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

Desarrollo y aplicaciones de un entorno de programación para Computación Celular: P-Lingua

Memoria presentada por
Ignacio Pérez Hurtado de Mendoza
para optar al grado de Doctor
por la Universidad de Sevilla

Ignacio Pérez Hurtado de Mendoza

V.º B.º Los Directores de la Tesis

Dr. D. Mario de J. Pérez Jiménez

Dr. D. Agustín Riscos Núñez

A Maca y Julio.
A J. Antonio, José M^a,
Alejandro y Carmen.
A Amalia.
Y, por supuesto, a Mina.

Agradecimientos

Dado que los comienzos son momentos difíciles para todo principiante, cualquier ayuda siempre es valiosa y digna de profunda gratitud. Por ello, deseo manifestar mi más sincero agradecimiento no sólo hacia las personas que han colaborado en la elaboración de esta memoria, sino también a todas aquellas que han hecho posible que me sienta orgulloso de haber tenido la fortuna de comenzar a trabajar en algo que me gusta y me llena de satisfacción. Mi agradecimiento se dirige especialmente a las personas que por su calidad humana enseñan con el ejemplo; pues, como afirman algunos grandes maestros, es ésta la única manera de enseñar y al mismo tiempo formar. Y en este contexto, me honro en destacar en primer lugar mi sentimiento de cordial gratitud hacia el profesor y maestro D. Mario de Jesús Pérez Jiménez, por tantos motivos que resulta imposible enumerarlos exhaustivamente en estas líneas (a menos que **P=NP**), pero sí puedo afirmar que están directamente relacionados con su calidad humana, paciencia, amistad, comprensión y su "defecto" de no cerrar los ojos ante los problemas de los demás.

Deseo asimismo agradecer a D. Agustín Riscos Núñez su incondicional apoyo, ayuda y aliento. A Dña. Mª Angels Colomer Cugat su contagiosa capacidad de trabajo, apoyo y confianza. A D. Fernando Sancho Caparrini y a D. Francisco José Romero Campero, sus múltiples y oportunos consejos. A D. Miguel Ángel Martínez del Amor su terapéutico sentido del humor, compañerismo y apoyo. A Dña. Ana Ruiz Gómez sus incontables ayudas en la traducción de textos, demostrando siempre una gran profesionalidad. A D. Miguel Ángel Gutiérrez Naranjo, el primero que usó P-Lingua en nuestro departamento, su apoyo y ayuda en la depuración de las herramientas desarrolladas. A Dña. Delia Balbontín Noval su confianza, apoyo y consejo desde el primer día. Asimismo, deseo hacer extensivo este sentimiento de gratitud al resto de los compañeros incluido el personal de administración y servicios, del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla.

Índice general

Índice de figuras

Introducción

La *Teoría de la Computabilidad* tiene su origen en la preocupación del hombre por facilitar el proceso de resolución de problemas a través de la búsqueda de métodos especiales que puedan ser llevados a cabo por personas o entidades que tengan la habilidad de realizar con exactitud ciertas tareas elementales, sin necesidad de tener un conocimiento exhaustivo acerca de las disciplinas en las que se enmarcan los problemas objetos de estudio.

Desde muy antiguo, se constató que con la ayuda de algunos aparatos mecánicos (por ejemplo, el ábaco que fue inventado en Oriente Próximo hacia el año 500 aC) era posible realizar ciertas operaciones básicas con la misma precisión pero con mucha más rapidez. El matemático árabe *Al'Khwarizmi* escribió un tratado de Álgebra y Astronomía, hacia el año 825 dC, en donde describe el sistema hindú de numeración, incluye procedimientos mecánicos para calcular fechas, y presenta las primeras tablas conocidas de algunas funciones trigonométricas. A partir de entonces, los procedimientos mecánicos se conocerían también por el nombre de *algoritmos*, en honor al científico árabe antes citado.

A lo largo de la historia existen muchos precedentes acerca de la construcción de distintos tipos de máquinas que sirvieron de asistente al hombre para la realización de determinados cálculos especialmente laboriosos. Entre ellas, cabe resaltar una máquina construida por B. Pascal en 1674 que era capaz de sumar y restar números naturales, o una máquina inspirada en la anterior que fue construida por G.W. Leibniz en 1694 y que era capaz de realizar multiplicaciones. No obstante, en este contexto hay que destacar por

encima de todo el esfuerzo denodado de Ch. Babbage que, entre 1833 y 1842, trató de construir una máquina (denominada *máquina analítica*) que fuera capaz no sólo de procesar información sino, además, de autocontrolar, en cierto sentido, su funcionamiento. Aunque esta máquina no pudo ser implementada en su época debido, básicamente, a las limitaciones tecnológicas, las ideas subyacentes en el diseño de dicha máquina son consideradas hoy día como el germen de la arquitectura de J. von Neumann, en la que se basa los principios fundamentales de los ordenadores electrónicos actuales.

