	$s \to s_I$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \le i \le 9)$; $s_O^- \to s_O^-$
1	$q_0 h^2 b_0 s_I + S$	$s_I s_I^- \to s_I^+$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_i^- \to s_i^-$ $(1 \le i \le 9)$; $s_O^- \to s_O^-$
2	$q_0 h^2 b_0 s_I^+ + S(I)$	$h \to \lambda$; $s_I^+ \to s_I^- s_1$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_i^- \to s_i^-$ $(1 \le i \le 9)$; $s_O^- \to s_O^-$
3	$q_0 b_0 s_1 + S$	

Esto prueba el resultado para el caso base, $k = 0$.
Sea $k \in \mathbf{N}$ y supongamos cierto el resultado para k. Sea $w = a_{i_1} \dots a_{i_k} a_{i_{k+1}} \in \Sigma^*_{TM}$. De la hipótesis de inducción se deduce que

$$\begin{aligned} M_1^{3k} &= q_0 h^{2^{k+1}} b_0 s + B_k + A_k + S \\ M_1^{3k+1} &= q_0 h^{2^{k+1}} b_0 s_I + B_k + A_k + S \\ M_1^{3k+2} &= q_0 h^{2^{k+1}} b_0 s_I^+ + B_k + A_k + S(I) \\ M_1^{3k+3} &= q_0 b_0 s_1 + B_k + A_k + S \end{aligned}$$

A partir de la configuración $\mathcal{C}^{3k}$ podemos derivar una nueva computación de la siguiente forma:

j	M_1^j	Multiconjunto de reglas aplicable
$3k$	$q_0 h^{2^{k+1}} b_0 s + B_k + A_k + S$	$s \to s_{a_{i_{k+1}}}$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9)$ $s_O^- \to s_O^-$
$3k+1$	$q_0 h^{2^{k+1}} b_0 s_{a_{i_{k+1}}} + B_k + A_k + S$	$s_{a_{i_{k+1}}} s_{a_{i_{k+1}}}^- \to s_{a_{i_{k+1}}}^+$ $s_O^- \to s_O^-$; $s_I^- \to s_I^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p \wedge j \ne i_{k+1})$ $s_i^- \to s_i^-$ $(1 \le i \le 9)$
$3k+2$	$q_0 h^{2^{k+1}} b_0 s_{a_{i_{k+1}}}^+ + B_k + A_k + S(a_{i_{k+1}})$	$h \to h^2 a_{i_{k+1}} b_{i_{k+1}}$; $s_{a_{i_{k+1}}}^+ \to s_{a_{i_{k+1}}}^- s$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9)$ $s_O^- \to s_O^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p \wedge j \ne i_{k+1})$
$3k+3$	$q_0 h^{2^{k+2}} b_0 s + B_{k+1} + A_{k+1} + S$	$s \to s_I$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9)$ $s_O^- \to s_O^-$
$3k+4$	$q_0 h^{2^{k+2}} b_0 s_I + B_{k+1} + A_{k+1} + S$	$s_I s_I^- \to s_I^+$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_i^- \to s_i^-$ $(1 \le i \le 9)$; $s_O^- \to s_O^-$
$3k+5$	$q_0 h^{2^{k+2}} b_0 s_I^+ + B_{k+1} + A_{k+1} + S$	$h \to \lambda$; $s_I^+ \to s_I^- s_1$; $s_O^- \to s_O^-$ $s_i^- \to s_i^-$ $(1 \le i \le 9)$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$
$3k+6$	$q_0 b_0 s_1 + B_{k+1} + A_{k+1} + S$	

Lo que completa la prueba del paso inductivo y, por tanto, la del teorema.

□

Seguidamente vamos a probar que en cada configuración de cualquier computación del P sistema existe, a lo sumo, un símbolo permitidor (s^+) o promotor (s) de esta fase.

Lema 4.2 *Sea* $A = \{\{s, s_I,\ s_I^+, s_{a_1}, \ldots, s_{a_p}, s_{a_1}^+, \ldots, s_{a_p}^+\}\}$. *Sea* $\mathcal{C}$ *una computación de* Π_{TM}. *Para cada* $k \in \mathbf{N}$ *se verifica que* $|M_1^k \cap A| \leq 1$.

Demostración:
Por inducción débil sobre k. El caso base, $k = 0$, es trivial, ya que $M_1^0 \cap A = \{s\}$. Sea $k \in \mathbf{N}$ y supongamos cierto el resultado para k. Sea $\mathcal{C}$ una computación de Π_{TM}. De la hipótesis de inducción resulta que $|M_1^k \cap A| \leq 1$.

- Si $M_1^k \cap A = \emptyset$, entonces $M_1^{k+1} \cap A = \emptyset$, ya que los símbolos permitidores sólo pueden ser generados a partir de símbolos promotores, y los símbolos promotores a partir de los permitidores.
- Si $M_1^k \cap A = \{s\}$, entonces
  $$M_1^{k+1} \cap A = \{s_I\} \vee \exists j\ (1 \leq j \leq p \wedge M_1^{k+1} \cap A = \{s_{a_j}\}$$
- Si $M_1^k \cap A = \{s_I\}$, entonces $M_1^{k+1} \cap A = \{s_I^+\}$.
- Si $M_1^k \cap A = \{s_I^+\}$, entonces $M_1^{k+1} \cap A = \emptyset$.
- Si $M_1^k \cap A = \{s_{a_j}\}$, entonces $M_1^{k+1} \cap A = \{s_{a_j}^+\}$.
- Si $M_1^k \cap A = \{s_{a_j}^+\}$, entonces $M_1^{k+1} \cap A = \{s\}$.

□

A continuación vamos a probar que si en un cierto momento de una computación del P sistema se ejecuta una regla del tipo $s \to s_I$, entonces en ningún instante posterior se volverá a ejecutar dicha regla ni ninguna del tipo $s \to s_{a_j}$.

Lema 4.3 *Sea* $\mathcal{C}$ *una computación de* Π_{TM} *tal que en el paso* t $(t \geq 1)$ *de* $\mathcal{C}$ *se dispara la regla* $s \to s_I$. *Entonces para cada* $k > t$ *se tiene que en el paso* k*-ésimo no se puede ejecutar la regla* $s \to s_I$ *ni la regla* $s \to s_{a_j}$, *con* $1 \leq j \leq p$.

Demostración:
Teniendo presente que en el paso t $(t \geq 1)$ se ejecuta la regla $s \to s_I$, del lema 4.2 resulta que $M_1^{t-1} \cap A = \{s\}$ y que $M_1^t \cap A = \{s_I\}$. De la estructura sintáctica de las reglas del P sistema resulta que $M_1^{t+1} \cap A = \{s_I^+\}$ y $M_1^{t+2} \cap A = \emptyset$. De donde se deduce que $M_1^k \cap A = \emptyset$, para cada $k > t + 2$, ya que con la aparición del objeto s_1 en la membrana del P sistema finaliza la fase de generación y nunca más volverán a aparecer símbolos s y s_I en la membrana.

Por tanto, hemos probado que $\forall k \geq t$ $(s \notin M_1^k)$. Luego, en ningún instante posterior a t en la computación $\mathcal{C}$ se dispara la regla $s \to s_I$, ya que esa regla no será aplicable en $\mathcal{C}^k$, para cada $k \geq t$. De manera análoga, las reglas $s \to s_{a_j}$ (con $1 \leq j \leq p$) tampoco serán aplicables en $\mathcal{C}^k$, para cada $k \geq t$.

□

Veamos seguidamente que a lo largo de la ejecución del P sistema, las reglas del tipo $s \to s_{a_j}$ ó $s \to s_I$ sólo se disparan en instantes del tipo $3k+1$, para ciertos números naturales $k \geq 1$.

Lema 4.4 *Sea $\mathcal{C}$ una computación de Π_{TM}. Para cada $t \geq 1$ se verifica que si en el paso t de la computación $\mathcal{C}$ se dispara una regla del tipo $s \to s_{a_j}$, entonces existe $k \in \mathbf{N}$ tal que $t = 3k+1$.*

Demostración:
Por inducción fuerte sobre t. El caso base, $t = 1$, es trivial (basta tomar $k = 0$).

Sea $t \geq 1$ y supongamos que el resultado es cierto para cada $r \leq t$ tal que $r \geq 1$. Supongamos que en el paso $t+1$ de la computación $\mathcal{C}$ se ha disparado una regla del tipo $s \to s_{a_j}$ (para un cierto j con $1 \leq j \leq p$). Entonces $M_1^t \cap A = \{s\}$.

Como $|M_1^{t+1} \cap A| \leq 1$ y $M_1^t \cap A = \{s\}$ resulta que existe j' tal que $1 \leq j' \leq p$ y $M_1^{t-1} \cap A = \{s_{a_{j'}}^+\}$. Teniendo presente que $|M_1^{t-2} \cap A| \leq 1$ y que $M_1^{t-1} \cap A = \{s_{a_{j'}}^+\}$ resulta que $M_1^{t-2} \cap A = \{s_{a_{j'}}\}$ (luego, $t-2 > 0$).

Como $|M_1^{t-3} \cap A| \leq 1$ resulta que en el paso $t-2$ se ha disparado la regla $s \to s_{a_{j'}}$.

Ahora bien, se tiene que $t+1 \geq 4$, ya que la segunda vez que se ejecuta una regla del tipo $s \to s_{a_j}$ es en el paso 4. Luego, $1 \leq t-2 \leq t$. Por tanto, de la hipótesis de inducción se deduce que existe $k \in \mathbf{N}$ tal que $t-2 = 3k+1$. Por tanto, $t+1 = 3(k+1)+1$.

□

Lema 4.5 *Sea $\mathcal{C}$ una computación de Π_{TM}. Para cada $t \geq 1$ se verifica que si en el paso t de la computación $\mathcal{C}$ se dispara la regla $s \to s_I$, entonces existe $k \in \mathbf{N}$ tal que $t = 3k+1$.*

Demostración:
Si $t = 1$, entonces el resultado es inmediato (basta tomar $k = 0$).

Supongamos que $t \geq 2$. Como en el paso t de la computación $\mathcal{C}$ se ejecuta la regla $s \to s_I$, se verificará que $M_1^{t-1} \cap A = \{s\}$. Ahora bien, teniendo presente que la regla $s \to s_I$ nunca se ha disparado antes de ejecutar el paso t de $\mathcal{C}$ (consecuencia del lema 4.3), y que $M_1^0 \cap A = \{s\}$, resulta que en algún paso anterior a t se ha debido ejecutar una regla del tipo $s \to s_{a_j}$, para algún j con $1 \leq j \leq p$. Por tanto, $t \geq 4$ ya que la ejecución de una regla del tipo $s \to s_{a_j}$ conlleva ejecutar en pasos sucesivos las reglas $s_{a_j} s_{a_j}^- \to s_{a_j}^+$ y $s_{a_j}^+ \to s_{a_j}^- s$.

Como $M_1^{t-1} \cap A = \{s\}$ existirá j tal que $1 \leq j \leq p$ y $M_1^{t-2} \cap A = \{s_{a_j}^+\}$. Por tanto, $M_1^{t-3} \cap A = \{s_{a_j}\}$. De donde se deduce que en el paso $t-3$ de la computación de $\mathcal{C}$ se ha ejecutado la regla $s \to s_{a_j}$. Por el lema 4.4, existe $k \in \mathbf{N}$ tal que $t - 3 = 3k + 1$. Luego, $t = 3(k+1) + 1$.

□

Veamos que en la fase de generación del dato de entrada, un punto crítico se produce en el momento en que se dispara la regla $s \to s_I$. Concretamente vamos a probar que si el disparo se produce en el instante $3k+1$, entonces dicha fase finaliza en el instante $3k + 3$ generando un dato de entrada de tamaño k. Para ello, establecemos previamente un lema.

Lema 4.6 *Sea $\mathcal{C}$ una computación de Π_{TM} tal que en el instante $t \geq 1$ se dispara la regla $s \to s_I$. Sea $k \in \mathbf{N}$ tal que $t = 3k + 1$. Entonces, para cada $r < k$ existen objetos $a_{i_1}, \ldots, a_{i_r}, a_{i_{r+1}}, b_{i_1}, \ldots b_{i_r} b_{i_{r+1}}$ tales que*

$$\begin{aligned}
M_1^{3r} &= q_0 h^{2^{r+1}} b_0 s + B_r + A_r + S \\
M_1^{3r+1} &= q_0 h^{2^{r+1}} b_0 s_{a_{i_{r+1}}} + B_r + A_r + S \\
M_1^{3r+2} &= q_0 h^{2^{r+1}} b_0 s_{a_{i_{r+1}}}^+ + B_r + A_r + S(a_{i_{r+1}}) \\
M_1^{3r+3} &= q_0 h^{2^{r+2}} b_0 s + B_{r+1} + A_{r+1} + S
\end{aligned}$$

Demostración:
Por inducción débil sobre r. Para probar el caso base, $r = 0$, tengamos presente que $M_1^0 = q_0 h^2 b_0 s + S$. Como la regla $s \to s_I$ únicamente se dispara en el instante $t = 3k + 1$, existirá i_1 con $1 \leq i_1 \leq p$ tal que en el paso 1 de $\mathcal{C}$ se dispara la regla $s \to s_{a_{i_1}}$. Entonces

j	M_1^j	Multiconjunto de reglas aplicable
0	$q_0h^2b_0s+S$	$s \to s_{a_{i_1}}\ (1 \le i_1 \le p)$
1	$q_0h^2b_0s_{a_{i_1}}+S$	$s_{a_{i_1}}s^-_{a_{i_1}} \to s^+_{a_{i_1}};\ s^-_I \to s^-_I;\ s^-_i \to s^-_i\ (1 \le i \le 9)$ $s^-_O \to s^-_O;\ s^-_{a_j} \to s^-_{a_j}\ (1 \le j \le p \wedge j \ne i_1)$
2	$q_0h^2b_0s^+_{a_{i_1}}+S(a_{i_1})$	$h^2 \to h^2a_{i_1}b_{i_1};\ s^+_{a_{i_1}} \to s^-_{a_{i_1}}s;\ s^-_I \to s^-_I$ $s^-_O \to s^-_O;\ s^-_i \to s^-_i\ (1 \le i \le 9)$ $s^-_{a_j} \to s^-_{a_j}\ (1 \le j \le p \wedge j \ne i_1)$
3	$q_0h^{2^2}b_0s+B_1+A_1+S$	

Lo que completa la prueba del resultado para el caso $r = 0$.

Sea r tal que $r < k-1$ y supongamos que el resultado es cierto para r. Entonces se verifica que existen objetos $a_{i_1} \dots a_{i_r}a_{i_{r+1}}b_{i_1} \dots b_{i_r}b_{i_{r+1}}$ tales que

$M_1^{3r+3} = q_0h^{2^{r+2}}b_0s + B_{r+1} + A_{r+1} + S$

Teniendo presente que $r+1 < k$ resulta que $3r+4 < 3k+1 = t$. Es decir, en el paso $3r+4$ no se ejecuta la regla $s \to s_I$. Por tanto, existirá i_{r+2} (con $1 \le i_{r+2} \le p$) tal que en el paso $3r+4$ de la computación $\mathcal{C}$ se ejecutarán las reglas

$s \to s_{a_{i_{r+2}}};\ s^-_{a_j} \to s^-_{a_j}\ (1 \le j \le p);\ s^-_I \to s^-_I;\ s^-_i \to s^-_i\ (1 \le i \le 9); s^-_O \to s^-_O$

Entonces

j	M_1^j	Multiconjunto de reglas aplicable
$3r+3$	$q_0h^{2^{r+2}}b_0s+B_{r+1}+A_{r+1}+S$	$s \to s_{a_{i_{r+2}}};\ s^-_{a_j} \to s^-_{a_j}\ (1 \le j \le p)$ $s^-_I \to s^-_I;\ s^-_i \to s^-_i\ (1 \le i \le 9)$ $s^-_O \to s^-_O$
$3r+4$	$q_0h^{2^{r+2}}b_0s_{a_{i_{r+2}}} + B_{r+1} + A_{r+1} + S$	$s_{a_{i_{r+2}}}s^-_{a_{i_{r+2}}} \to s^+_{a_{i_{r+2}}};\ s^-_I \to s^-_I$ $s^-_{a_j} \to s^-_{a_j}\ (1 \le j \le p \wedge j \ne i_{r+2})$ $s^-_i \to s^-_i\ (1 \le i \le 9); s^-_O \to s^-_O$
$3r+5$	$q_0h^{2^{r+2}}b_0s^+_{a_{i_{r+2}}} + B_{r+1} + A_{r+1} + S(a_{i_{r+2}})$	$h \to h^2a_{i_{r+2}}b_{i_{r+2}};\ s^+_{a_{i_{r+2}}} \to s^-_{a_{i_{r+2}}}s$ $s^-_{a_j} \to s^-_{a_j}\ (1 \le j \le p \wedge j \ne i_{r+2})$ $s^-_I \to s^-_I;\ s^-_i \to s^-_i\ (1 \le i \le 9)$ $s^-_O \to s^-_O$
$3r+6$	$q_0h^{2^{r+3}}b_0s+B_{r+2}+A_{r+2}+S$	

Lo que completa la demostración del lema.

□

Teorema 4.7 *Sea $\mathcal{C}$ una computación de Π_{TM}. Supongamos que en dicha computación la regla $s \to s_I$ se dispara en el instante $t \geq 1$. Entonces existe $k \in \mathbf{N}$ y existen objetos $a_{i_1}, \ldots, a_{i_k}, b_{i_1}, \ldots b_{i_k}$ tales que $t = 3k+1$ y, además,*

$$\begin{aligned} M_1^{3k} &= q_0 h^{2^{k+1}} b_0 s + B_k + A_k + S \\ M_1^{3k+1} &= q_0 h^{2^{k+1}} b_0 s_I + B_k + A_k + S \\ M_1^{3k+2} &= q_0 h^{2^{k+1}} b_0 s_I^+ + B_k + A_k + S(I) \\ M_1^{3k+3} &= q_0 b_0 s_1 + B_k + A_k + S \end{aligned}$$

Demostración:
Sea $\mathcal{C}$ una computación de Π_{TM}. Entonces $M_1^0 = q_0 h^2 b_0 s + S$.
Si $t \geq 1$ es el instante en que se dispara la regla $s \to s_I$ a lo largo de la ejecución de $\mathcal{C}$, entonces por el lema 4.5 existe $k \in \mathbf{N}$ tal que $t = 3k+1$. Además, aplicando el lema 4.6 (tomando $r = k-1$) se deduce la existencia de objetos $a_{i_1}, \ldots, a_{i_k}, b_{i_1}, \ldots b_{i_k}$ tales que

$$\begin{aligned} M_1^{3r} &= q_0 h^{2^{r+1}} b_0 s + B_r + A_r + S \\ M_1^{3r+1} &= q_0 h^{2^{r+1}} b_0 s_{a_{i_{r+1}}} + B_r + A_r + S \\ M_1^{3r+2} &= q_0 h^{2^{r+1}} b_0 s^+_{a_{i_{r+1}}} + B_r + A_r + S(a_{i_{r+1}}) \\ M_1^{3r+3} &= q_0 h^{2^{r+2}} b_0 s + B_{r+1} + A_{r+1} + S = q_0 h^{k+1} b_0 s + B_k + A_k + S \end{aligned}$$

Teniendo presente que en el paso $t = 3k+1$ se dispara la regla $s \to s_I$, resulta que

j	M_1^j	Multiconjunto de reglas aplicable
$3k$	$q_0 h^{k+1} b_0 s + B_k + A_k + S$	$s \to s_I;\ s^-_{a_j} \to s^-_{a_j}\ (1 \leq j \leq p)$ $s^-_I \to s^-_I;\ s^-_O \to s^-_O$ $s^-_i \to s^-_i\ (1 \leq i \leq 9)$
$3k+1$	$q_0 h^{k+1} b_0 s_I + B_k + A_k + S$	$s_I s^-_I \to s^+_I;\ s^-_{a_j} \to s^-_{a_j}\ (1 \leq j \leq p)$ $s^-_i \to s^-_i\ (1 \leq i \leq 9); s^-_O \to s^-_O$
$3k+2$	$q_0 h^{k+1} b_0 s^+_I + B_k + A_k + S(I)$	$h \to \lambda;\ s^+_I \to s^-_I s_1;\ s^-_O \to s^-_O$ $s^-_{a_j} \to s^-_{a_j}\ (1 \leq j \leq p)$ $s^-_i \to s^-_i\ (1 \leq i \leq 9)$
$3k+3$	$q_0 b_0 s_1 + B_k + A_k + S$	

Lo que completa la demostración del teorema.

□

A continuación vamos a analizar el comportamiento de las computaciones del P sistema en las que nunca se ejecuta la regla $s \to s_I$.

Teorema 4.8 *Sea $\mathcal{C}$ una computación de Π_{TM} tal que la regla $s \to s_I$ no se dispara en ningún instante. Entonces para cada número natural $k \in \mathbf{N}$ existen objetos $a_{i_1}, \ldots, a_{i_k}, a_{i_{k+1}}, b_{i_1}, \ldots b_{i_k}, b_{i_{k+1}}$ tales que*

$$\begin{aligned} M_1^{3k} &= q_0 h^{2^{k+1}} b_0 s + B_k + A_k + S \\ M_1^{3k+1} &= q_0 h^{2^{k+1}} b_0 s_{a_{i_{k+1}}} + B_k + A_k + S \\ M_1^{3k+2} &= q_0 h^{2^{k+1}} b_0 s^+_{a_{i_{k+1}}} + B_k + A_k + S(a_{i_{k+1}}) \\ M_1^{3k+3} &= q_0 h^{2^{k+2}} b_0 s + B_{k+1} + A_{k+1} + S \end{aligned}$$

Demostración:
Por inducción débil sobre k. Para probar el caso base, $k = 0$, observemos que, teniendo presente que la regla $s \to s_I$ no se ejecuta, la configuración $\mathcal{C}^1$ se obtiene de $\mathcal{C}^0$ ejecutando las siguientes reglas para un cierto objeto a_{i_1}:

j	M_1^j	Multiconjunto de reglas aplicable
0	$q_0 h^2 b_0 s + S$	$s \to s_{a_{i_1}}$; $s^-_{a_j} \to s^-_{a_j}$ $(1 \le j \le p)$; $s^-_I \to s^-_I$ $s^-_i \to s^-_i$ $(1 \le i \le 9)$; $s^-_O \to s^-_O$
1	$q_0 h^2 b_0 s_{a_{i_1}} + S$	$s_{a_{i_1}} s^-_{a_{i_1}} \to s^+_{a_{i_1}}$; $s^-_{a_j} \to s^-_{a_j}$ $(1 \le j \le p \wedge j \ne i_1)$ $s^-_I \to s^-_I$; $s^-_i \to s^-_i$ $(1 \le i \le 9)$; $s^-_O \to s^-_O$
2	$q_0 h^2 b_0 s^+_{a_{i_1}} + S(a_{i_1})$	$h \to h^2 a_{i_1} b_{i_1}$; $s^+_{a_{i_1}} \to s^-_{a_{i_1}} s$; $s^-_I \to s^-_I$; $s^-_O \to s^-_O$ $s^-_i \to s^-_i$ $(1 \le i \le 9)$ $s^-_{a_j} \to s^-_{a_j}$ $(1 \le j \le p \wedge j \ne i_1)$
3	$q_0 h^{2^2} b_0 s + B_1 + A_1 + S$	

Lo que prueba el resultado para el caso $k = 0$.
Sea $k \in \mathbf{N}$ y supongamos que el resultado es cierto para k. Entonces existen objetos $a_{i_1}, \ldots, a_{i_k}, a_{i_{k+1}}, b_{i_1}, \ldots b_{i_k}, b_{i_{k+1}}$ tales que

$$M_1^{3k} = q_0 h^{2^{k+1}} b_0 s + B_k + A_k + S$$

$$M_1^{3k+1} = q_0 h^{2^{k+1}} b_0 s_{a_{i_{k+1}}} + B_k + A_k + S$$
$$M_1^{3k+2} = q_0 h^{2^{k+1}} b_0 s^+_{a_{i_{k+1}}} + B_k + A_k + S(a_{i_{k+1}})$$
$$M_1^{3k+3} = q_0 h^{2^{k+2}} b_0 s + B_{k+1} + A_{k+1} + S$$

Teniendo presente que la regla $s \to s_I$ no se ejecuta nunca en esta computación, resulta que la configuración $\mathcal{C}^{3k+4}$ se obtiene de $\mathcal{C}^{3k+3}$ ejecutando las siguientes reglas (para un cierto objeto $a_{i_{k+2}}$):

j	M_1^j	Multiconjunto de reglas aplicable
$3k+3$	$q_0 h^{2^{k+2}} b_0 s + B_{k+1} + A_{k+1} + S$	$s \to s_{a_{i_{k+2}}}; s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_i^- \to s_i^-$ $(1 \le i \le 9)$
$3k+4$	$q_0 h^{2^{k+2}} b_0 s_{a_{i_{k+2}}} + B_{k+1} + A_{k+1} + S$	$s_{a_{i_{k+2}}} s^-_{a_{i_{k+2}}} \to s^+_{a_{i_{k+2}}}$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \le i \le 9); s_O^- \to s_O^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p \wedge j \ne i_{k+2})$
$3k+5$	$q_0 h^{2^{k+2}} b_0 s^+_{a_{i_{k+2}}} + B_{k+1} + A_{k+1} + S(a_{i_{k+2}})$	$h \to h^2 a_{i_{k+2}} b_{i_{k+2}}; s_I^- \to s_I^-$ $s_O^- \to s_O^-$; $s^+_{a_{i_{k+2}}} \to s^-_{a_{i_{k+2}}} s$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p \wedge j \ne i_{k+2})$ $s_i^- \to s_i^-$ $(1 \le i \le 9)$
$3k+6$	$q_0 h^{2^{k+3}} b_0 s + B_{k+2} + A_{k+2} + S$	

Lo que completa la demostración del teorema.

□

Nota: Es interesante observar nuevamente que en esta etapa de generación del dato de entrada de la máquina de Turing, un *punto crítico* de las computaciones del P sistema se produce en el instante en que se dispara la regla $s \to s_I$.

4.2 Etapa de simulación de la función de transición

En esta sección se trata de demostrar que el P sistema Π_{TM}, tras la fase de generación de un dato de entrada arbitrario de la máquina de Turing TM, actúa

de forma determinista y *simula* un paso de computación de TM con el dato de entrada $w = a_{i_1} \dots a_{i_k} \in \Sigma^*_{TM}$ generado en la fase no determinista. Para ello distinguiremos varios casos, según la cabeza de trabajo esté analizando la primera casilla u otra distinta, según el elemento de la casilla analizada sea o no el símbolo blanco, o según el elemento que se vaya a escribir en la correspondiente casilla sea o no el símbolo blanco.

Nota: A lo largo de esta sección notaremos $A_k = a_{i_1}^2 \dots a_{i_k}^{2^k}$, $B_l = b_{t_1}^2 \dots b_{t_l}^{2^k}$, $B'_l = b'^2_{t_1} \dots b'^{2^l}_{t_l}$, $B_l(j) = B_l - \{b_{t_j}^{2^j}\}$ y $B'_l(j) = B'_l - \{b'^{2^j}_{t_j}\}$ a fin de simplificar un poco el desarrollo de las pruebas.

Caso 1: $\delta(q_r, \triangleright) = (q_{r'}, \triangleright, 1)$.

En este caso se trata de simular un paso de transición de la computación de la máquina de Turing TM con dato de entrada $w = a_{i_1} \dots a_{i_k} \in \Sigma^*_{TM}$, suponiendo que en un cierto instante el estado de la máquina es q_r, el contenido de la cinta es $\triangleright a_{t_1} \dots a_{t_l} BBB \dots$, la cabeza analiza la primera casilla y la máquina evoluciona de acuerdo con lo especificado por la función de transición, que en el caso que nos ocupa consiste en cambiar al estado $q_{r'}$, no escribir nada y desplazarse una casilla hacia la derecha.

Sea $\mathcal{C}$ una computación del P sistema Π_{TM} tal que, en el instante d, codifica la situación antes descrita de la máquina de Turing tras haber finalizado la fase de generación del dato de entrada w. Es decir, tal que

$$M_1^d = q_r b_0 s_1 + B_l + A_k + S$$

De acuerdo con el valor de la función de transición en este caso, resulta que $Q(r,0) = r', A(r,0) = 0$ y $D(r,0) = 1$. Por tanto, hemos de probar que, de forma determinista, a partir de la configuración $\mathcal{C}^d$ se llega a una cierta configuración $\mathcal{C}^{d'}$ caracterizada por la condición siguiente:

$$M_1^{d'} = q_{r'} h s_8 b_0 + B_l + A_k + S$$

recordando que la aparición del símbolo s_8 en la membrana nos indica la finalización de la etapa de simulación correspondiente al paso de transición considerado en este caso.

Observemos que en la descripción de la configuración $\mathcal{C}^d$ puede suceder que algún elemento a_{t_j} sea el símbolo blanco B. En este caso convendremos que en nuestra representación el objeto b_{t_j} no aparezca; es decir, que $b_{t_j} = \lambda$.

Teorema 4.9 *Sea $\mathcal{C}$ una computación de Π_{TM}. Sea $d \geq 1$ tal que el contenido de M_1^d es $q_r b_0 s_1 + B_l + A_k + S$. Si $\delta(q_r, \triangleright) = (q_{r'}, \triangleright, 1)$, entonces se verifica que $M_1^{d+12} = q_{r'} h b_0 s_8 + B_l + A_k + S$.*

Demostración:
Se tiene que:

j	M_1^j	Multiconjunto de reglas aplicable
d	$q_r b_0 s_1 + B_l + A_k + S$	$s_1 s_1^- \to s_1^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 1)$
$d+1$	$q_r s_1^+ + B_l + A_k + S(1)$	$b_0 \to b_0 b_0';\ b_{t_j} \to b_{t_j} b_{t_j}' (1 \le j \le l)$ $s_1^+ \to s_1^- s_2;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 1)$
$d+2$	$q_r b_0 b_0' s_2 + B_l + B_l' + A_k + S$	$s_2 \to s_3;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-;\ s_i^- \to s_i^-\ (1 \le i \le 9); s_O^- \to s_O^-$
$d+3$	$q_r b_0 b_0' s_3 + B_l + B_l' + A_k + S$	$s_3 s_3^- \to s_3^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 3)$
$d+4$	$q_r b_0 b_0' s_3^+ + B_l + B_l' + A_k + S(3)$	$b_{t_j}'^2 \to \lambda\ (1 \le j \le l);\ s_3^+ \to s_3^- s_4$ $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p); s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 3); s_O^- \to s_O^-$
$d+5$	$q_r b_0 b_0' s_4 + B_l + A_k + S$	$s_4 s_4^- \to s_4^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 4)$
$d+6$	$q_r b_0 b_0' s_4^+ + B_l + A_k + S(4)$	$s_4^+ \to s_4^- s_5;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 4)$
$d+7$	$q_r b_0 b_0' s_5 + B_l + A_k + S$	$s_5 s_5^- \to s_5^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 5)$
$d+8$	$q_r b_0 b_0' s_5^+ + B_l + A_k + S(5)$	$q_r b_0' \to q_{r'} b_0' c_0 h;\ s_5^+ \to s_5^- s_6; s_I^- \to s_I^-$ $s_O^- \to s_O^-;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 5)$
$d+9$	$q_{r'} h b_0 b_0' c_0 s_6 + B_l + A_k + S$	$s_6 \to s_7;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9); s_O^- \to s_O^-$
$d+10$	$q_{r'} h b_0 b_0' c_0 s_7 + B_l + A_k + S$	$s_7 s_7^- \to s_7^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 7)$

j	M_1^j	Multiconjunto de reglas aplicable
$d+11$	$q_{r'}hb_0b_0'c_0s_7^+ + B_l + A_k + S(7)$	$b_0b_0' \to \lambda;\ c_0 \to b_0; s_7^+ \to s_7^- s_8$ $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p); s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 7); s_O^- \to s_O^-$
$d+12$	$q_{r'}hb_0s_8 + B_l + A_k + S$	

Lo que completa la demostración del teorema.

□

Caso 2: $\delta(q_r, B) = (q_{r'}, B, x)$, con $x \in \{-1, 0, 1\}$.

En este caso se trata de simular un paso de transición de la computación de la máquina de Turing TM con dato de entrada $w = a_{i_1} \dots a_{i_k} \in \Sigma_{TM}^*$, suponiendo que en un cierto instante el estado de la máquina es q_r, el contenido de la cinta es $\triangleright a_{t_1} \dots a_{t_j} \dots a_{t_l} BBB \dots$, con $a_{t_j} = B$, la cabeza analiza la casilla j y la máquina evoluciona de acuerdo con lo especificado por la función de transición.

Sea $\mathcal{C}$ una computación del P sistema Π_{TM} tal que, en el instante d, codifica la situación antes descrita de la máquina de Turing tras haber finalizado la fase de generación del dato de entrada w. Es decir, tal que

$$M_1^d = q_r h^j b_0 s_1 + B_l(j) + A_k + S$$

De acuerdo con el valor de la función de transición en este caso, resulta que $Q(r, B) = r'$; $A(r, B) = B$ y $D(r, B) = x$, con $x \in \{-1, 0, 1\}$. Por tanto, hemos de probar que, de forma determinista, a partir de la configuración $\mathcal{C}^d$ se llega a una cierta configuración $\mathcal{C}^{d'}$ caracterizada por la condición siguiente:

$$M_1^{d'} = q_{r'} h^{j+x} b_0 s_8 + B_l(j) + A_k + S$$

Esto es lo que expresa el siguiente teorema.

Teorema 4.10 *Sea $\mathcal{C}$ una computación de Π_{TM}. Sea $d \ge 1$ tal que*

$$M_1^d = q_r h^j b_0 s_1 + B_l(j) + A_k + S$$

Sea $\delta(q_r, B) = (q_{r'}, B, x)$, con $x \in \{-1, 0, 1\}$. Entonces se verifica que

$$M_1^{d+5j+12} = q_{r'} h^{j+x} b_0 s_8 + B_l(j) + A_k + S$$

Demostración:
Se tiene que:

j	M_1^j	Multiconjunto de reglas aplicable
d	$q_r h^j b_0 s_1 + B_l(j) + A_k + s_1 S$	$s_1 s_1^- \to s_1^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 1)$
$d+1$	$q_r h^j b_0 s_1^+ + B_l(j) + A_k + S(1)$	$h \to hh'; b_0 \to b_0 b_0'$ $b_{t_i} \to b_{t_i} b'_{t_i} (1 \le i \le l \wedge i \ne j)$ $s_1^+ \to s_1^- s_2; s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 1)$
$d+2$	$q_r h^j h'^j b_0 s_2 + B_l(j) + B_l'(j) + A_k + S$	

Aserto 1: *Para cada α tal que $1 \le \alpha \le j$ se verifica que $M_1^{d+2+3\alpha} = q_r h^j h'^{(j-\alpha)} b_0 s_2 + B_l(j) + B_l'(j,\alpha) + A_k + S$, donde $B_l'(j,\alpha) = \{b'_{t_\alpha} \ldots b'^{2^{l-\alpha}}_{t_l}\} - \{b'^{2^{j-\alpha}}_{t_j}\}$.*
Prueba del aserto 1: Por inducción sobre α. Para ello, comencemos probando el caso base $\alpha = 1$.

j	M_1^j	Multiconjunto de reglas aplicable
$d+2$	$q_r h^j h'^j b_0 s_2 + B_l(j) + B_l'(j) + A_k + S$	$h' s_2 s_2^- \to s_2^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2)$
$d+3$	$q_r h^j h'^{(j-1)} b_0 s_2^+ + B_l(j) + B_l'(j) + A_k + S(2)$	$b'^2_{t_i} \to b''_{t_i} (1 \le i \le l \wedge i \ne j)$ $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2)$
$d+4$	$q_r h^j h'^{(j-1)} b_0 s_2^+ + B_l(j) + B_l''(j,1) + A_k + S(2)$	$b_0' \to \lambda;\ b''_{t_i} \to b'_{t_i} (1 \le i \le l \wedge i \ne j)$ $s_2^+ \to s_2^- s_2; s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2)$
$d+5$	$q_r h^j h'^{(j-1)} b_0 s_2 + B_l(j) + B_l'(j,1) + A_k + S$	

Lo que prueba el resultado para el caso $\alpha = 1$.

Sea $\alpha \geq 1$ tal que $\alpha < j$ y supongamos cierto el resultado para α. Sea $k = d + 2 + 3\alpha$. Entonces

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_r h^j h'^{(j-\alpha)} s_2 b_0 + B_l(j) + B_l'(j,\alpha) + A_k + S$	$h' s_2 s_2^- \to s_2^+$(pues $j - \alpha > 0$) $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 2)$ $s_O^- \to s_O^-$
$k+1$	$q_r h^j h'^{(j-\alpha-1)} b_0 s_2^+ + B_l(j) + B_l'(j,\alpha) + A_k + S(2)$	$b_{t_i}'^2 \to b_{t_i}''(\alpha+1 \leq i \leq l)$; $s_I^- \to s_I^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 2)$ $s_O^- \to s_O^-$
$k+2$	$q_r h^j h'^{(j-\alpha-1)} b_0 b_{t_\alpha}' s_2^+ + B_l(j) + B_l''(j,\alpha+1) + A_k + S(2)$	$b_{t_\alpha}' \to \lambda$; $b_{t_i}'' \to b_{t_i}'(\alpha+1 \leq i \leq l)$ $s_2^+ \to s_2^- s_2$; $s_I^- \to s_I^-$; $s_O^- \to s_O^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9)$
$k+3$	$q_r h^j h'^{(j-\alpha-1)} b_0 s_2^+ + B_l(j) + B_l'(j,\alpha+1) + A_k + S$	

Lo que completa la prueba del aserto 1.