A finales del siglo XIX empieza a tomar cuerpo la creencia de que existen problemas que no pueden ser resueltos mediante procedimientos mecánicos. Ahora bien, para poder asegurar la existencia de tales problemas no era suficiente considerar la idea informal de algoritmo. Se necesitaba una definición rigurosa de dicho concepto a fin de poder afirmar que un cierto problema no puede ser resuelto por ningún algoritmo que verifique las condiciones exigidas en la definición formal. Entre 1931 y 1936 aparecen los primeros modelos formales de computación, debidos a K. Gödel, A. Church, S. Kleene y A. Turing.

Por otra parte, la aparición de las primeras máquinas de propósito general, a finales de la década de los cuarenta del pasado siglo, da origen a lo que podríamos denominar *Teoría de la Computabilidad práctica*, entendida como disciplina cuyo objetivo es determinar los problemas abstractos que pueden ser resueltos por máquinas reales para *ejemplares de tamaño suficientemente grande*. Éste es, propiamente, el origen de la *Teoría de la Complejidad Computacional*.

Muchos de los esfuerzos realizados en la historia de la Teoría de la Computabilidad, en general, y de la Teoría de la Complejidad Computacional, en particular, han estado encaminados hacia el análisis y desarrollo de lo que podríamos denomianr *sistemas artificiales*. Hoy día está constatado que los sistemas/organismos vivos también disponen de mecanismos de procesamiento de la información que les permiten, entre otras cosas, mantener el equilibrio termodinámico, adaptarse al entorno y evolucionar en un sentido que favorezca su propia existencia.

La *Computación Natural* es una disciplina cuyo objetivo principal es el estudio y simulación de los procesos dinámicos que se dan en la Naturaleza y que son susceptibles de ser interpretados como procedimientos de cálculo. En ella, se investigan modelos y técnicas computacionales inspiradas en la Naturaleza con el objetivo de entender más y mejor el mundo que nos rodea, en términos de procesamiento de la información. Dentro de la Computación Natural se han desarrollado, por una parte, modelos de computación que pueden ser implementados usando materiales naturales (como moléculas de ADN, haces de luz, etc.) con el fin de realizar computaciones usando esos sustratos alternativos en lugar de procesadores basados en la manipulación electrónica del silicio. Por otra parte, se han desarrollado modelos de computación inspirados en la Naturaleza cuyo objetivo es desarrollar técnicas y herramientas que permitan la resolución eficiente de problemas abstractos. Entre las ramas correspondientes a la primera aproximación, destacamos la *Computación molecular basada en ADN* (iniciada con el experimento de L. Adleman a finales de 1994) y en algunos ámbitos se considera también la *Computación cuántica* que trata de generar nuevos paradigmas computacionales basados en los principios de la física cuántica. Entre las ramas de la segunda aproximación destacamos la *Computación evolutiva* (inspirada en conceptos tales como *evolución* y *selección* a fin de obtener soluciones adecuadas de problemas de optimización mediante estrategias de búsqueda en una "población" de soluciones candidatas) y la *Computación celular con membranas*, *Membrane Computing* (disciplina creada por Gh. Păun a finales de 1998 que proporciona dispositivos teóricos distribuidos, masivamente paralelos y no deterministas inspirados en la estructura y el funcionamiento de las células de los organismos vivos).

El objetivo principal de una investigación científica es conseguir un conocimiento y un control más profundo de alguna parte del universo, ninguna de las cuales es lo suficientemente simple como para poder ser explicada adecuadamente sin la ayuda de procesos de abstracción; es decir, mecanismos que permitan reemplazar la parte concreta objeto de estudio por un *modelo* que capture los hechos más relevantes y, a la vez, tenga una estructura general más

simple. El uso de modelos es indispensable debido a que el sistema objeto de investigación suele ser demasiado complejo debido, básicamente, a que o bien el número de interacciones entre los elementos del sistema es demasiado grande, o bien aparecen en el sistema una serie de elementos que no son accesibles a la observación.

En el caso de fenómenos biológicos, tanto a nivel microscópico como a nivel macroscópico, considerados como sistemas complejos, resulta que su dinámica o comportamiento no puede ser inferido a partir del análisis independiente de cada una de sus componentes principales. Con frecuencia, existe una jerarquía en distintos niveles de organización que corresponden a escalas de tiempo en los que ocurren los procesos y a una escala espacial en la que se desenvuelve el sistema. Uno de los principales retos de la Ciencia en la actualidad es el desarrollo de modelos matemático–computacionales que contemplen estas jerarquías con el fin de conseguir un conocimiento más profundo acerca de los sistemas que se estudian.