■

Aplicando el aserto 1 al caso $\alpha = j$ y haciendo $k = d + 2 + 3j$ se deduce que

j	M_1^j	Reglas aplicables
k	$q_r h^j b_0 s_2 + B_l(j) + B_l'(j,j) + A_k + S$	$s_2 \to s_3$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \leq i \leq 9)$ $s_O^- \to s_O^-$
$k+1$	$q_r h^j b_0 s_3 + B_l(j) + B_l'(j,j) + A_k + S$	$s_3 s_3^- \to s_3^+$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 3)$ $s_O^- \to s_O^-$
$k+2$	$q_r h^j b_0 s_3^+ + B_l(j) + B_l'(j,j) + A_k + S(3)$	$b_{t_i}'^2 \to \lambda$ $(j+1 \leq i \leq l)$ $s_3^+ \to s_3^- s_4$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \leq i \leq 9)$ $s_O^- \to s_O^-$
$k+3$	$q_r h^j b_0 s_4 + B_l(j) + A_k + S$	$s_4 s_4^- \to s_4^+$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 4)$ $s_O^- \to s_O^-$

j	M_1^j	Reglas aplicables
$k+4$	$q_r h^j b_0 s_4^+ + B_l(j) + A_k + S(4)$	$h \to hh'$; $s_4^+ \to s_4^- s_5$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 4)$; $s_O^- \to s_O^-$
$k+5$	$q_r h^j h'^j b_0 s_5 + B_l(j) + A_k + S$	$s_5 s_5^- \to s_5^+$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 5)$ $s_O^- \to s_O^-$
$k+6$	$q_r h^j h'^j b_0 s_5^+ + B_l(j) + S(5)$	(si $x = 1$ entonces $q_r \to q_{r'} h$) (si $x = 0$ entonces $q_r \to q_{r'}$) (si $x = -1$ entonces $q_r h \to q_{r'}$) $s_5^+ \to s_5^- s_6$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 5)$ $s_O^- \to s_O^-$
$k+7$	$q_{r'} h^{j+x} h'^j b_0 s_6 + B_l(j) + A_k + S$	

A continuación establecemos un nuevo aserto.

Aserto 2: *Para cada α tal que $1 \le \alpha \le j$ se verifica que el contenido de $M_1^{d+3j+9+2\alpha} = q_{r'} h^{j+x} h'^{(j-\alpha)} b_0 s_6 + B_l(j) + A_k + S$.*

Prueba del aserto 2: Por inducción sobre α. Para ello, comencemos probando el caso base $\alpha = 1$. Sea $k = d + 3j$.

j	M_1^j	Multiconjunto de reglas aplicable
$k+9$	$q_{r'} h^{j+x} h'^j b_0 s_6 + B_l(j) + A_k + S$	$h' s_6 s_6^- \to s_6^+$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 6)$ $s_O^- \to s_O^-$
$k+10$	$q_{r'} h^{j+x} h'^{(j-1)} b_0 s_6^+ + B_l(j) + A_k + S(6)$	$s_6^+ \to s_6^- s_6$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 6)$ $s_O^- \to s_O^-$
$k+11$	$q_{r'} h^{j+x} h'^{(j-1)} b_0 s_6 + B_l(j) + A_k + S$	

Lo que prueba el resultado para el caso $\alpha = 1$.

Sea $\alpha \ge 1$ tal que $\alpha < j$ y supongamos cierto el resultado para α. Hagamos $k = d + 3j + 9 + 2\alpha$. Entonces

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}h'^{(j-\alpha)}b_0s_6 +$ $B_l(j) + A_k + S$	$h's_6s_6^- \to s_6^+$ (pues $j-\alpha>0$) $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 6)$ $s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}h'^{(j-\alpha-1)}b_0s_6^+ +$ $B_l(j) + A_k + S(6)$	$s_6^+ \to s_6^-s_6$; $s_I^- \to s_I^-$; $s_O^- \to s_O^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 6)$
$k+2$	$q_{r'}h^{j+x}h'^{(j-\alpha-1)}b_0s_6$ + $B_l(j) + A_k + S$	

Lo que completa la demostración del aserto 2. ■

Aplicando el aserto 2 al caso $\alpha = j$ se deduce que

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}b_0s_6 +$ $B_l(j)+A_k+S$	$s_6 \to s_7$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9)$; $s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}b_0s_7 +$ $B_l(j)+A_k+S$	$s_7s_7^- \to s_7^+$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 7)$ $s_O^- \to s_O^-$
$k+2$	$q_{r'}h^{j+x}b_0s_7^+ +$ $B_l(j) + A_k +$ $S(7)$	$s_7^+ \to s_7^-s_8$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \le j \le p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \le i \le 9 \wedge i \ne 7)$; $s_O^- \to s_O^-$
$k+3$	$q_{r'}h^{j+x}b_0s_8 +$ $B_l(j)+A_k+S$	

siendo $k = d + 5j + 9$.
Lo que completa la demostración del teorema. □

Caso 3: $\delta(q_r, B) = (q_{r'}, a_z, x)$, con $a_z \ne B$ y $x \in \{-1, 0, 1\}$.

En este caso se trata de simular un paso de transición de la computación de la máquina de Turing TM con dato de entrada $w = a_{i_1} \dots a_{i_k} \in \Sigma_{TM}^*$, suponiendo que en un cierto instante el estado de la máquina es q_r, el contenido de la cinta es $\triangleright a_{t_1} \dots a_{t_j} \dots a_{t_l} BBB \dots$, con $a_{t_j} = B$, la cabeza analiza la casilla j y la máquina evoluciona de acuerdo con lo especificado por la función de transición.

Sea $\mathcal{C}$ una computación del P sistema Π_{TM} tal que, en el instante d, codifica la situación antes descrita de la máquina de Turing; es decir, tal que

$$M_1^d = q_r h^j b_0 s_1 + B_l(j) + A_k + S$$

De acuerdo con el valor de la función de transición en este caso, resulta que $Q(r,B) = r'; A(r,B) = z$ y $D(r,B) = x$. Por tanto, hemos de probar que, de forma determinista, a partir de la configuración $\mathcal{C}^d$ se llega a una configuración $\mathcal{C}^{d'}$ caracterizada por la condición siguiente:

$$M_1^{d'} = q_{r'} h^{j+x} b_0 b_z^{2^j} s_8 + B_l(j) + A_k + S$$

Esto es lo que expresa el siguiente teorema.

Teorema 4.11 *Sea $\mathcal{C}$ una computación de Π_{TM}. Sea $d \geq 1$ tal que*

$$M_1^d = q_r h^j b_0 s_1 + B_l(j) + A_k + S$$

en donde $j \geq 1$. Si $\delta(q_r, B) = (q_{r'}, a_z, x)$, con $a_z \neq B$ y $x \in \{-1, 0, 1\}$ entonces se verifica que $M_1^{d+5j+12} = q_{r'} h^{j+x} b_0 b_z^{2^j} s_8 + B_l(j) + A_k + S$.

Demostración:
Razonando como aparece en la prueba del teorema anterior (inmediatamente después del aserto 1), se obtiene que

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_r h^j h'^j b_0 s_5^+ +$ $B_l(j) + A_k + S(5)$	(si $x = 1$, $q_r \to q_{r'} c_z h$) (si $x = 0$, $q_r \to q_{r'} c_z$) (si $x = -1$, $q_r h \to q_{r'} c_z$) $s_5^+ \to s_5^- s_6$; $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_O^- \to s_O^-$ $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 5)$
$k+1$	$q_{r'} h^{j+x} h'^j b_0 c_z s_6 +$ $B_l(j) + A_k + S$	

donde $k = d + 3j + 8$.

A continuación establecemos el siguiente aserto.

Aserto 3: *Para cada α tal que $1 \leq \alpha \leq j$ se verifica que el contenido de $M_1^{d+3j+9+2\alpha} = q_{r'} h^{j+x} h'^{(j-\alpha)} b_0 c_z^{2^\alpha} s_6 + B_l(j) + A_k + S$.*

Prueba del aserto 3: Por inducción sobre α. Para ello, comencemos probando el caso base $\alpha = 1$. Sea $k = d + 3j + 9$.

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}h'^{j}b_0c_zs_6 +$ $B_l(j) + A_k + S$	$h's_6s_6^- \to s_6^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \neq 6)$ $s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}h'^{(j-1)}b_0c_zs_6^+ +$ $B_l(j) + A_k + S(6)$	$c_z \to c_z^2;\ s_6^+ \to s_6^-s_6$ $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_O^- \to s_O^-$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \neq 6)$
$k+2$	$q_{r'}h^{j+x}h'^{(j-1)}b_0c_z^{2^1} +$ $B_l(j) + A_k + S$	

Lo que prueba el resultado para el caso base $\alpha = 1$.

Sea $\alpha \ge 1$ tal que $\alpha < j$ y supongamos cierto el resultado para α. Sea $k = d + 3j + 9 + 2\alpha$. Entonces

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}h'^{(j-\alpha)}b_0c_z^{2^\alpha}s_6 +$ $B_l(j) + A_k + S$	$h's_6s_6^- \to s_6^+$(pues $j - \alpha > 0$) $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \neq 6)$ $s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}h'^{(j-\alpha-1)}b_0c_z^{2^\alpha}s_6^+ +$ $B_l(j) + A_k + S(6)$	$c_z \to c_z^2;\ s_6^+ \to s_6^-s_6$ $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \neq 6)$
$k+2$	$q_{r'}h^{j+x}h'^{(j-\alpha-1)}b_0c_z^{2^{\alpha+1}}s_6+$ $B_l(j) + A_k + S$	

Lo que completa la prueba del aserto.

■

Aplicando el aserto 3 al caso $\alpha = j$ se deduce que, tomando $k = d + 5j + 9$:

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}b_0c_z^{2^j}s_6 +$ $B_l(j) + A_k + S$	$s_6 \to s_7;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9); s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}b_0c_z^{2^j}s_7 +$ $B_l(j) + A_k + S$	$s_7s_7^- \to s_7^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \neq 7); s_O^- \to s_O^-$

j	M_1^j	Multiconjunto de reglas aplicable
$k+2$	$q_{r'}h^{j+x}b_0c_z^{2^j}s_7^+ +$ $B_l(j)+A_k+S(7)$	$c_z \to b_z;\ s_7^+ \to s_7^- s_8;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-;\ s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 7)$ $s_O^- \to s_O^-$
$k+3$	$q'_r h^{j+x}b_0 b_z^{2^j} s_8 +$ $B_l(j) + A_k + S$	

Lo que completa la demostración del teorema.

□

Caso 4: $\delta(q_r, a_{t_j}) = (q_{r'}, B, x)$, con $a_{t_j} \neq B$ y $x \in \{-1, 0, 1\}$.

En este caso se trata de simular un paso de transición de la computación de la máquina de Turing TM con dato de entrada $w = a_{i_1} \dots a_{i_k} \in \Sigma_{TM}^*$, suponiendo que en un cierto instante el estado de la máquina es q_r, el contenido de la cinta es $\triangleright a_{t_1} \dots a_{t_j} \dots a_{t_l} BBB \dots$, con $a_{t_j} \neq B$, la cabeza analiza la casilla j y la máquina evoluciona de acuerdo con lo especificado por la función de transición.

Sea $\mathcal{C}$ una computación del P sistema Π_{TM} tal que, en el instante d, codifica la situación antes descrita de la máquina de Turing; es decir, tal que

$$M_1^d = q_r h^j b_0 s_1 + B_l + A_k + S$$

De acuerdo con el valor de la función de transición en este caso, resulta que $Q(r, a_{t_j}) = r'; A(r, a_{t_j}) = B$ y $D(r, a_{t_j}) = x$. Por tanto, hemos de probar que, de forma determinista, a partir de la configuración $\mathcal{C}^d$ se llega a una configuración $\mathcal{C}^{d'}$ caracterizada por la condición siguiente:

$$M_1^{d'} = q_{r'} h^{j+x} b_0 s_8 + B_l(j) + A_k + S$$

Esto es lo que expresa el siguiente teorema.

Teorema 4.12 *Sea $\mathcal{C}$ una computación de Π_{TM}. Sea $d \geq 1$ tal que*

$$M_1^d = q_r h^j b_0 s_1 + B_l + A_k + S$$

en donde $j \geq 1$ y $a_{t_j} \neq B$. Si $\delta(q_r, a_{t_j}) = (q_{r'}, B, x)$, con $x \in \{-1, 0, 1\}$, entonces se verifica que $M_1^{d+5j+12} = q_{r'} h^{j+x} b_0 s_8 + B_l(j) + A_k + S$.

Demostración:
Se verifican las siguientes relaciones:

j	M_1^j	Multiconjunto de reglas aplicable
d	$q_r h^j b_0 s_1 + B_l + A_k + S$	$s_1 s_1^- \to s_1^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 1); s_O^- \to s_O^-$
$d+1$	$q_r h^j b_0 s_1^+ + B_l + A_k + S$	$h \to hh';\ b_0 \to b_0 b_0';\ b_{t_i} \to b_{t_i} b_{t_i}'\ (1 \le i \le l)$ $s_1^+ \to s_1^- s_2; s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 1); s_O^- \to s_O^-$
$d+2$	$q_r h^j h'^j b_0 b_0' s_2 + B_l + B_l' + A_k + S$	

A continuación establecemos el siguiente aserto:

Aserto 4: *Para cada α tal que $1 \le \alpha \le j$ se verifica que* $M_1^{d+2+3\alpha} = q_r h^j h'^{(j-\alpha)} b_0 s_2 + B_l + B_{l,\alpha}' + A_k + S$, *donde* $B_{l,j} = b_{t_j} b_{t_{j+1}}^2 \ldots b_{t_l}^{2^{l-j}}$.

Prueba del aserto 4: Por inducción sobre α. Para ello, comencemos probando el caso base $\alpha = 1$.

j	M_1^j	Multiconjunto de reglas aplicable
$d+2$	$q_r h^j h'^j b_0 b_0' s_2 + B_l + B_l' + A_k + S$	$h' s_2 s_2^- \to s_2^+$ (pues $j > 0$) $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2);\ s_O^- \to s_O^-$
$d+3$	$q_r h^j h'^{(j-1)} b_0 s_2^+ + B_l + B_l' + A_k + S(2)$	${b_{t_i}'}^2 \to b_{t_i}''\ (1 \le i \le l);\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2)$ $s_O^- \to s_O^-$
$d+4$	$q_r h^j h'^{(j-1)} b_0 b_0' s_2^+ + B_l + B_{l,1}'' + A_k + S(2)$	$b_0' \to \lambda;\ b_{t_i}'' \to b_{t_i}'\ (1 \le i \le l)$ $s_2^+ \to s_2^- s_2;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p); s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2); s_O^- \to s_O^-$
$d+5$	$q_r h^j h'^{(j-1)} b_0 s_2^+ + B_l + B_{l,1}' + A_k + S$	

Lo que prueba el resultado para el caso $\alpha = 1$.

Sea $\alpha \ge 1$ tal que $\alpha < j$ y supongamos cierto el resultado para α. Sea $k = d + 2 + 3\alpha$. Entonces

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_r h^j h'^{(j-\alpha)} b_0 s_2 + B_l + B_{l,\alpha}' + A_k + S$	$h' s_2 s_2^- \to s_2^+$(pues $j - \alpha > 0$) $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p); s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2);\ s_O^- \to s_O^-$
$k+1$	$q_r h^j h'^{(j-\alpha-1)} b_0 s_2^+ + B_l + B_{l,\alpha}' + A_k + S(2)$	${b_{t_i}'}^2 \to b_{t_i}''\ (\alpha + 1 \le i \le l)$ $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2); s_O^- \to s_O^-$

j	M_1^j	Multiconjunto de reglas aplicable
$k+2$	$q_r h^j h'^{(j-\alpha-1)} b_0 b'_{t_\alpha} s_2^+ +$ $B_l + B''_{l,\alpha+1} + A_k + S(2)$	$b'_{t_\alpha} \to \lambda;\ b''_{t_i} \to b'_{t_i}\ (\alpha+1 \le i \le l)$ $s_2^+ \to s_2^- s_2; s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 2)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$
$k+3$	$q_r h^j h'^{(j-\alpha-1)} b_0 s_2 +$ $B_l + B'_{l,\alpha+1} A_k + S$	

Lo que completa la demostración del aserto 4.

■

Aplicando el aserto 4 al caso $\alpha = j$ se deduce que, tomando $k = d + 2 + 3j$:

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_r h^j b_0 s_2 + B_l +$ $B'_{l,j} + A_k + S$	$s_2 \to s_3;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9);\ s_O^- \to s_O^-$
$k+1$	$q_r h^j b_0 s_3 + B_l +$ $B'_{l,j} + A_k + S$	$s_3 s_3^- \to s_3^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 3)$ $s_O^- \to s_O^-$
$k+2$	$q_r h^j b_0 s_3^+ + B_l +$ $B'_{l,j} + A_k + S$	$b'^{\,2}_{t_i} \to \lambda\ (j+1 \le i \le l);\ s_3^+ \to s_3^- s_4$ $s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p); s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 3); s_O^- \to s_O^-$
$k+3$	$q_r h^j b_0 b'_{t_j} s_4 + B_l +$ $A_k + S$	$s_4 s_4^- \to s_4^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 4)$ $s_O^- \to s_O^-$
$k+4$	$q_r h^j b_0 b'_{t_j} s_4^+ + B_l +$ $A_k + S(4)$	$h \to hh';\ s_4^+ \to s_4^- s_5;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 4)$ $s_O^- \to s_O^-$
$k+5$	$q_r h^j h'_j b_0 b'_{t_j} s_5 + B_l +$ $A_k + S$	$s_5 s_5^- \to s_5^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 5)$ $s_O^- \to s_O^-$
$k+6$	$q_r h^j h'^j b_0 b'_{t_j} s_5^+ +$ $B_l + A_k + S(5)$	(si $x = 1$ entonces $q_r b'_{t_j} \to q_{r'} b'_{t_j} h$) (si $x = 0$ entonces $q_r b'_{t_j} \to q_{r'} b'_{t_j}$) (si $x = -1$ entonces $q_r b'_{t_j} h \to q_{r'} b'_{t_j}$) $s_5^+ \to s_5^- s_6;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 5)$

j	M_1^j	Multiconjunto de reglas aplicable
$k+7$	$q_{r'}h^{j+x}h'^{j}b_0b'_{t_j}s_6 + B_l + A_k + S$	

A continuación establecemos un nuevo aserto.

Aserto 5: *Para cada α tal que $1 \leq \alpha \leq j$ se verifica que*

$$M_1^{d+3j+9+2\alpha} = q_{r'}h^{j+x}h'^{(j-\alpha)}b_0b'^{2^\alpha}_{t_j}s_6 + B_l + A_k + S.$$

Prueba del aserto 5: Por inducción sobre α. Para ello, comencemos probando el caso base $\alpha = 1$. Sea $k = d + 3j + 9$.