Hasta la presente se han considerado muchos tipos de modelos formales para el análisis de distintos fenómenos biológicos, desde aproximaciones específicamente matemáticas, como los modelos basados en sistemas de ecuaciones diferenciales hasta aproximaciones computacionales (y, por tanto, discretas) basadas en redes de Petri o en Membrane Computing. Desde luego, un mismo fenómeno o sistema complejo puede ser modelizado de manera exitosa desde diferentes aproximaciones y, cada una de ellas, con sus ventajas e inconvenientes pero con la posibilidad de ser usadas de manera conjunta y complementaria, en cierto sentido, a fin de mejorar el conocimiento acerca de la dinámica del sistema.

En el caso particular del desarrollo de modelos basados en la Computación celular con membranas, surgen algunas dificultades añadidas que se derivan del hecho de carecer de implementaciones (en medios electrónicos, bioquímicos, etc.) de los dispositivos (distribuidos, masivamente paralelos y no deterministas) de dicha aproximación computacional, denominados *sistemas P*. Por ello, para conseguir validar un modelo diseñado en este marco es imprescindible el desarrollo de aplicaciones software y hardware que permitan realizar

simulaciones del sistema a partir de distintos escenarios y contrastar los resultados obtenidos con datos experimentales extraídos a partir de pruebas y observaciones.

Es sabido que todo modelo de computación consta de una especificación sintáctica para su representación, y su dinámica se rige por medio de una semántica formal. Denominaremos *simulador* de un modelo formal de computación a toda aplicación software/hardware que describa la especificación a través de un cierto lenguaje de programación y capture la semántica mediante la implementación de un algoritmo de simulación que debe reproducir la dinámica con fidelidad; es decir, cada paso de computación del modelo formal es replicado en el simulador a través un número finito de pasos, de tal manera que el simulador sea capaz de proporcionar los elementos básicos del modelo que han intervenido de forma relevante en ese paso. Habitualmente, un programa de estas características consta de tres partes bien diferenciadas: (a) la definición del modelo formal que se desea simular; (b) el núcleo que implementa algún algoritmo de simulación; y (c) la interfaz de salida que muestra o graba la información relevante para el usuario.

A lo largo de los diez últimos años, se ha desarrollado un gran número de aplicaciones informáticas para la simulación de sistemas P, bien con una finalidad fundamentalmente pedagógica, o bien para el estudio y análisis de fenómenos de la vida real modelizados computacionalmente con sistemas celulares. La mayoría de estas aplicaciones se encuentran recopiladas en la página web oficial de los sistemas P [126].

Esta memoria se enmarca en el ámbito de las aplicaciones prácticas de la Computación celular con membranas y aborda, básicamente, dos puntos: (a) el desarrollo y posterior aplicación de técnicas y herramientas informáticas reutilizables y eficientes; y (b) el diseño de un marco general de modelización basado en sistemas P, así como su aplicación al estudio de la dinámica de dos ecosistemas reales de especial relevancia.

Contenido de la memoria

La presente memoria está estructurada en tres partes que constan de un total de siete capítulos cuyos contenidos se describen sucintamente a continuación.

Parte I: Preliminares

La memoria comienza con un breve recorrido histórico acerca del desarrollo y evolución de diversos conceptos relacionados con la Teoría de la Computación y la Teoría de la Complejidad Computacional. Se analizan las limitaciones que tienen los dispositivos reales construidos en el marco de dichas teorías, a la hora de resolver eficientemente problemas concretos que son relevantes en la vida real, lo cual hace necesario la búsqueda de nuevos paradigmas de computación que permitan superar algunas de esas limitaciones. Se presentan los conceptos básicos de la Computación Natural y, en particular, de la rama de la Computación celular con membranas, *Membrane Computing*, en la que se enmarcan los trabajos que se presentan en esta memoria.

En el **Capítulo 2** se analiza la problemática actual relativa a la modelización de procesos complejos de la realidad, justificando la necesidad de utilizar modelos formales a fin de conseguir ciertos avances de tipo cualitativo. Se describen brevemente algunas aproximaciones clásicas, así como otras más recientes de tipo computacional que tratan de capturar la aleatoriedad inherente a los procesos biológicos. Finalmente, se presenta un marco de especificación para el diseño de modelos basados en el paradigma de *Membrane Computing*.

Parte II: Aplicaciones de software en *Membrane Computing*

El **Capítulo 3** está dedicado a justificar la necesidad de desarrollar herramientas informáticas para poder analizar la bondad de los modelos computacionales diseñados, a través de una validación experimental basada en simulaciones. Se presenta una breve panorámica de los simuladores de sistemas P desarrollados hasta la fecha y se describen los elementos comunes que tienen todos ellos y que han de ser implementados.