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}h'^{j}b_0b'_{t_j}s_6 + B_l + A_k + S$	$h's_6s_6^- \to s_6^+; s_I^- \to s_I^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 6); s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}h'^{(j-1)}b_0b'_{t_j}s_6^+ + B_l + A_k + S(6)$	$b'_{t_j} \to b'^2_{t_j}$; $s_6^+ \to s_6^-s_6$; $s_I^- \to s_I^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_O^- \to s_O^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 6)$
$k+2$	$q_{r'}h^{j+x}h'^{(j-1)}b_0b'^2_{t_j}s_6 + B_l + A_k + S$	

Lo que prueba el resultado para el caso base $\alpha = 1$.

Sea $\alpha \geq 1$ tal que $\alpha < j$ y supongamos cierto el resultado para α. Sea ahora $k = d + 3j + 9 + 2\alpha$. Entonces:

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}h'^{(j-\alpha)}b_0b'^{2^\alpha}_{t_j}s_6 + B_l + A_k + S$	$h's_6s_6^- \to s_6^+$ (pues $j - \alpha > 0$) $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 6)$ $s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}h'^{(j-\alpha-1)}b_0b'^{2^\alpha}_{t_j}s_6^+ + B_l + A_k + S(6)$	$b'_{t_j} \to b'^2_{t_j}$; $s_6^+ \to s_6^-s_6$; $s_I^- \to s_I^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_O^- \to s_O^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 6)$
$k+2$	$q_{r'}h^{j+x}h'^{(j-\alpha-1)}b_0b'^{2^{\alpha+1}}_{t_j}s_6 + B_l + A_k + S$	

Lo que completa la prueba del aserto.

■

Aplicando el aserto anterior al caso $\alpha = j$, se obtiene que, tomando $k = d + 5j + 9$

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}b_0b_{t_j}'^{2^j}s_6 +$ $B_l + A_k + S$	$s_6 \to s_7;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9); s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}b_0b_{t_j}'^{2^j}s_7 +$ $B_l + A_k + S$	$s_7 s_7^- \to s_7^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 7); s_O^- \to s_O^-$
$k+2$	$q_{r'}h^{j+x}b_0b_{t_j}'^{2^j}s_7^+ +$ $B_l + A_k + S(7)$	$b_{t_j}b_{t_j}' \to \lambda;\ s_7^+ \to s_7^- s_8;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 7); s_O^- \to s_O^-$
$k+3$	$q_{r'}h^{j+x}b_0s_8 +$ $B_l(j) + A_k + S$	

Lo que completa la demostración del teorema.

□

Caso 5: $\delta(q_r, a_{t_j}) = (q_{r'}, a_z, x)$, con $a_{t_j} \ne B$, $a_z \ne B$ y $x \in \{-1, 0, 1\}$.

Razonando de manera análoga a como se ha hecho en los casos anteriores, se trata de probar el siguiente resultado:

Teorema 4.13 *Sea $\mathcal{C}$ una computación de Π_{TM}. Sea $d \ge 1$ tal que*

$$M_1^d = q_r h^j b_0 s_1 + B_l + A_k + S$$

en donde $j \ge 1$ y $a_{t_j} \ne B$. Si $\delta(q_r, a_{t_j}) = (q_{r'}, a_z, x)$, con $a_z \ne B$ y $x \in \{-1, 0, 1\}$, entonces se verifica que $M_1^{d+5j+12} = q_{r'}h^{j+x}b_0b_z^{2^j}s_8 + B_l(j) + A_k + S$.

Demostración:
Razonando como se ha hecho en la prueba del teorema anterior, se tiene que, tomando $k = d + 3j + 8$:

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_r h^j h'^j b_0 b_{t_j}' s_5^+ +$ $B_l + A_k + S(5)$	(si $x = 1$ entonces $q_r b_{t_j}' \to q_{r'} b_{t_j}' c_z h$) (si $x = 0$ entonces $q_r b_{t_j}' \to q_{r'} b_{t_j}' c_z$) (si $x = -1$ entonces $q_r b_{t_j}' h \to q_{r'} b_{t_j}' c_z$) $s_5^+ \to s_5^- s_6;\ s_{a_j}^- \to s_{a_j}^-\ (1 \le j \le p)$ $s_I^- \to s_I^-;\ s_i^- \to s_i^-\ (1 \le i \le 9 \wedge i \ne 5)$ $s_O^- \to s_O^-$
$k+1$	$q_{r'}h^{j+x}h'^j b_0 b_{t_j}' c_z +$ $B_l + A_k + S$	

A continuación establecemos el siguiente aserto.

Aserto 6: *Para cada* α *tal que* $1 \leq \alpha \leq j$ *se verifica que*

$$M_1^{d+3j+9+2\alpha} = q_{r'} h^{j+x} h'^{(j-\alpha)} b_0 b_{t_j}'^{2^\alpha} c_z^{2^\alpha} s_6 + B_l + A_k + S$$

Prueba del aserto 6: Por inducción sobre α. Para ello, comencemos probando el caso base $\alpha = 1$. Sea $k = d + 3j + 9$, entonces,

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'} h^{j+x} h'^{j} b_0 b'_{t_j} c_z s_6 + B_l + A_k + S$	$h' s_6 s_6^- \to s_6^+$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$ $s_I^- \to s_I^-; s_O^- \to s_O^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 6)$
$k+1$	$q_{r'} h^{j+x} h'^{(j-1)} b_0 b'_{t_j} c_z s_6^+ + B_l + A_k + S(6)$	$b'_{t_j} \to b'^2_{t_j}$; $c_z \to c_z^2$; $s_6^+ \to s_6^- s_6$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 6)$ $s_O^- \to s_O^-$
$k+2$	$q_{r'} h^{j+x} h'^{(j-1)} b_0 b'_0 b'^2_{t_j} c_z^{2^1} s_6 + B_l + A_k + S$	

Lo que prueba el resultado para el caso base $\alpha = 1$.

Sea $\alpha \geq 1$ tal que $\alpha < j$ y supongamos cierto el resultado para α. Sea $k = d + 3j + 9 + 2\alpha$, entonces:

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'} h^{j+x} h'^{(j-\alpha)} b_0 b_{t_j}'^{2^\alpha} c_z^{2^\alpha} + B_l + A_k + S$	$h' s_6 s_6^- \to s_6^+$ (pues $j - \alpha > 0$) $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 6)$ $s_O^- \to s_O^-$
$k+1$	$q_{r'} h^{j+x} h'^{(j-\alpha-1)} b_0 b_{t_j}'^{2^\alpha} c_z^{2^\alpha} s_6^+ + B_l + A_k + S(6)$	$b'_{t_j} \to b'^2_{t_j}$; $c_z \to c_z^2$; $s_6^+ \to s_6^- s_6$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 6)$ $s_O^- \to s_O^-$
$k+2$	$q_{r'} h^{j+x} h'^{(j-\alpha-1)} b_0 b_{t_j}'^{2^{\alpha+1}} c_z^{2^{\alpha+1}} s_6 + B_l + A_k + S$	

Lo que completa la prueba del aserto.

■

Aplicando el aserto 6 al caso $\alpha = j$ resulta que, tomando $k = d + 5j + 9$:

j	M_1^j	Multiconjunto de reglas aplicable
k	$q_{r'}h^{j+x}b_0b_{t_j}'^{2^j}c_z^{2^j}s_6 + B_l + A_k + S$	$s_6 \to s_7;\ s_{\bar{a}_j} \to s_{\bar{a}_j}\ (1 \le j \le p)$ $s_{\bar{I}} \to s_{\bar{I}}; s_{\bar{O}} \to s_{\bar{O}}$ $s_{\bar{i}} \to s_{\bar{i}}\ (1 \le i \le 9)$
$k+1$	$q_{r'}h^{j+x}b_0b_{t_j}'^{2^j}c_z^{2^j}s_7 + B_l + A_k + S$	$s_7 s_7^- \to s_7^+; s_{\bar{I}} \to s_{\bar{I}}; s_{\bar{O}} \to s_{\bar{O}}$ $s_{\bar{a}_j} \to s_{\bar{a}_j}\ (1 \le j \le p)$ $s_{\bar{i}} \to s_{\bar{i}}\ (1 \le i \le 9 \wedge i \ne 7)$
$k+2$	$q_{r'}h^{j+x}b_0b_{t_j}'^{2^j}c_z^{2^j}s_7^+ + B_l + A_k + S(7)$	$b_{t_j}b_{t_j}' \to \lambda;\ c_z \to b_z;\ s_7^+ \to s_7^- s_8$ $s_{\bar{a}_j} \to s_{\bar{a}_j}\ (1 \le j \le p); s_{\bar{I}} \to s_{\bar{I}}$ $s_{\bar{i}} \to s_{\bar{i}}\ (1 \le i \le 9 \wedge i \ne 7); s_{\bar{O}} \to s_{\bar{O}}$
$k+3$	$q_{r'}h^{j+x}b_0b_z^{2^j}s_8 + B_l(j) + A_k + S$	

Lo que completa la demostración del teorema.

□

4.3 Etapa de chequeo del estado final

En la simulación de la máquina de Turing determinista, TM, a través del P sistema con salida externa, Π_{TM}, la aparición del objeto s_8 en la membrana del sistema nos indica que ha terminado la simulación de un paso de transición. Entonces hemos de proceder a comprobar si tras la ejecución de dicho paso se ha llegado o no a una configuración de aceptación. Veamos seguidamente cómo se puede simular este proceso en el P sistema Π_{TM}. Para ello, si se llega a una tal configuración (caracterizada por el estado q_Y), la membrana sólo contendrá simbolos del alfabeto de entrada de la máquina; si se alcanza el estado q_N, entonces la computación entrará en un bucle infinito; y, si no se alcanza ni el estado q_Y ni el estado q_N, entonces aparecerá nuevamente en la membrana el objeto s_1, lo que nos indicará que ha de volver a realizarse un nuevo paso de transición.

Teorema 4.14 *Sea $\mathcal{C}$ una computación de Π_{TM}. Sea $d \ge 1$ tal que*

$$M_1^d = q_r h^j b_0 s_8 + B_l + A_k + S$$

en donde $j \ge 0$. Se verifica:

1. *Si $q_r = q_Y$, entonces $M_1^{d+4} = A_k = \{\{a_{i_1}^2 \dots a_{i_k}^{2^k}\}\}$.*

2. *Si* $q_r = q_N$, *entonces para cada* $\alpha \geq 1$ *se tiene que*
$$M_1^{d+2+\alpha} = q_N h^j b_0 + B_l + A_k + S$$

3. *Si* $q_r \in Q - \{q_Y, q_N\}$, *entonces* $M_1^{d+2} = q_r h^j b_0 s_1 + B_l + A_k + S$.

Demostración:
Se tiene la siguiente relación

j	M_1^j	Multiconjunto de reglas aplicable
d	$q_r h^j b_0 s_8 + B_l + A_k + S$	$s_8 s_8^- \to s_8^+;\ s_{a_j}^- \to s_{a_j}^-\ (1 \leq j \leq p)$ $s_I^- \to s_I^-;\ s_i^- \to s_i^-\ (1 \leq i \leq 9 \wedge i \neq 8)$ $s_O^- \to s_O^-$
$d+1$	$q_r h^j b_0 s_8^+ + B_l + A_k + S(8)$	

Distingamos tres casos.

Caso 1: Supongamos que $q_r = q_Y$.

j	M_1^j	Multiconjunto de reglas aplicable
$d+1$	$q_r h^j b_0 s_8^+ + B_l + A_k + S(8)$	$q_Y \to q_Y s_9;\ s_8^+ \to s_8^-$ $s_{a_j}^- \to s_{a_j}^-\ (1 \leq j \leq p); s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \leq i \leq 9 \wedge i \neq 8); s_O^- \to s_O^-$
$d+2$	$q_Y h^j b_0 s_9 + B_l + A_k + S$	$s_9 s_9^- \to \lambda;\ s_{a_j}^- \to s_{a_j}^-\ (1 \leq j \leq p)$ $s_I^- \to s_I^-; s_i^- \to s_i^-\ (1 \leq i \leq 9 \wedge i \neq 9)$ $s_O^- \to s_O^-$
$d+3$	$q_Y h^j b_0 + B_l + A_k + S(9)$	$s_{a_1}^- \ldots s_{a_p}^- s_I^- s_1^- \ldots s_8^- s_0^- \to \lambda;\ q_Y \to \lambda$ $h \to \lambda$ (si $j \geq 1$); $b_0 \to \lambda$ $b_{t_i} \to \lambda\ (1 \leq i \leq p)$
$d+4$	A_k	

Caso 2: Supongamos que $q_r = q_N$.

j	M_1^j	Multiconjunto de reglas aplicable
$d+1$	$q_r h^j b_0 s_8^+ + B_l + A_k + S(8)$	$q_N \to q_N;\ s_8^+ \to s_8^-$ $s_{a_j}^- \to s_{a_j}^-\ (1 \leq j \leq p \wedge j);\ s_I^- \to s_I^-$ $s_i^- \to s_i^-\ (1 \leq i \leq 9 \wedge i \neq 8); s_O^- \to s_O^-$
$d+2$	$q_N h^j b_0 + B_l + A_k + S$	

En tal situación, para cada $\alpha \geq 1$ se tendrá que la configuración $\mathcal{C}^{d+2+\alpha}$ se obtiene de la configuración $\mathcal{C}^{d+2+\alpha-1}$ ejecutando las reglas

$s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_I^- \to s_I^-$; $s_i^- \to s_i^-$ $(1 \leq i \leq 9)$; $s_O^- \to s_O^-$

Además se tendrá que $M_1^{d+2+\alpha} = M_1^{d+2}$.

Caso 3: Supongamos que $q_r \in Q - \{q_Y, q_N\}$.

j	M_1^j	Multiconjunto de reglas aplicable
$d+1$	$q_r h^j b_0 s_8^+ + B_l + A_k + S(8)$	$q_r \to q_r s_1$; $s_8^+ \to s_8^-$ $s_{a_j}^- \to s_{a_j}^-$ $(1 \leq j \leq p)$; $s_I^- \to s_I^-$ $s_i^- \to s_i^-$ $(1 \leq i \leq 9 \wedge i \neq 8)$; $s_O^- \to s_O^-$
$d+2$	$q_r h^j b_0 s_1 + B_l + A_k + S$	

□

4.4 Etapa de producción del dato de salida

En la simulación de la máquina de Turing determinista, TM, a través del P sistema con salida externa, Π_{TM}, la aparición del objeto s_9 en la membrana del sistema nos indica que se ha llegado a una configuración de aceptación. En ese caso, hemos de devolver al entorno la cadena de entrada $w = a_{i_1} \ldots a_{i_k} \in \Sigma_{TM}^*$ que había sido generada en la fase no determinista, a fin de justificar que dicha cadena es *aceptada* por el P sistema Π_{TM}. El instante en que por primera vez el símbolo s_O^- *no* aparece en la membrana, nos indicará que ha llegado el momento de producir la salida de la computación. Teniendo presente que en la semántica de los P sistemas con salida externa, el orden de salida al entorno es tenida en cuenta, representaremos el contenido del entorno como un multiconjunto ordenado, qué indique cuál es la sucesión de objetos que se han ido expulsando del sistema.

Teorema 4.15 *Sea $\mathcal{C}$ una computación de Π_{TM}. Sea $d \geq 1$ tal que $M_1^d = a_{i_1}^2 \ldots a_{i_k}^{2^k}$ y el entorno está vacío.*
Se verifica:

1. *Si $k = 1$, entonces $M_1^{d+3} = \emptyset$ y $M_{env}^{d+3} = (a_{i_1})$.*

2. *Si* $k \geq 2$, *entonces* $M_1^{d+2k+1} = \emptyset$ *y* $M_{env}^{d+2k+1} = (a_{i_1}, \ldots, a_{i_k})$.

Demostración:

1. Supongamos que $k = 1$.

j	M_1^j	Multiconjunto de reglas aplicable
d	$a_{i_1}^2$	$a_{i_1}^2 \to d_{i_1}$
$d+1$	d_{i_1}	$d_{i_1} \to a_{i_1}$
$d+2$	a_{i_1}	$a_{i_1} \to (a_{i_1}, out)$
$d+3$	$\emptyset$	

   Y $M_{env}^{d+3} = (a_{i_1})$.

2. Supongamos que $k \geq 2$. En primer lugar establecemos el siguiente aserto:

   Aserto 7: *Para cada* α *tal que* $2 \leq \alpha \leq k$ *se verifica que* $M_1^{d+2\alpha} = a_{i_\alpha} a_{i_{\alpha+1}}^2 \ldots a_{i_k}^{2^{k-\alpha}}$ *y* $M_{env}^{d+2\alpha} = (a_{i_1}, \ldots, a_{i_{\alpha-1}})$.

   Prueba del aserto 7: Por inducción sobre α. Para ello, comencemos probando el caso base $\alpha = 2$.

j	M_1^j	Multiconjunto de reglas aplicable
d	$a_{i_1}^2 \ldots a_{i_k}^{2^k}$	$a_{i_j}^2 \to d_{i_j}$ $(1 \leq j \leq k)$
$d+1$	$d_{i_1} d_{i_2}^2 \ldots d_{i_k}^{2^{k-1}}$	$d_{i_j} \to a_{i_j}$ $(1 \leq j \leq k)$
$d+2$	$a_{i_1} a_{i_2}^2 \ldots a_{i_k}^{2^{k-1}}$	$a_{i_j}^2 \to d_{i_j}$ $(2 \leq j \leq k)$
$d+3$	$a_{i_1} d_{i_2} d_{i_3}^2 \ldots d_{i_k}^{2^{k-2}}$	$a_{i_1} \to (a_{i_1}, out)$; $d_{i_j} \to a_{i_j}$ $(2 \leq j \leq k)$
$d+4$	$a_{i_2} a_{i_3}^2 \ldots a_{i_k}^{2^{k-2}}$	

   Y $M_{env}^{d+4} = (a_{i_1})$. Lo que prueba el resultado para el caso $\alpha = 1$.

   Sea $\alpha \geq 2$ tal que $\alpha < k$ y supongamos cierto el resultado para α. Es decir, supongamos por hipótesis de inducción que $M_1^{d+2\alpha} = a_{i_\alpha} a_{i_{\alpha+1}}^2 \ldots a_{i_k}^{2^{k-\alpha}}$ y $M_{env}^{d+2\alpha} = (a_{i_1}, \ldots, a_{i_{\alpha-1}})$.

   La configuración $\mathcal{C}^{d+2\alpha+1}$ se obtiene de $\mathcal{C}^{d+2\alpha}$ ejecutando las reglas $a_{i_j}^2 \to d_{i_j}$ $(\alpha+1 \leq j \leq k)$. Luego $M_1^{d+2\alpha+1} = a_{i_\alpha} d_{i_{\alpha+1}} \ldots d_{i_k}^{2^{k-\alpha-1}}$ y $M_{env}^{d+2\alpha+1} = M_{env}^{d+2\alpha}$.

La configuración $\mathcal{C}^{d+2\alpha+2}$ se obtiene de $\mathcal{C}^{d+2\alpha+1}$ ejecutando las reglas $a_{i_\alpha} \to (a_{i_\alpha}, out)$; $d_{i_j} \to a_{i_j}$ $(\alpha+1 \leq j \leq k)$. Luego $M_1^{d+2\alpha+2} = a_{i_{\alpha+1}} \ldots a_{i_k}^{2^{k-\alpha-1}}$ y $M_{env}^{d+2\alpha+1} = (a_{i_1}, \ldots, a_{i_{\alpha-1}} a_{i_\alpha})$.

Lo que completa la prueba del aserto.

■

Aplicando el aserto anterior al caso $\alpha = k$, se deduce que $M_1^{d+2k} = a_{i_k}$ y $M_{env}^{d+2k} = (a_{i_1}, \ldots, a_{i_{k-1}})$.

La configuración $\mathcal{C}^{d+2k+1}$ se obtiene de $\mathcal{C}^{d+2k}$ ejecutando la regla $a_{i_k} \to (a_{i_k}, out)$. Luego $M_1^{d+2k+1} = \emptyset$ y $M_{env}^{d+2k+1} = (a_{i_1}, \ldots, a_{i_{k-1}}, a_{i_k})$.

□

4.5 Análisis de la simulación

En esta sección se trata de probar que la simulación de una máquina de Turing determinista por un P sistema con salida externa (y, por tanto, por un P sistema básico de transición) tiene un coste adicional de tipo polinomial.

Teorema 4.16 *Toda maquina de Turing determinista puede ser simulada por una P sistema con salida externa con un coste adicional de tipo polinomial.*

Demostración:
En primer lugar, hallemos el tiempo de ejecución de la simulación.

Sea $w \in \Sigma_{TM}^*$ un dato de entrada de la máquina de Turing, de tamaño k. Supongamos que $w \in L_{TM}$. Entonces se tiene que:

1. La generación del dato de entrada, w, en el P sistema Π_{TM} requiere $3k+3$ pasos.

2. La simulación de cada paso de transición requiere un coste del orden $O(5j+12)$, siendo j la casilla que está analizando la cabeza lectora.

3. El chequeo del estado final requiere 4 pasos.

4. La producción de una salida de tamaño k requiere $2k+1$ pasos.

Por tanto, si el tiempo que la máquina TM necesita para aceptar el dato w es del orden $O(t(k))$, entonces el P sistema Π_{TM} necesitará $O(5k + 5t(k) + 20) \subseteqq O(k + t(k))$ pasos.

Finalmente, hallemos el coste que se necesita para la construcción del P sistema Π_{TM} a partir de la máquina de Turing TM.

Sea TM una máquina de Turing determinista tal que el conjunto de estados tiene cardinal $n + 2$, el alfabeto de trabajo tiene cardinal $m + 2$ y el alfabeto de entrada tiene cardinal p (con $p \leq m$). Analicemos los recursos que son necesarios para construir Π_{TM}.

1. El grado del P sistema es 1.
2. El numero de símbolos utilizados es $n + 4m + 5p + 39 \in \theta(n + m)$.
3. El número total de reglas usados es $n + 10m + 9p + 42 + O(mn) \subseteqq O(mn)$.
4. La lóngitud máxima de una regla es $p + 10 \in O(m)$.
5. El número total de relaciones de prioridad es $n + 11m + 2m^2 + 2p^2 + 4p + 32 + O(nm) \subseteqq O(m^2 + mn)$.

En consecuencia, la construcción del P sistema Π_{TM} a partir de la máquina TM tiene un coste adicional polinomial, del orden $O(m^2 + mn)$ siendo $\theta(n)$ el orden de los estados y $\theta(m)$ el orden del cardinal del alfabeto de trabajo.

□

Capítulo 5

Clases de Complejidad en P sistemas de aceptación

Uno de los objetivos principales de la *Teoría de la Complejidad* consiste en proporcionar herramientas que permitan la clasificación de problemas de acuerdo con la cantidad de recursos que son necesarios para su resolución.
En este capítulo se presentan las primeras *clases de complejidad* en P sistemas, que permiten detectar algunas de las dificultades inherentes a la resolución computacional de ciertos problemas y proporcionan una clasificación de los *problemas abstractos* en función de los recursos que necesitan para ser resueltos en el modelo de Computación Celular con Membranas. Desde luego, dicha clasificación exige una definición precisa y rigurosa del concepto de *problema abstracto* así como del modelo que se va a considerar.

Toda clase de complejidad suele estar caracterizada por una serie de parámetros:

- El *modelo* de computación (en nuestro caso, los P sistemas de aceptación).
- El *modo* de computación (en nuestro caso, determinista y paralelo).
- La *medida* de complejidad (el tiempo y el espacio).
- La *cota* superior de recursos (que será una función total computable de $\mathbf{N}^+$ en $\mathbf{N}^+$).

Comenzaremos definiendo lo que entendemos por *problema de decisión*, y por P *sistema de aceptación*. Posteriormente, introduciremos el concepto de *resolubilidad* de un problema de decisión a través de una familia de P sistemas de aceptación.

5.1 Clases de complejidad

Definición 5.1 *Un problema de decisión, DP, es un par (I_{DP}, a_{DP}) que verifica las condiciones siguientes:*

- *I_{DP} es un lenguaje sobre un alfabeto finito (cuyos elementos se denominan instancias o ejemplares del problema).*
- *a_{DP} es una función total booleana sobre I_{DP}.*

Nota: Teniendo presente que todo alfabeto finito puede ser codificado a partir del alfabeto $\{0,1\}$ mediante una función computable de coste *lineal*, resulta que toda instancia de cualquier problema de decisión puede considerarse como una cadena del alfabeto $\{0,1\}$.

Definición 5.2 *Un P sistema de aceptación es un par ordenado (Π, θ_Π) tal que Π es un P sistema y θ_Π es una función total booleana sobre el conjunto de todas las computaciones de parada de Π.*

Sea $\mathcal{C}$ una computación de parada de Π. Si $\theta_\Pi(\mathcal{C}) = 1$, entonces diremos que $\mathcal{C}$ es de aceptación; en caso contrario, diremos que $\mathcal{C}$ es de rechazo.

Usualmente la función θ_Π viene definida implícitamente por la semántica del modelo de computación celular con membranas Π.

Definición 5.3 *Sean X una clase de P sistemas de aceptación sin membrana de entrada, f una función total computable de $\mathbf{N}^+$ en $\mathbf{N}^+$ y DP un problema de decisión. Diremos que el problema DP pertenece a la clase de complejidad $\mathbf{MC_X}(f)$ si existe una familia, $\mathbf{\Pi^X_{DP}} = (\Pi^X_{DP}(w))_{w \in I_{DP}}$, de P sistemas tal que:*

1. *La familia $\mathbf{\Pi^X_{DP}}$ es X–consistente; es decir, para cada $w \in I_{DP}$, el P sistema $\Pi^X_{DP}(w)$ pertenece a la clase X.*
2. *La familia $\mathbf{\Pi^X_{DP}}$ es DP–uniforme; es decir, existe una máquina de Turing que, a partir de $w \in I_{DP}$ construye el P sistema $\Pi^X_{DP}(w)$ en tiempo*

polinomial. Con otras palabras, existen una máquina de Turing, TM', y un polinomio, $p(n)$, que dependen del problema DP, tal que para cada $w \in I_{DP}$, $TM'(w)$ para en, a lo sumo, $p(|w|)$ pasos y devuelve el P sistema $\Pi^X_{DP}(w)$.

3. *La familia $\mathbf{\Pi^X_{DP}}$ está f–acotada; es decir, para cada $w \in I_{DP}$ toda computación de $\Pi^X_{DP}(w)$ para en $O(f(|w|))$ pasos.*

4. *La familia $\mathbf{\Pi^X_{DP}}$ es DP–adecuada; es decir, para cada $w \in I_{DP}$ se tiene que $a_{DP}(w) = 1$ si y sólo si toda computación de $\Pi^X_{DP}(w)$ es de aceptación.*

En la situación de la definición anterior, diremos que el problema de decisión DP es *resoluble por la familia $\mathbf{\Pi^X_{DP}}$ de P sistemas de aceptación sin entrada*, en tiempo $O(f)$.
Obsérvese que de acuerdo con la definición anterior, el P sistema $\Pi^X_{DP}(w)$ es el encargado de analizar la aceptación o rechazo de la instancia w del problema DP.
A continuación, definimos la resolubilidad en tiempo lineal y polinomial a través de familias de P sistemas.

Definición 5.4 *Se definen las clases de complejidad lineal y polinomial, respectivamente, para P sistemas de aceptación sin membrana de entrada como sigue:*

$$\mathbf{LMC_X} = \bigcup_{f \text{ lineal}} \mathbf{MC_X}(f) \quad y \quad \mathbf{PMC_X} = \bigcup_{f \text{ polinomial}} \mathbf{MC_X}(f)$$

Seguidamente vamos a introducir conceptos análogos para P sistemas de aceptación con entrada.

Definición 5.5 *Sean X una clase de P sistemas de aceptación con membrana de entrada, f una función total computable de $\mathbf{N}^+$ en $\mathbf{N}^+$ y DP un problema de decisión. Diremos que el problema DP pertenece a la clase de complejidad $\mathbf{MC^*_X}(f)$ si existe una familia, $\mathbf{\Pi^X_{DP}} = (\Pi^X_{DP}(n))_{n \in \mathbf{N}^+}$, de P sistemas tal que:*

1. *La familia $\mathbf{\Pi^X_{DP}}$ es X–consistente; es decir, para cada $n \in \mathbf{N}^+$, el P sistema $\Pi^X_{DP}(n)$ pertenece a la clase X.*

2. *La familia* $\mathbf{\Pi^X_{DP}}$ *es DP–uniforme; es decir, existe una máquina de Turing que, a partir de* $n \in \mathbf{N}^+$ *construye el P sistema* $\Pi^X_{DP}(n)$ *en tiempo polinomial. Con otras palabras, existe una máquina de Turing,* TM'*, y un polinomio,* $p(n)$*, que depende del problema DP, tal que para cada* $n \in \mathbf{N}^+$*,* $TM'(n)$ *para en, a lo sumo,* $p(n)$ *pasos y devuelve el P sistema* $\Pi^X_{DP}(n)$*.*

3. *La familia* $\mathbf{\Pi^X_{DP}}$ *está* f*–acotada; es decir, para cada* $n \in \mathbf{N}^+$ *toda computación de* $\Pi^X_{DP}(n)$ *para en* $O(f(n))$ *pasos.*

4. *La familia* $\mathbf{\Pi^X_{DP}}$ *es DP–adecuada; es decir, existe una codificación polinomial,* g*, de las instancias del problema DP en objetos de entrada de los P sistemas (multiconjuntos, cadenas, etc.), de tal manera que para cada* $w \in I_{DP}$ *se tiene que* $a_{DP}(w) = 1$ *si y sólo si toda computación de* $\Pi^X_{DP}(|w|)$ *con entrada* $g(w)$ *es de aceptación.*

   Nota: *Que* g *sea una codificación polinomial de las instancias del problema DP en objetos de entrada de los P sistemas* $\Pi^X_{DP}(n)$*, significa que existe una máquina de Turing,* TM''*, y un polinomio* $q(n)$ *que depende del problema DP, tal que para cada* $w \in I_{DP}$*,* $TM''(w)$ *para en, a lo sumo,* $q(|w|)$ *pasos y, además, devuelve* $g(w)$*, que es un objeto de entrada del P sistema* $\Pi^X_{DP}(|w|)$*.*

En la situación de la definición anterior, diremos que el problema de decisión DP es *resoluble por la familia* $\mathbf{\Pi^X_{DP}}$ *de P sistemas de aceptación con entrada*, en tiempo $O(f)$.
Obsérvese que de acuerdo con la definición anterior, para cada $n \in \mathbf{N}^+$ el P sistema $\Pi^X_{DP}(n)$ es el encargado de analizar la aceptación o rechazo de todas las instancias del problema de tamaño n.
De forma análoga a como se ha hecho anteriormente, definimos la resolubilidad en tiempo lineal y polinomial a través de familias de P sistemas.

Definición 5.6 *Se definen las clases de complejidad lineal y polinomial, respectivamente, para P sistemas de aceptación con membrana de entrada como sigue:*

$$\mathbf{LMC^*_X} = \bigcup_{f \text{ lineal}} \mathbf{MC^*_X}(f) \quad y \quad \mathbf{PMC^*_X} = \bigcup_{f \text{ polinomial}} \mathbf{MC^*_X}(f)$$

5.2 Clase de complejidad del problema SAT

A continuación se presenta un resultado que pone de manifiesto la potencia computacional de ciertas variantes de P sistemas. Para ello, introducimos una variante de P sistemas que hace uso de *membranas activas*, y demostramos que el problema **SAT** pertenece a la clase de complejidad lineal de dichos P sistemas.
Siguiendo [39], un *P sistema con membranas activas* es una tupla

$$\Pi = (V, H, \mu, w_1, \ldots, w_m, R)$$

en donde:

1. $m \geq 1$.
2. V es un alfabeto.
3. H es un conjunto finito de etiquetas para las membranas. Se reserva la etiqueta 0 para la membrana piel.
4. μ es una estructura de membranas, de grado m, etiquetadas con elementos de H. Las membranas de μ pueden ser neutras o estar polarizadas (positiva o negativamente).
5. $w_1, \ldots, w_m$ son multiconjuntos sobre V, que expresan el contenido inicial de las membranas de μ.
6. R es un conjunto finito de *reglas de evolución* de los siguientes tipos:
   (a) $[_i x \to y]_i^\alpha$, donde $x \in V$, $y \in V^*$, $\alpha \in \{0, +, -\}$.
   Regla interna, que no afecta al exterior de la membrana i, ni a su polaridad.
   (b) $[_i x]_i^\alpha \to y[_i]_i^\beta$, donde $x, y \in V$, $\alpha, \beta \in \{0, +, -\}$.
   Un elemento puede salir de la membrana, posiblemente transformado en otro y, simultáneamente, modificar la polaridad de dicha membrana.
   (c) $x[_i]_i^\alpha \to [_i y]_i^\beta$, donde $x, y \in V$, $\alpha, \beta \in \{0, +, -\}$.
   Un elemento puede entrar en la membrana, posiblemente transformado en otro y, simultáneamente, modificar la polaridad de dicha membrana.

(d) $[_i x]_i^\alpha \to y$, donde $x, y \in V$, $\alpha \in \{0, +, -\}$, $i \neq 0$.
Un elemento es transformado en otro y, simultáneamente, la membrana i es disuelta (la piel no se puede disolver).

(e) $[_i x]_i^\alpha \to [_i y]_i^\beta [_i z]_i^\gamma$, donde $x, y, z \in V$, $\alpha, \beta, \gamma \in \{0, +, -\}$, $i \neq 0$.
División de la membrana i, pudiendo eventualmente transformarse el elemento independientemente para cada membrana resultante. No se puede ejecutar en la piel.

Las reglas se aplican de acuerdo con los siguientes principios:

- Todas las reglas son aplicadas en paralelo y de forma maximal. En un paso, un elemento de una membrana sólo puede ser usado por una operación (elegida de manera no determinista en caso de que haya varias posibilidades).
- Si una membrana es disuelta, su contenido (multiconjunto y membranas interiores a la misma) pasa a formar parte de la membrana inmediatamente exterior a ella.
- Todas los elementos no especificados por la operación a aplicar permanecen invariables.

Notemos por **MA** la clase de P sistemas de aceptación deterministas que hacen uso de membranas activas (es decir, que contienen reglas del tipo (e)). Se verifica el siguiente resultado:

Teorema 5.7 $\mathbf{SAT} \in \mathbf{LMC}^*_{\mathbf{MA}}$

Demostración:
Para demostrar este resultado, construiremos una familia de P sistemas de aceptación que hacen uso de membranas activas, de forma que, fijado el número de cláusulas y de variables, el P sistema diseñado, que tiene adecuadamente codificada la fórmula en forma normal conjuntiva, devuelva Yes en caso de que dicha fórmula sea satisfactible.
Supondremos que $\varphi = C_1 \wedge \ldots \wedge C_m$ es una fórmula en forma normal conjuntiva sobre el conjunto de variables $X = \{x_1, \ldots, x_n\}$, donde $C_i \subseteq X \cup \overline{X}$.
El P sistema diseñado para las fórmulas de tamaño (n, m) es

$$\Pi^{\mathbf{MA}}_{\mathbf{SAT}}([n, m]) = (V, \{1, 2\}, [_1\ [_2\ ]_2\ ]_1, w_1, w_2, R)$$

donde $[n,m]$ es una codificación del par (n,m) (por ejemplo la *codificación par* $[n,m] = \frac{(n+m)\cdot(n+m+1)}{2} + n$), y:

- El alfabeto es

$$\begin{aligned} V = \quad & \{x_{i,j}, \bar{x}_{i,j} : \ 1 \leq i \leq m, \ 1 \leq j \leq n\} \cup \\ & \{d_k : \ 1 \leq k \leq n\} \cup \\ & \{c_k : \ 1 \leq k \leq m+1\} \cup \\ & \{r_{i,k} : \ 1 \leq i \leq m, 1 \leq k \leq 2n\} \cup \\ & \{Yes\} \end{aligned}$$

- El contenido inicial de las membranas, que codifica la fórmula φ, es:

$$\begin{aligned} w_1 = \quad & \lambda \\ w_2 = \quad & \{x_{i,j} : \ x_j \in C_i\} \cup \{\bar{x}_{i,j} : \ \bar{x}_j \in C_i\} \cup \{d_1\} \end{aligned}$$

- El conjunto de reglas, R, viene dado por:

$\{[_2 d_k]_2^0 \to [_2 d_k]_2^+ [_2 d_k]_2^- : \ 1 \leq k \leq n\}$

$\{[_2 x_{i,1} \to r_{i,1}]_2^+, \ [_2 \bar{x}_{i,1} \to r_{i,1}]_2^- : \ 1 \leq i \leq m\}$

$\{[_2 x_{i,1} \to \lambda]_2^-, \ [_2 \bar{x}_{i,1} \to \lambda]_2^+ : \ 1 \leq i \leq m\}$

$\{[_2 x_{i,1} \to x_{i,j-1}]_2^+, \ [_2 x_{i,1} \to x_{i,j-1}]_2^- : \ 1 \leq i \leq m, \ 2 \leq j \leq n\}$

$\{[_2 \bar{x}_{i,1} \to \bar{x}_{i,j-1}]_2^+, \ [_2 \bar{x}_{i,1} \to \bar{x}_{i,j-1}]_2^- : \ 1 \leq i \leq m, \ 2 \leq j \leq n\}$

$\{[_2 d_k]_2^+ \to [_2 \ ]_2^0 d_k, \ d_k [_2 \ ]_2^0 \to [_2 d_{k+1}]_2^0, \ [_2 d_k]_2^- \to [_2 \ ]_2^0 d_k : \ 1 \leq k \leq n\}$

$\{[_2 r_{i,k} \to r_{i,k+1}]_2^0 : \ 1 \leq i \leq m, \ 1 \leq k \leq 2(n-1)\}$

$\{[_1 d_k \to d_{k+1}]_1^0 : \ n \leq k \leq 2n-1\}$

$d_{2n} [_2 \ ]_2^0 \to [_2 c_1]_2^+$

$[_2 r_{1,2n-1}]_2^+ \to [_2 \ ]_2^- r_{1,2n-1}$

$\{[_2 c_k \to c_{k+1}]_2^- : \ 1 \leq k \leq m\}$

$\{[_2 r_{i,2n-1} \to r_{i-1,2n-1}]_2^- : \ 1 \leq i \leq m\}$

$r_{1,2n-1} [_2 \ ]_2^- \to [_2 r_{0,2n-1}]_2^+$

$[_2 c_{m+1}]_2^+ \to [_2 \ ]_2^+ Yes$

$[_1 Yes]_1^0 \to [_1 \ ]_1^0 Yes$

Esencialmente, el proceso que sigue el P sistema anterior para resolver la satisfactibilidad de la fórmula es crear una cantidad exponencial de membranas en las que la carga de las mismas determina en cada paso una posible asignación de la variable asociada a dicho paso, decidiéndose qué cláusulas resultan satisfactibles por la nueva asignación.
Obsérvese que tanto la estructura de membranas del P sistema diseñado, como el conjunto de reglas, dependen únicamente del tamaño de la fórmula, por lo que podría ser considerado como un P sistema con entrada en la membrana 2, al que se proporciona adecuadamente codificada la fórmula proposicional (por medio de una codificación lineal en el tamaño de dicha fórmula).
Es fácil comprobar que, en un número de pasos lineal en el tamaño de φ, el P sistema presentado devuelve Yes en caso de que dicha fórmula sea satisfactible.
□

Capítulo 6

La conjetura $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ a través de los P sistemas de decisión

El objetivo de este capítulo es caracterizar la conjetura $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ mediante la irresolubilidad en tiempo polinomial de problemas **NP**–completos a través de familias de P sistemas de decisión. La caracterización que se presenta está basada en dos resultados: (a) toda máquina de Turing determinista puede ser simulada en tiempo polinomial por una familia de P sistemas deterministas de decisión; y (b) si un problema de decisión es resoluble en tiempo polinomial por una familia de P sistemas deterministas de decisión, entonces cada P sistema de la familia se puede simular, en tiempo polinomial, a través de una máquina de Turing determinista que, a su vez, puede ser construida en tiempo polinomial.

6.1 Simulación de máquinas de Turing a través de P sistemas

Definición 6.1 *Sea TM una máquina de Turing (como dispositivo de aceptación de lenguajes). El problema de decisión asociado a TM es el siguiente: $DP_{TM} = (\Sigma^*_{TM}, a_{TM})$; en donde $a_{TM}(w) = 1$ si y sólo si la máquina de Turing TM acepta el dato w.*

Definición 6.2 *Diremos que una máquina de Turing, TM, es simulada en*

tiempo $O(f)$ (con $f : \mathbf{N}^+ \to \mathbf{N}^+$ función total computable) en una clase, X, de P sistemas de aceptación sin membrana de entrada (resp. con membrana de entrada), si el problema de decisión asociado, DP_{TM}, pertenece a la clase $MC_X(f)$(resp. $MC_X^*(f)$).

Dicho con otras palabras, una máquina de Turing, TM, es simulada en tiempo $O(f)$ (con $f : \mathbf{N}^+ \to \mathbf{N}^+$ función total computable) en una clase, X, de P sistemas de aceptación (con o sin membrana de entrada) si existe una familia $\mathbf{\Pi_{DP}^X}$ (del tipo $(\Pi_{DP}^X(w))_{w \in I_{DP}}$ en el caso sin entrada y del tipo $(\Pi_{DP}^X(n))_{n \in \mathbf{N}^+}$ en el caso con entrada) tal que dicha familia es X–consistente, DP_{TM}–uniforme, está f–acotada y es DP_{TM}–adecuada.

Teorema 6.3 *Toda máquina de Turing determinista puede ser simulada en tiempo polinomial por una familia de P sistemas deterministas de decisión válidos. Es decir, toda máquina de Turing determinista puede ser simulada en tiempo polinomial en la clase de los P sistemas de decisión deterministas.*

Demostración:
Sea TM una máquina de Turing determinista. Sea $\Gamma_{TM} = \{B, \triangleright, a_1, \ldots, a_m\}$ el alfabeto de trabajo, y $\Sigma_{TM} = \{a_1, \ldots, a_p\}$, con $p \leq m$, el alfabeto de entrada. Supongamos que el conjunto de estados de la máquina es $Q_{TM} = \{q_N, q_Y, q_0, \ldots, q_n\}$. Sea $\delta_{TM}(q_i, a_j) = (q_{Q(i,j)}, a_{A(i,j)}, D(i,j))$ la función de transición de TM. Notaremos $a_B = B$ y $a_0 = \triangleright$.
A continuación describimos la familia de P sistemas deterministas de decisión $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \ldots, \Pi_{TM}(k), \ldots)$ que va a simular a la máquina TM: para cada $k \in \mathbf{N}$, el P sistema de decisión $\Pi_{TM}(k)$ está caracterizado por los siguientes elementos:

- Alfabeto de entrada: $\Sigma_k = \{\langle a, i\rangle : a \in \Sigma_{TM}, 1 \leq i \leq k\}$
- Alfabeto de trabajo: $\Gamma_k = \{\langle a, i\rangle : a \in \Sigma_{TM}, 0 \leq i \leq k\} \cup \{t_i : 1 \leq i \leq k\} \cup$
  $\{s_i^-, s_i^+, s_i : i \in \{T_1, T_2, F, S, 1, \ldots, 9\}\} \cup$
  $\{q_N, q_Y, h, h', Yes, No\} \cup$
  $\{q_i : 0 \leq i \leq n\} \cup \{b_i, b_i', b_i'', c_i : 0 \leq i \leq m\}$
- Estructura de membranas: $\mu_\Pi = [_1\,]_1$
- Membrana de entrada: $i_\Pi = 1$
- Multiconjunto inicial: $\mathcal{M}_1 = \{\{q_0, b_0, s_{T_1}^-, s_{T_2}^-, s_F^-, s_S^-, s_1^-, \ldots, s_9^-, s_{T_1}\}\}$
- Reglas de evolución: $R = R_0 \cup R_1 \cup R_2 \cup R_3 \cup R_4$, en donde:

$\bullet R_0 = R_{0,1} \cup R_{0,2} \cup R_{0,3} \cup R_{0,4}$, con

$$R_{0,1} \equiv s_{T_1} s_{T_1}^- \to s_{T_1}^+ > s_{T_1}^- \to s_{T_1}^- > \left\{ \begin{array}{l} \langle a_i, j \rangle \to \langle a_i, j \rangle t_j \quad (1 \leq i \leq p, 1 \leq j \leq k) \\ s_{T_1}^+ \to s_{T_1}^- s_{T_2} \end{array} \right.$$

$$R_{0,2} \equiv \left\{ \begin{array}{l} s_{T_2} s_{T_2}^- \to s_{T_2}^+ > s_{T_2}^- \to s_{T_2}^- > t_1^2 s_{T_2}^+ \to s_{T_2}^- s_F > \cdots > t_k^2 s_{T_2}^+ \to s_{T_2}^- s_F \\ > t_1 \ldots t_k s_{T_2}^+ \to s_{T_2}^- s_S > s_{T_2}^+ \to s_{T_2}^- s_F \end{array} \right.$$

$$R_{0,3} \equiv s_F s_F^- \to s_F^+ > s_F^- \to s_F^- > \left\{ \begin{array}{l} s_{T_1}^- s_{T_2}^- s_S^- s_1^- \ldots s_9^- s_F^+ q_0 b_0 \to (No, out) \\ \langle a_i, j \rangle \to \lambda \quad (1 \leq i \leq p, 1 \leq j \leq k) \\ t_j \to \lambda \quad (1 \leq j \leq k) \end{array} \right.$$

$$R_{0,4} \equiv \left\{ \begin{array}{l} s_S s_S^- \to s_S^+ > s_S^- \to s_S^- > \left\{ \begin{array}{l} \langle a_i, j \rangle \to \langle a_i, j-1 \rangle^2 \quad (1 \leq i \leq p, 1 \leq j \leq k) \\ \langle a_i, 0 \rangle \to b_i \quad (1 \leq i \leq p) \end{array} \right\} \\ > s_S^+ \to s_S^- s_1 \end{array} \right.$$

$\bullet R_1 = R_{1,1} \cup R_{1,2} \cup R_{1,3}$, con

$$R_{1,1} \equiv s_1 s_1^- \to s_1^+ > s_1^- \to s_1^- > \left\{ \begin{array}{l} h \to h h' \\ b_i \to b_i b_i' \quad (0 \leq i \leq m) \\ s_1^+ \to s_1^- s_2 \end{array} \right.$$

$$R_{1,2} \equiv \left\{ \begin{array}{l} h' s_2 s_2^- \to s_2^+ > s_2 \to s_3 > s_2^- \to s_2^- > \\ \qquad > b_i'^2 \to b_i'' \quad (0 \leq i \leq m) > \left\{ \begin{array}{l} b_i' \to \lambda \quad (0 \leq i \leq m) \\ b_i'' \to b_i' \quad (0 \leq i \leq m) \\ s_2^+ \to s_2^- s_2 \end{array} \right. \end{array} \right.$$

$$R_{1,3} \equiv s_3 s_3^- \to s_3^+ > s_3^- \to s_3^- > \left\{ \begin{array}{l} b_i'^2 \to \lambda \quad (0 \leq i \leq m) \\ s_3^+ \to s_3^- s_4 \end{array} \right.$$

$\bullet R_2 = R_{2,1} \cup R_{2,2}$, con

$$R_{2,1} \equiv s_4 s_4^- \to s_4^+ > s_4^- \to s_4^- > \left\{ \begin{array}{l} h \to h h' \\ s_4^+ \to s_4^- s_5 \end{array} \right.$$

$$R_{2,2} \equiv s_5 s_5^- \to s_5^+ > s_5^- \to s_5^- > \left\{ \begin{array}{l} \text{Reglas para la función} \\ \text{de transición} \\ s_5^+ \to s_5^- s_6 \end{array} \right.$$

Las reglas para la función de transición, δ_{TM}, son las siguientes:

Caso 1: Estado q_r, elemento $a_s \neq B$

Movimiento	*Reglas*	
izquierda	$q_r b'_s h \to q_{Q(r,s)} b'_s c_{A(r,s)},$	si $A(r,s) \neq B$
	$q_r b'_s h \to q_{Q(r,s)} b'_s,$	si $A(r,s) = B$
igual	$q_r b'_s \to q_{Q(r,s)} b'_s c_{A(r,s)},$	si $A(r,s) \neq B$
	$q_r b'_s \to q_{Q(r,s)} b'_s,$	si $A(r,s) = B$
derecha	$q_r b'_s \to q_{Q(r,s)} b'_s c_{A(r,s)} h,$	si $A(r,s) \neq B$
	$q_r b'_s \to q_{Q(r,s)} b'_s h,$	si $A(r,s) = B$

Caso 2: Estado q_r, sin elemento

Movimiento	*Reglas*	
izquierda	$q_r h \to q_{Q(r,s)} c_{A(r,s)},$	si $A(r,s) \neq B$
	$q_r h \to q_{Q(r,s)},$	si $A(r,s) = B$
igual	$q_r \to q_{Q(r,s)} c_{A(r,s)},$	si $A(r,s) \neq B$
	$q_r \to q_{Q(r,s)},$	si $A(r,s) = B$
derecha	$q_r \to q_{Q(r,s)} c_{A(r,s)} h,$	si $A(r,s) \neq B$
	$q_r \to q_{Q(r,s)} h,$	si $A(r,s) = B$

Para evitar conflictos, cada regla del caso 1 tiene una prioridad mayor que cada regla del caso 2.

- $R_3 = R_{3,1} \cup R_{3,2}$, con

$$R_{3,1} \equiv h' s_6 s_6^- \to s_6^+ > s_6 \to s_7 > s_6^- \to s_6^- > \begin{cases} b'_i \to b'^2_i & (0 \leq i \leq m) \\ c_i \to c_i^2 & (0 \leq i \leq m) \\ s_6^+ \to s_6^- s_6 \end{cases}$$

$$R_{3,2} \equiv s_7 s_7^- \to s_7^+ > s_7^- \to s_7^- > \begin{cases} b_i b'_i \to \lambda & (0 \leq i \leq m) \\ c_i \to b_i & (0 \leq i \leq m) \\ s_7^+ \to s_7^- s_8 \end{cases}$$

- $R_4 = R_{4,1} \cup R_{4,2}$, con

$$R_{4,1} \equiv s_8 s_8^- \to s_8^+ > s_8^- \to s_8^- > \begin{cases} q_Y \to q_Y s_9, \quad q_N \to q_N s_9 \\ q_i \to q_i s_1 \quad (0 \leq i \leq n) \\ s_8^+ \to s_8^- \end{cases}$$

$$R_{4,2} \equiv \quad s_9 s_9^- \to \lambda > s_9^- \to s_9^- > \begin{cases} s_{T_1}^- s_{T_2}^- s_F^- s_S^- s_1^- \ldots s_8^- \to \lambda \\ q_Y \to (Yes, out) \\ q_N \to (No, out) \\ h \to \lambda \\ b_i \to \lambda \quad (0 \leq i \leq m) \end{cases}$$

Seguidamente veamos que $\mathbf{\Pi_{TM}} = (\Pi_{TM}(1), \Pi_{TM}(2), \ldots, \Pi_{TM}(k), \ldots)$ es una familia de P sistemas deterministas de decisión que simula a la máquina TM.

Obviamente se trata de una familia de P sistemas deterministas válidos. Más aún, $\mathbf{\Pi_{TM}}$ es una *familia uniforme*. En efecto: supongamos que el número de estados de la máquina TM es $n+2$, el alfabeto de trabajo tiene tamaño $m+2$ y el alfabeto de entrada tiene tamaño p (con $p \leq m$). Entonces los recursos necesarios para construir el P sistema $\Pi_{TM}(k)$ son los siguientes:

1. El tamaño del alfabeto de trabajo Γ_k es $p{\cdot}k{+}k{+}4m{+}n{+}45 \in \theta(k{\cdot}(m{+}n))$.
2. El grado del P sistema es 1.
3. El tamaño de la configuración inicial para cada $x \in \Sigma_{TM}^k$ es $k{+}16 \in \theta(k)$.
4. El número total de reglas es del orden $O(p \cdot k + n \cdot m) = O(k \cdot (m+n))$.
5. La mayor de las longitudes de una regla es $16 \in O(1)$.

Veamos ahora que, para cada $k \in \mathbb{N}$, el P sistema $\Pi_{TM}(k)$ es *adecuado*. En efecto: sea L el lenguaje decidido por TM. Para decidir si una cadena $a_{i_1} \ldots a_{i_k} \in \Sigma_{TM}$, de tamaño k, pertenece a L, codificamos dicha cadena por el multiconjunto $\{\{\langle a_{i_1}, 1\rangle, \ldots, \langle a_{i_k}, k\rangle\}\}$ que va a ser la entrada del P sistema $\Pi_{TM}(k)$. Las reglas de este P sistema han sido escogidas adecuadamente de tal manera que la simulación de la computación de TM con dato de entrada la cadena citada procede en una serie de etapas:

1. Comprobar que el multiconjunto recibido codifica una cadena de longitud k. Para ello debemos verificar que para cada $j = 1, \ldots, k$, exista un par, y sólo uno, con segunda componente j. En caso de que esto no sea así, el P sistema para y devuelve No.
2. Transformar el multiconjunto de entrada en otro multiconjunto que codifica la cadena de entrada en base 2 (a fin de especificar el símbolo que está situado en cada una de las celdas de la cinta).

3. Leer el elemento de la celda que está siendo analizada por la cabeza de trabajo.

4. Hallar el nuevo elemento que hay que escribir en la celda que está siendo analizada, mover la cabeza de trabajo y cambiar al nuevo estado (todo ello de acuerdo con la función de transición).

5. Borrar el viejo elemento y escribir el nuevo elemento en la celda analizada.

6. Chequear si se ha alcanzado un estado final: si no, repetir el proceso desde la etapa 3; si se alcanza el estado q_Y, entonces aceptar la cadena; si se alcanza el estado q_N, entonces rechazar la cadena.

Finalmente veamos que, para cada $k \in \mathbb{N}$, el P sistema $\Pi_{TM}(k)$ está *polinomialmente acotado*. En efecto, si $x \in \Sigma_{TM}^k$ es una cadena de entrada de la máquina de Turing de tamaño k, se tiene (razonando como en la sección 4.5 del capítulo 4):

1. La comprobación de que el multiconjunto recibido codifique correctamente una cadena de longitud k necesita 4 pasos en caso afirmativo y 6 en caso negativo (en este último caso el P sistema ya para).

2. La generación del multiconjunto que codifica, en base 2, las celdas no vacías en la configuración inicial de TM para x, necesita $k+3$ pasos del P sistema.

3. La simulación de cada paso de transición de la máquina de Turing necesita un coste del orden $O(5j+12)$, en donde j es la celda que se analiza por la cabeza de trabajo de la máquina.

4. Comprobar si se ha alcanzado un estado final necesita 2 pasos.

5. La producción del dato de salida tras haber detectado la presencia de un estado final, necesita 2 pasos.

En consecuencia, si la máquina de Turing TM acepta o rechaza el dato de entrada x en $O(t(k))$ pasos, entonces el P sistema $\Pi_{TM}(k)$ necesita $O(k+t(k)^2)$ pasos para realizar lo mismo.

□

Sea $\mathcal{D}$ la clase de los P sistemas deterministas de decisión válidos. Notemos **NPC** a la clase de los problemas **NP**–completos. Seguidamente vamos a establecer una condición suficiente para que se verifique la relación $\mathbf{P} \neq \mathbf{NP}$ en términos de irresolubilidad de problemas **NP**–completos en la clase de P sistemas citados.

Lema 6.4 *Si existe un problema* **NP***-completo que es irresoluble en tiempo polinomial, respecto del tamaño del dato de entrada, por una familia de P sistemas deterministas de decisión, entonces* $\mathbf{P} \neq \mathbf{NP}$. *Es decir,*

$$(\exists Y\ (Y \in \mathbf{NPC}\ \wedge\ Y \notin \mathbf{PMC}^{*}_{\mathcal{D}})) \longrightarrow \mathbf{P} \neq \mathbf{NP}$$

Demostración:
Supongamos que $\mathbf{P} = \mathbf{NP}$. Sea Y un problema **NP**–completo arbitrario. Entonces $Y \in \mathbf{P}$. Luego existe una máquina de Turing determinista, TM_Y, que resuelve el problema Y en tiempo polinomial.

Entonces, del teorema 6.3 se deduce la existencia de una familia de P sistemas deterministas de decisión, $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{TM_Y}} = (\Pi^{\mathcal{D}}_{TM_Y}(1), \ldots, \Pi^{\mathcal{D}}_{TM_Y}(k), \ldots)$, que simula la máquina de Turing TM_Y en tiempo polinomial.
Veamos que la familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{TM_Y}}$ resuelve el problema Y en tiempo polinomial; es decir, que $Y \in \mathbf{PMC}^{*}_{\mathcal{D}}$. En efecto:

- Para cada $k \in \mathbf{N}^{+}$ se tiene que $\Pi^{\mathcal{D}}_{TM_Y}(k)$ es un P sistema determinista válido de decisión.
- Teniendo presente que la familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{TM_Y}}$ *simula* en tiempo polinomial la máquina de Turing TM_Y, resulta que dicha familia *resuelve* el problema de decisión asociado a esa máquina y, en consecuencia, la familia citada es Y–uniforme y polinomialmente acotada.
- La familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{TM_Y}}$ es Y–adecuada. En efecto: Para cada $w \in I_Y$ se verifica lo siguiente:

$$\begin{aligned} a_Y(w) = 1 &\iff TM_Y \text{ acepta } w \\ &\iff \Pi^{\mathcal{D}}_{TM_Y}(|w|) \text{ acepta } g(w) \text{ (siendo } g \text{ una adecuada} \\ &\qquad \text{codificación polinomial de cadenas por} \\ &\qquad \text{multiconjuntos)} \\ &\iff \text{Toda computación de } \Pi^{\mathcal{D}}_{TM_Y}(|w|) \text{ con entrada} \\ &\qquad g(w) \text{ es de aceptación} \end{aligned}$$

Por tanto, se tendría que $Y \in \mathbf{PMC}^*_{\mathcal{D}}$. Lo que contradice la hipótesis de partida.

□

6.2 Simulación de P sistemas a través de máquinas de Turing

En esta sección se trata de establecer una condición necesaria para que se verifique la relación $\mathbf{P} \neq \mathbf{NP}$ en términos de irresolubilidad de problemas **NP**–completos en la clase de los P sistemas deterministas de decisión válidos. Para ello, veamos en primer lugar que los P sistemas de la clase citada pueden ser simulados por máquinas de Turing deterministas con costes adicionales de tipo polinomial.

Lema 6.5 *Sea Y un problema de decisión. Sea $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{Y}} = (\Pi^{\mathcal{D}}_Y(n))_{n\in\mathbf{N}^+}$ una familia de P sistemas deterministas de decisión válidos tal que resuelve el problema Y en tiempo polinomial. Para cada $n \in \mathbf{N}^+$ se puede construir en tiempo polinomial una máquina de Turing determinista, $TM_{Y,n}$, que simula al P sistema $\Pi^{\mathcal{D}}_Y(n)$ en tiempo polinomial, en el siguiente sentido:*

1. *Existe una máquina de Turing, TM', y un polinomio $s(n)$ tal que para cada $n \in \mathbf{N}^+$, $TM'(n)$ para en, a lo sumo, $s(n)$ pasos y devuelve la máquina de Turing $TM_{Y,n}$.*
2. *Para cada $w \in I_Y$, la máquina $TM_{Y,s_Y(w)}$ acepta w si y sólo si el P sistema $\Pi^{\mathcal{D}}_Y(|w|)$ acepta el multiconjunto $g(w)$, siendo g una adecuada codificación polinomial de cadenas por multiconjuntos.*
3. *Para cada $n \in N^+$, la máquina $TM_{Y,n}$ trabaja en tiempo polinomial; es decir, existe un polinomio $t(k)$ que depende sólo de la familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{Y}}$, tal que para cada $w \in I_Y$ la máquina $TM_{Y,|w|}$ para en $O(t(|w|))$ pasos.*

Demostración:
Sea $n \in \mathbf{N}^+$. Notemos

- A_n = número de membranas de $\Pi^{\mathcal{D}}_Y(n)$,
- B_n = número de símbolos del alfabeto de $\Pi^{\mathcal{D}}_Y(n)$,

- C_n = suma de las longitudes de las reglas de $\Pi_Y^{\mathcal{D}}(n)$,
- D_n = número de símbolos de la configuración inicial de $\Pi_Y^{\mathcal{D}}(n)$,

El alfabeto de trabajo de la máquina de Turing $TM_{Y,n}$ es $\Gamma_n \cup \{0, 1, *, \delta\}$, siendo Γ_n el alfabeto del P sistema $\Pi_Y^{\mathcal{D}}(n)$. A continuación describimos las cintas que va a tener la máquina de Turing $TM_{Y,n}$.

- Una *cinta de entrada.*
- $A_n \cdot B_n$ *cintas principales* para representar (en base 2) las multiplicidades de cada símbolo en cada membrana, a lo largo de la computación.
- $A_n \cdot B_n$ *cintas auxiliares de cálculo* para realizar las sumas y restas (en base 2), correspondientes a la ejecución de las reglas aplicables.
- $3 \cdot A_n$ cintas para representar la estructura de membranas. Concretamente para cada membrana-nodo se consideran tres cintas-membranas tales que:
  - La primera cinta codifica:
    - En su primera casilla, una marca ($*$ o B) para recorrer el árbol.
    - En su segunda casilla, una marca (δ o B) para representar si hay disolución.
    - En el resto de la cinta, se representa en base 2 la etiqueta de la membrana.
  - La segunda cinta codifica en base 2 la etiqueta del padre de dicha membrana.
  - La tercera cinta codifica en base 2 el nivel de dicha membrana en el árbol.
- Una *cinta–entorno* para representar la salida de la computación (usando para ello la primera casilla: Yes para aceptar y No para rechazar). En esta cinta sólo se escribirán los símbolos Yes y No.
- Para cada membrana, k, se consideran r_k cintas (siendo r_k el número de reglas de la membrana k) que representen las relaciones de prioridad y el carácter de aplicabilidad de acuerdo con el siguiente criterio:

– Si la regla R_k^i tiene mayor prioridad que R_k^j, entonces colocaremos un 1 en la posición (i,j) de la que llamaremos *matriz de prioridad* de la membrana k. Esta parte será estática, es decir, su contenido no se modifica a lo largo de la ejecución de la máquina.

	R_k^1	R_k^2	...	...	...	$R_k^{r_k}$		
R_k^1			...	...	...	...	...	...
R_k^2			...	...	...	...	...	...
$\vdots$			...	...	...	...	...	...
$\vdots$			...	...	...	...	...	...
$R_k^{r_k}$			...	...	...	...	...	...

– Si la regla R_k^i es aplicable, entonces pondremos un 1 en la posición (i,i). Esta parte será dinámica, es decir, puede ser modificada durante la ejecución.

El número total de este tipo de cintas es del orden $O((r_1+\cdots+r_{A_n})\cdot A_n) \subseteq O(C_n \cdot A_n)$.

- Para cada regla $r = (u_r, v_r, \delta_r)$ con $v_r = (b_1, in_1)\ldots(b_{A_n}, in_{A_n})(c, out)$ $(d, here)$, consideramos $3 \cdot B_n + A_n \cdot B_n + 1$ cintas:

  – B_n cintas, una por cada símbolo del alfabeto, que codifique el multiconjunto (multiplicidades de cada símbolo, en base 2) que determina el antecedente de la regla.

  – B_n cintas, una por cada símbolo del alfabeto, que codifique el multiconjunto asociado a *here*.

  – B_n cintas, una por cada símbolo del alfabeto, que codifique el multiconjunto asociado a *out*.

  – $A_n \cdot B_n$ cintas tales que para cada $j = 1,\ldots,A_n$ hay B_n cintas, una por cada símbolo del alfabeto, que codifique el multiconjunto asociado a in_j.

  – 1 cinta para codificar en la primera casilla con 1 ó 0 si está *habilitada* la regla, y en la segunda casilla con δ o B si contiene o no *disolución*.

Así pues, en cada membrana el número de cintas será $r_k(3B_n+A_n\cdot B_n+1)$. Luego el número total de *cintas–reglas* será del orden $O(A_n \cdot B_n \cdot C_n)$.

Seguidamente describimos la situación inicial (*estática*) de la máquina de Turing; es decir, la configuración básica de la máquina antes de recibir el dato de entrada.

- Las cintas principales (al igual que las cintas auxiliares de cálculo) codifican los multiconjuntos de la configuración inicial del P sistema $\Pi_Y^{\mathcal{D}}(n)$, sin dato de entrada.
- La cinta-membrana codifica la estructura de membranas inicial del P sistema $\Pi_Y^{\mathcal{D}}(n)$.
- La cinta-entorno tiene inicialmente todas sus casillas en blanco.
- En la matriz de prioridad se representarán dos elementos:
  - La relación de prioridad (estática): colocando un 1 en la posición (i, j) de la matriz correspondiente a la membrana k si la regla R_k^i tiene una prioridad mayor que la regla R_k^j.
  - La condición de aplicabilidad de las reglas (dinámica): que inicialmente aparecen en blanco.
- Las cintas-reglas de cada membrana codificarán las reglas de dicha membrana de acuerdo con el siguiente criterio:
  - Las B_n cintas del *antecedente* (respectivamente, del *here* o del *out*) codifican las multiplicidades que cada símbolo tiene en el antecedente (respectivamente, en el *here* o en el *out*).
  - Para cada $j = 1, \ldots, A_n$, las B_n cintas asociadas a la membrana j, una por cada símbolo, codifican la multiplicidad de cada símbolo que corresponde al in_j.
  - La primera casilla de la última cinta aparece inicialmente con valor 1, y está reservada para codificar la propiedad de *habilitación* de la regla (es decir, inicialmente todas las reglas se consideran habilitadas).
  - La segunda casilla de la última cinta contendrá un símbolo δ si la regla contiene disolución y un símbolo blanco en caso contrario.

Para describir la simulación del P sistema $\Pi_Y^{\mathcal{D}}(n)$ a través de la máquina de Turing $TM_{Y,n}$, en primer lugar hay que observar que dado un dato de entrada

del P sistema hay que *añadirlo* a la configuración inicial del mismo a través de su membrana de entrada. Por ello, en la máquina de Turing se simula este proceso como sigue:

- Para cada símbolo de la cinta de entrada hay que sumar 1 a la multiplicidad del mismo en la correspondiente cinta-membrana-de-entrada-del-símbolo.
- El número de pasos que se necesita para llevar a cabo este proceso es el siguiente: para cada símbolo hay que sumar 1 a la correspondiente cinta-membrana-de-entrada-de-ese-símbolo. Dicha membrana contendrá una multiplicidad del símbolo menor o igual que D_n; sumarle 1 tiene un coste del orden $O(\lg D_n)$. Por tanto, si el dato de entrada es de tamaño k, codificar ese dato en la cinta-membrana-de-entrada de $TM_{Y,n}$ tiene un coste del orden $O(k \cdot \lg D_n)$.

La simulación de un paso de transición del P sistema $\Pi_Y^{\mathcal{D}}(n)$ se hará en varias etapas:

1. Seleccionar los conjuntos de reglas aplicables en cada membrana de la configuración correspondiente al paso que se analiza.
2. Actualización de los multiconjuntos de cada membrana (incluido, en su caso, el entorno).
3. Marcar las membranas que se van a disolver.
4. Habilitar todas las reglas.
5. Desactivar todas las reglas aplicables.
6. Actualizar la estructura de membranas.
7. Chequear la cinta del entorno y decidir, en su caso.

A continuación vamos a ver cómo se implementa en la máquina de Turing cada una de estas etapas e iremos calculando los costes respectivos.

1. Seleccionar los conjuntos de reglas aplicables en cada membrana
En esta etapa se procede así: para cada membrana de la configuración con la que se comienza a simular el paso hacer lo siguiente:

- Considerar la primera regla habilitada de dicha membrana.
- Si dicha regla es aplicable, entonces colocar un 1 en la correspondiente diagonal de la correspondiente matriz de prioridad, y deshabilitar todas las reglas de menor prioridad.
- Si una regla aplicable contiene disolución, marcar la membrana.

Ahora bien, ¿qué tiempo es necesario para decidir si una regla $r = (u_r, v_r, \delta_r)$ con $v_r = (b_1, in_1) \dots (b_{A_n}, in_{A_n})(c, out)(d, here)$, de una membrana k en un instante t, es aplicable?

- En primer lugar se mira si u_r está contenido en el multiconjunto de la membrana k en el instante t. Para ello

  Para cada símbolo del alfabeto hacer

  hallar la multiplicidad, x, de dicho símbolo en la regla

  hallar la multiplicidad, y, de dicho símbolo en M_k^t

  si $x > y$ devolver NO

  El número de pasos a realizar es del orden $O(B_n \cdot (C_n + D_n))$.

- Para cada $j = 1, \dots, A_n$ hacer

  si $b_j \neq \lambda$ entonces mirar si j es hijo de k

  El número de pasos a realizar es del orden $O(A_n \cdot (B_n + A_n))$.

- Si k es la membrana raíz y $\delta_r = 1$ devolver NO.

  El número de pasos a realizar es del orden $O(1)$.

En caso de que la regla sea aplicable hemos de *activarla* colocando un 1 en la diagonal correspondiente a la regla en la matriz de prioridad. Si está en la membrana k el número de pasos a realizar es del orden $O(r_k) \subseteq O(C_n)$. Además, hemos de *deshabilitar* todas las reglas de la membrana k que tienen menor prioridad, que requiere un número de pasos del orden $O(r_k) \subseteq O(C_n)$.

Por tanto, en cada regla de la membrana k el número de pasos a realizar es del orden $O(A_n \cdot B_n + A_n^2 + C_n^2 + C_n \cdot D_n)$.
En consecuencia, el número total de pasos a realizar en la etapa 1 es del orden $O(A_n \cdot (A_n \cdot B_n + A_n^2 + C_n^2 + C_n \cdot D_n)) = O(A_n^3 + A_n^2 \cdot B_n + A_n \cdot C_n^2 + A_n \cdot C_n \cdot D_n))$.

2. Marcar las membranas a disolver

Para marcar las membranas a disolver basta analizar las diagonales de las matrices de prioridad de cada membrana para decidir qué reglas son aplicables, y para cada una de ellas determinar si disuelve o no la membrana a la que pertenece, marcando en este caso con δ la casilla de la cinta asociada a la disolución de dicha membrana. El número total de pasos para implementar esta acción es del orden $O(A_n \cdot C_n^2)$.

3. Actualización de los multiconjuntos de cada membrana

Veamos cómo actualizar los multiconjuntos tras la ejecución de la regla $r = (u_r, v_r, \delta_r)$ con $v_r = (b_{j_1}, in_{j_1}) \ldots (b_{j_p}, in_{j_p})(c, out)(d, here)$, de una membrana k en un instante t.

Teniendo presente que inicialmente el número total de símbolos que tenemos en los multiconjuntos es del orden $O(D_n)$, resulta que tras el primer paso, el número total de símbolos será del orden $O(D_n + C_n)$, tras el segundo paso será del orden $O(D_n + 2C_n)$. Luego en M_k^t todos los símbolos aparecerán con una multiplicidad del orden $O(D_n + t \cdot C_n)$; es decir, los números a sumar tendrán $O(\lg(D_n + t \cdot C_n))$ dígitos y, por tanto, su suma o resta requerirá un número de pasos del orden $O(\lg(D_n + t \cdot C_n))$.

Teniendo presente que el número total de sumas y restas es del orden $O(A_n \cdot C_n + 2C_n + C_n)$, el tiempo necesario para la actualización de todos los multiconjuntos de la membrana k será $O((A_n \cdot C_n + 2C_n + C_n) \cdot \lg(D_n + t \cdot C_n)) = O(A_n \cdot C_n \cdot \lg(D_n + t \cdot C_n))$

Y el tiempo necesario para la actualización de todos los multiconjuntos de todas las membranas será del orden

$$O(A_n \cdot A_n \cdot C_n \cdot \lg(D_n + t \cdot C_n)) = O(A_n^2 \cdot C_n \cdot \lg(D_n + t \cdot C_n))$$

Desde luego, la actualización del entorno requiere un tiempo constante ya que sólo hay que hacerla en el último paso de la computación.

4. Habilitar todas las reglas

Basta ir a la primera casilla de las últimas cintas asociadas a las reglas y poner un 1. El número de pasos que se necesita es del orden $O(r_1 + \cdots + r_k) \subseteq O(C_n)$.

5. Desactivar todas las reglas aplicables

Basta poner un cero en todas las diagonales de las matrices de prioridad asociadas a cada membrana. Para la membrana k el tiempo que se necesita es $1+2+\cdots+r_k \in \theta(r_k^2) \subseteq O(C_n)$. Por tanto, el tiempo total que se necesita en esta etapa es del orden

$$\theta(r_1^2+\cdots+r_{A_n}^2) \subseteq O(A_n \cdot C_n^2)$$

6. Actualizar la estructura de membranas

Para actualizar la estructura de membranas se procede como sigue:

- Se efectúa un recorrido en anchura del árbol, nivel por nivel.
- Cada vez que encontremos un nodo marcado con disolución procedemos así:
  - Se elimina dicho nodo.
  - Se reestructuran los padres de sus hijos y los niveles de sus descendientes.
  - Se vuelca su contenido en el padre.

Esto se puede implementar como sigue:

para $k = 1$ hasta $O(A_n)$ hacer
  para cada nodo de nivel k no marcado hacer
    marcar el nodo
    si dicho nodo contiene la marca δ entonces
      disolver dicho nodo (poniendo B en la primera casilla de la cinta correspondiente a su nivel)
      actualizar los padres de sus hijos
      actualizar los niveles de sus descendientes
      volcar el contenido en su padre

Veamos el número de pasos que se necesita para llevar a cabo dichas tareas.

- El número de vueltas del bucle es $O(A_n)$.
- En cada vuelta, se analizan todos los nodos de ese nivel (coste $O(A_n)$) y para cada nodo no marcado se hace lo siguiente:
  - Se marca el nodo con $*$: $O(1)$.
  - Se mira si tiene la marca δ: $O(1)$.

- En caso, afirmativo,
  - Se disuelve: $O(1)$.
  - Se actualizan los padres de sus hijos: $O(A_n)$.
  - Se actualizan los niveles de sus descendientes: $O(A_n \cdot \lg A_n)$.
  - Se vuelca el contenido en el de su padre: $O(C_n \cdot (\lg(D_n + tC_n))$.

Por tanto, el número total de pasos a realizar en esta etapa es del orden

$$O(A_n^2 \cdot A_n \cdot \lg A_n + C_n \lg(D_n + t \cdot C_n))$$

7. Chequear la cinta del entorno y decidir, en su caso

Para chequear la cinta del entorno basta analizar su primera casilla y ver si hay un símbolo Yes o No. El número de pasos necesarios es del orden $O(1)$.

Coste en tiempo de la simulación

En consecuencia, el número total de pasos a realizar en la simulación del P sistema $\Pi_Y^{\mathcal{D}}(n)$ a través de la máquina de Turing $TM_{Y,n}$, es del orden

$$O(A_n^2 \cdot C_n^2 \cdot \lg A_n \cdot \lg(D_n + t \cdot C_n) + A_n^3 + A_n^2 \cdot B_n + A_n \cdot C_n^2 + A_n \cdot C_n \cdot D_n)$$

Las máquinas de Turing trabajan en tiempo polinomial

Veamos que para cada $n \in \mathbf{N}^+$ la máquina de Turing $TM_{Y,n}$ trabaja en tiempo polinomial. En efecto: Como la familia $\mathbf{\Pi_Y^{\mathcal{D}}}$ es Y–uniforme, existe un polinomio $p(n)$ que depende de la familia tal que para cada $n \in \mathbf{N}^+$ se tiene que $A_n, B_n, C_n, D_n \in O(p(n))$. Sea $w \in I_Y$ tal que $|w| = n$. Sea $t(n)$ un polinomio tal que $\Pi_Y^{\mathcal{D}}(n)$ decide $g(w)$ en $t(n)$ pasos. Entonces el número de pasos que necesita la máquina de Turing $TM_{Y,n}$ para decidir el dato de entrada w es del orden

$$O(t(n) \cdot (A_n^2 \cdot C_n^2 \cdot \lg A_n \cdot \lg(D_n + t(n) \cdot C_n) + A_n^3 + A_n^2 \cdot B_n + A_n \cdot C_n \cdot D_n))$$

Es decir, la máquina de Turing $TM_{Y,n}$ trabaja en tiempo del orden

$$O(p^4(n) \cdot \lg(p(n)) \cdot \lg(p(n) + t(n)p(n)) \subseteq O(\tau^6(n))$$

siendo $\tau(n) = p(n) + t(n)$.

Coste de la construcción de las máquinas de Turing

Veamos ahora que la construcción de la máquina de Turing $TM_{Y,n}$ a partir del P sistema $\Pi_Y^{\mathcal{D}}(n)$ es polinomial. En efecto:

- El número total de cintas es del orden $O(A_n \cdot B_n \cdot C_n)$.

- El proceso de inicialización requiere:
  - Cintas principales y auxiliares de cálculo:
    $$O(2A_n \cdot B_n \cdot D_n) = O(A_n \cdot B_n \cdot D_n)$$
  - Cinta-membrana: $O(A_n)$.
  - Cinta de entrada y Cinta-entorno: $O(1)$.
  - Matrices de prioridad: $O(C_n^2)$.
  - Cintas-reglas: $O(A_n \cdot B_n \cdot C_n \cdot D_n)$.
- Función de transición.

  Sea $w \in I_Y^n$ un dato de entrada del problema de tamaño n. Para la simulación de un paso de transición de la computación de $\Pi_Y^{\mathcal{D}}(n)$ con entrada w a través de la máquina de Turing $TM_{Y,n}$ se necesita, en primer lugar, un estado por cada símbolo $s \in \Gamma_n$ a fin de controlar la actualización de la cinta correspondiente de la membrana de entrada que está asociada a ese símbolo (pues hay que *trasladar* el dato w desde la cinta de entrada de la máquina $TM_{Y,n}$ a las cintas de dicha máquina que codifican la membrana de entrada del P sistema $\Pi_Y^{\mathcal{D}}(n)$).

  Por otra parte, es suficiente considerar un estado por cada procedimiento *básico* que hay que realizar. Ahora bien, es constante el número de tales procedimientos básicos (leer el contenido de una cinta, comparar, sumar o restar dos números, consultar y, en su caso, cambiar el contenido de una casilla, realizar un recorrido en anchura de un árbol, eliminar un nodo de un árbol, etc), así como el número de estados que una máquina de Turing necesita para implementar dichos procedimientos. Por tanto, el número total de estados que se necesitan para realizar la simulación sería del orden $O(B_n)$.

  En consecuencia, el número de pares ordenados que determinan la función de transición será del orden $O(B_n^2)$.

En consecuencia, el tiempo necesario para la construcción de la máquina de Turing $TM_{Y,n}$ es del orden $O(A_n \cdot B_n^2 \cdot C_n^2 \cdot D_n) \subseteq O(p^6(n))$.

□

Teorema 6.6 *Si* $\mathbf{P} \neq \mathbf{NP}$ *entonces ningún problema* $\mathbf{NP}$*-completo puede ser resuelto en tiempo polinomial, respecto del tamaño del dato de entrada, por una familia de P sistemas deterministas de decisión válidos. Es decir,*

$$\mathbf{P} \neq \mathbf{NP} \Rightarrow \forall\, Y\ (Y \in \mathbf{NPC} \longrightarrow Y \notin \mathbf{PMC}^*_{\mathcal{D}})$$

Demostración:
Supongamos que existiera un problema $\mathbf{NP}$–completo, Y, tal que $Y \in \mathbf{PMC}^*_{\mathcal{D}}$. Entonces existiría una familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{Y}} = (\Pi^{\mathcal{D}}_Y(n))_{n\in\mathbf{N}^+}$, de P sistemas deterministas de decisión válidos (y, por tanto, con entrada) que resuelve el problema Y en tiempo polinomial. Es decir, tal que:

1. La familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{Y}}$ es $\mathcal{D}$–consistente; es decir, para cada $n \in \mathbf{N}^+$, el P sistema $\Pi^{\mathcal{D}}_X(n)$ es un P sistema determinista de decisión válido.

2. La familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{Y}}$ es Y–uniforme; es decir, existe una máquina de Turing que, a partir de $n \in \mathbf{N}^+$ construye $\Pi^{\mathcal{D}}_Y(n)$ en tiempo polinomial. Con otras palabras, existe una máquina de Turing, TM', y un polinomio, $p(n)$, dependiente del problema Y tal que para cada $n \in \mathbf{N}^+$, $TM'(n)$ para en, a lo sumo, $p(n)$ pasos y devuelve el P sistema $\Pi^{\mathcal{D}}_Y(n)$.

3. Existe un polinomio $r(n)$ tal que la familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{Y}}$ está r–acotada; es decir, para cada $n \in \mathbf{N}^+$ toda computación de $\Pi^{\mathcal{D}}_Y(n)$ para en $O(r(n))$ pasos.

4. La familia $\mathbf{\Pi}^{\mathcal{D}}_{\mathbf{Y}}$ es Y–adecuada; es decir, existe una codificación polinomial, g, de las instancias del problema Y en multiconjuntos de tal manera que para cada $w \in I_Y$ se tiene que $a_Y(w) = 1$ si y sólo si toda computación de $\Pi^{\mathcal{D}}_Y(|w|)$ con entrada $g(w)$ es de aceptación. Que g sea una codificación polinomial de las instancias del problema Y en multiconjuntos significa que existe una máquina de Turing, TM'', y un polinomio $q(n)$ tal que para cada $w \in I_Y$, $TM''(w)$ para en, a lo sumo, $q(|w|)$ pasos y, además, devuelve $g(w) \in \mathbf{M}(\Gamma_{|w|})$, siendo $\Gamma_{|w|}$ el alfabeto del P sistema $\Pi^{\mathcal{D}}_Y(|w|)$.

Del lema 6.5 se deduce que para cada $n \in \mathbf{N}^+$ se puede construir en tiempo polinomial una máquina de Turing determinista, $TM_{Y,n}$ que simula al P sistema $\Pi^{\mathcal{D}}_Y(n)$ en tiempo polinomial, en el sentido indicado en el lema.
Consideremos la siguiente máquina de Turing, TM_Y, "universal" respecto de la familia de máquinas $\{TM_{Y,n} :\ n \in \mathbf{N}^+\}$:

Entrada: $w \in I_Y$.
Simular $TM_{Y,|w|}$ con entrada w.

Veamos que la máquina de Turing determinista TM_Y resuelve el problema Y en tiempo polinomial.

1. TM_Y resuelve el problema Y.

   En efecto: sea $w \in I_Y$ tal que $a_Y(w) = 1$. Entonces toda computación de $\Pi_Y^{\mathcal{D}}(|w|)$ es de aceptación. En particular, la máquina $\Pi_Y^{\mathcal{D}}(|w|)$ acepta el multiconjunto $g(w)$ y, en consecuencia, la máquina de Turing $TM_{Y,|w|}$ acepta w. Recíprocamente, sea $w \in I_Y$ tal que TM_Y acepta w. Entonces $TM_{Y,|w|}$ acepta w. Luego $\Pi_Y^{\mathcal{D}}(|w|)$ acepta $g(w)$. Por tanto, $a_Y(w) = 1$.

2. TM_Y trabaja en tiempo polinomial.

   En efecto: sea $w \in I_Y$. Sea $k = |w|$. En tiempo $O(p^6(k))$ se construye la máquina de Turing $TM_{Y,k}$. En tiempo $O(t(k))$ se decide si la máquina $TM_{Y,k}$ acepta w. Por tanto, la máquina TM_Y decide la aceptación de w en tiempo $O(p^6(k) + t(k))$.

Por tanto, tendríamos que $Y \in \mathbf{P}$. En consecuencia, se tendría que $\mathbf{P} = \mathbf{NP}$.
□

6.3 Caracterización de la relación $\mathbf{P} \neq \mathbf{NP}$ a través de P sistemas

Estamos ya en condiciones de establecer caracterizaciones de la relación $\mathbf{P} \neq \mathbf{NP}$ mediante la irresolubilidad de problemas **NP**–completos por familias de P sistemas deterministas de decisión.

Teorema 6.7 *Son equivalentes las proposiciones siguientes:*

1. $\mathbf{P} \neq \mathbf{NP}$.
2. $\exists\, Y\ (Y \in \mathbf{NPC} \ \wedge\ Y \notin \mathbf{PMC}_{\mathbf{D}}^{*})$.

   Es decir, existe un problema **NP***-completo que no puede ser resuelto en tiempo polinomial, respecto del tamaño del dato de entrada, por una familia de P sistemas deterministas de decisión válidos.*

3. $\forall\, Y\ (Y \in \mathbf{NPC} \longrightarrow Y \notin \mathbf{PMC}^{*}_{\mathbf{D}})$.

   Es decir, ningún problema **NP***-completo puede ser resuelto en tiempo polinomial, respecto del tamaño del dato de entrada, por una familia de P sistemas deterministas de decisión válidos.*

Demostración:

La implicación (2) $\Rightarrow$ (1) se deduce del lema 6.5.
La implicación (1) $\Rightarrow$ (3) se deduce del teorema 6.6.
La implicación (3) $\Rightarrow$ (2) es trivial, ya que la clase **NPC** es no vacía.

□

Para terminar, queremos hacer notar que el establecimiento de estas equivalencias proporciona nuevos medios para atacar la conjetura $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ y, a la vez, abre nuevas vías de investigación futura en el marco de los modelos de computación celular con membranas a fin de poder afrontar el reto sugerido.

Por una parte, para obtener una respuesta afirmativa a la conjetura citada parece necesario la búsqueda de técnicas sofisticadas de diseño de P sistemas de decisión que puedan proporcionar una solución polinomial de algún problema **NP**-completo.

Por otra parte, si se busca una respuesta negativa habría que desarrollar herramientas que permitan establecer que una cierta propiedad (en este caso, *no trabajar* en tiempo polinomial) la verifiquen *todos* los procedimientos del modelo que resuelvan un problema **NP**-completo. Lo que puede conducirnos al estudio de nuevos conceptos relativos a la *reducibilidad* en tiempo polinomial, tanto de problemas de decisión como de familias de P sistemas, e, indirectamente, con algunas cuestiones relacionadas con la *indecidibilidad* de problemas en el modelo de Computación Celular con Membranas.

Bibliografía

[1] Adleman, L. Molecular Computation of Solutions to Combinatorial Problems. *Science*, 268 (November 1994), 1021–1024.

[2] Arroyo, F.; Luengo, C.; Baranda, V.; Mingo, L.F. A software simulation of transition P systems in Haskell. *Pre-proceedings of Workshop on Membrane Computing.* MolCoNet project- IST-2001-32008, publication No. 1 (Gh. Păun and C. Zandron, eds), (2002), 29–43.

[3] Balbontín, D.; Pérez-Jiménez, M.J.; Sancho, F. A MzScheme implementation of transition P systems. *Pre-proceedings of Workshop on Membrane Computing.* MolCoNet project- IST-2001-32008, publication No. 1 (Gh. Păun and C. Zandron, eds), (2002), 65–80.

[4] 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.

[5] Berry, G.; Boudol, G. The chemical asbtract machine. *Theoretical Computer Science*, 96 (1992), 217–248.

[6] Blum, M. A machine-independent theory of the complexity of recursive functions. *Journal ACM*, vol. 14, núm. 2 (1967), 322-336.

[7] Castellanos, J.; Păun, Gh.; Rodríguez-Patón, A. Computing with membranes: P systems with worm-objects. CDMTCS, Technical Report Nº 123, 2000.

[8] CHURCH, A. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1 (1936), 40–41.

[9] CHURCH, A. An unsolvable problem of elementary number systems. *American Journal of Mathematics*, 58 (1936), 345–363.

[10] COOK, S.A. The complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium in Theory of Computing*, (1971), 151–158.

[11] DASSOW, J.; PĂUN, G. *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[12] FEYNMAN R.P. There's plenty of room at the bottom. En *Miniaturization* (D.H. Hilbert, ed.), Reinhold, 1961, 282–296.

[13] FREUND, R.; MARTÍN-VIDE, C; PĂUN, G. From Regulated Rewriting to Computing with membranes: Collapsing Hierarchies. Submitted, 2001.

[14] GAREY, M.R.; JOHNSON, D.S. *Computers and intractability*, W.H. Freeman and Company, New York, 1979.

[15] GÖDEL, K. Uber formal unentscheidbare Sätze Principia Mathematica und verwandter Systeme, I. *Monast. Math. Phys.*, 38 (1931), 173–198.

[16] 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.

[17] 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.

[18] HEAD, T. Aqueous Simulations of Membrane Computations. *Romanian Journal of Information Science and Technology*, vol. 4, núm. 1-2 (2001).

[19] HOLLAND, J.H. *Adaptation in natural and artificial systems.* MIT Press, 1975.

[20] ITO, M.; MARTÍN-VIDE, C.; PĂUN, G. A characterization of Parikh sets of ETOL languages in terms of P systems. Submitted, 2001.

[21] KARI, L. DNA Computing: Arrival of Biological Mathematics. *The Mathematical Intelligencer*, Springer–Verlag, New York, vol. 19, num. 2 (1997), 9–22.

[22] KARP, R.M. Reducibility among combinatorial problems. *Complexity of Computer Computations*, Plenum Press, New York, 1972, 85–104.

[23] KRISHNA, S. N. *Languages of P Systems: Computability and Complexity*, Ph.D. Thesis, Indian Institute of Technology, Madras, 2001.

[24] 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.

[25] 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.

[26] KRISHNA, S. N.; RAMA, R. P systems with replicated rewriting, *Journal of Automata, Languages and Combinatorics*, vol. 6, núm. 3 (2001), 345–350.

[27] 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.

[28] MANCA, V. String rewriting and metabolism: A logical perspective. En *Computing with Bio-Molecules. Theory and Experiments*, Springer-Verlag, Singapore, 1998, 36–60.

[29] MARTÍN-VIDE, C.; MITRANA, V.; PĂUN, G. On the power of P systems with valuations. *Computación y sistemas*, to appear.

[30] MARTÍN-VIDE, C.; PĂUN, G.; ROZENBERG, G. Membrane systems with carriers. *Theoretical Computer Science*, 270 (2002), 779–796.

[31] MCCULLOCH, W.S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, vol. 5 (1943), 115–133.

[32] MINSKY, M.; PAPERT, S. *Perceptions*, Cambridge, M.A. MIT Press.

[33] OBTULOWICZ, A. Membrane computing and one-way functions. *International Journal of Foundations of Computer Science*, 12, 4 (2001), 551–558.

[34] 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.

[35] PĂUN, A.; PĂUN, G. The power of communication: P systems with Symporters-Antiporters. *New Generation Computers.*

[36] 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)

[37] 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`.

[38] PĂUN, GH. Further research topics about P systems. *Pre-Proceedings of Workshop on Membrane Computing*, Curtea de Arges, Romania, Agosto 2001, Technical Report 17/01 of Research Group on Mathematical Mathematical Linguistics, Rovira i Virgili University, Tarragona, España, 2001, 243–250.

[39] PĂUN, GH. *Membrane Computing. An introduction.* Springer-Verlag, 2002

[40] PĂUN, GH.; PÉREZ-JIMÉNEZ, M.J.; SANCHO, F. On the Reachability Problem for P systems with Porters. *Proceedings of the 10th International Conference on Automata and Formal Languages*, Debrecen, Hungary, 2002.

[41] PĂUN, GH.; PÉREZ-JIMÉNEZ, M.J. Recent Computing Models Inspired from Biology: DNA and Membrane Computing. *Theoria*, en prensa.

[42] PĂUN GH., ROZENBERG, G. A guide to membrane computing. *Theoretical Computer Science*, to appear.

[43] PĂUN, GH.; ROZENBERG, GR.; SALOMAA, A. *DNA Computing. New Computing Paradigms*, Springer, 1998.

[44] Pérez-Jiménez, M.J.; Sancho, F. A formalization of transition P systems. *Fundamenta Informaticae*, vol. 49, 1–3 (2002), 261–272.

[45] 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.

[46] Pérez-Jiménez, M.J.; A. Romero Jiménez; Sancho, F. Decision P systems and the $\mathbf{P} \neq \mathbf{NP}$ conjecture. *Pre-proceedings of Workshop on Membrane Computing.* MolCoNet project- IST-2001-32008, publication No. 1 (Gh. Păun and C. Zandron, eds), (2002), 345–354.

[47] Romero-Jiménez, A.; Pérez-Jiménez, M.J. Simulating Turing Machines by P systems with External Output. *Fundamenta Informaticae*, vol. 49, 1–3 (2002), 273–287.

[48] Romero-Jiménez, A.; Pérez-Jiménez, M.J. Generation of Diophantine Sets by Computing P Systems with External Output. *Proceedings of the 3rd International Conference on Unconventional Models of Computation*, Himeji, Japan, 2002.

[49] Rosenblatt, F. The perception: a probabilistic model for information storage and organization in the brain. *Psychological Review*, vol. 65 (1959), 368–408.

[50] Salomaa, A. *Formal Languages.* Academic Press, New York, 1973.

[51] Sancho-Caparrini, F. *Verificación de programas en modelos de computación no convencionales.* Tesis Doctoral. Universidad de Sevilla, 2002.

[52] 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.

[53] Turing, A.M., Computability and λ-definability. *Journal of Symbolic Logic*, 2 (1937), 153–163.

[54] 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.

[55] P systems web page: http://psystems.disco.unimib.it/

[56] Página Web del Grupo de Investigación en Computación Natural de la Universidad de Sevilla: http://www.cs.us.es/gcn