En el **Capítulo 4** se presenta un entorno de programación para el paradigma de *Membrane Computing*. Dicho entorno está compuesto de: (a) un lenguaje de programación para la especificación de sistemas P (*P-Lingua*); (b) una biblioteca que implementa el procesamiento de ficheros y diversos algoritmos de simulación (*pLinguaCore*); y (c) una serie de herramientas para la línea de comandos. La última parte está dedicada a la presentación de ejemplos de código de distintos sistemas P, incluyendo dos familias que proporcionan soluciones eficientes del problema **SAT** de la satisfactibilidad de la Lógica Proposicional.

Parte III: Aplicación al estudio de ecosistemas reales

En el **Capítulo 5** se describe un marco específico para la modelización computacional de ecosistemas basado en *Membrane Computing*. En particular, se presentan las concreciones del marco general para los sistemas P probabilísticos y, además, se describe con detalle un algoritmo de simulación que trata de capturar la semántica probabilística. Además, se presenta el software *EcoSim 2.0* que es una familia de herramientas para la simulación de modelos de ecosistemas y que se ha desarrollado utilizando *P-Lingua* y *pLinguaCore*. El capítulo finaliza con un ejemplo simple de ilustración acerca de las *interacciones tritróficas*.

El **Capítulo 6** está dedicado al diseño de modelos basados en sistemas P de dos ecosistemas reales. El primero de ellos se refiere a un ecosistema de la zona pirenaico–catalana en donde habita una ave carroñera en peligro de extinción, el *quebrantahuesos*; el segundo está dedicado a la modelización computacional de un ecosistema del pantano de Ribarroja, gestionado por Endesa S.A. en el que una especie exótica invasora (el *mejillón cebra*) está causando importantes problemas de sostenibilidad del medio ambiente con la destrucción de especies autóctonas, así como graves problemas de tipo económico al provocar importantes destrozos en instalaciones diversas de la compañía en torno a dicho pantano. En ambos casos, se han diseñado sendos modelos basados en sistemas P multientornos funcionales–probabilísticos para los cuales se han desarrollado aplicaciones informáticas *ad hoc* que permitan

la validación experimental de los modelos y su uso para la realización de experimentos virtuales a partir de diferentes escenarios de interés para los expertos.

La memoria finaliza con un breve capítulo dedicado a la presentación de conclusiones y líneas de trabajo futuro de investigación.

Aportaciones

Entre las aportaciones originales más relevantes que se recogen en la presente memoria, caben destacar las siguientes:

- Desarrollo de un marco formal de modelización basado en el paradigma de *Membrane Computing* que facilita el estudio y análisis de procesos complejos de la realidad, en general, y procesos biológicos, en particular, considerados tanto a nivel microscópico como al macroscópico, capturando su inherente aleatoriedad mediante el uso de estrategias estocásticas y probabilísticas.
- Desarrollo de *P-Lingua*, el primer lenguaje de programación de alto nivel que permite la descripción de sistemas P de una manera sencilla, paramétrica y modular. El lenguaje se presenta junto con una serie de bibliotecas y herramientas asociadas, conformando un entorno completo de programación. Todo el conjunto se ha desarrollado bajo los estándares del software libre, con licencia GNU GPL y, actualmente, existe una comunidad de usuarios y desarrolladores de P-Lingua en las Universidades de Sevilla, Lleida, Pitesti (Rumanía), Sheffield (U.K.), París XII (Francia) y Universidad Autónoma de Madrid.
- Especificación de elementos comunes que se deben implementar en los simuladores de sistemas P, así como el diseño e integración de diversos algoritmos de simulación, en el marco de P-Lingua.
- Desarrollo de las primeras aplicaciones informáticas para la validación experimental y la experimentación virtual sobre modelos de ecosistemas

reales basados en sistemas P. La peculiaridad de estas aplicaciones radica en que están diseñadas para usuarios finales que no necesitan conocer los detalles del modelo para su ejecución; es decir, las simulaciones se realizan de forma transparente. Las herramientas están siendo actualmente usadas por expertos en ecología con la finalidad de filtrar hipótesis plausibles que permitan gestionar de manera eficiente los ecosistemas reales objetos de estudio, en función de sus necesidades.

Publicaciones

Algunos de los resultados más relevantes del trabajo desarrollado en esta memoria han sido publicados en las revistas y monografías que se citan a continuación.

1. M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A Computational Modeling for Ecosystems Based on P Systems. *Natural Computing*, versión online (`http://dx.doi.org/10.1007/s11047-010-9191-3`).

2. M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A P system based model of an ecosystem of some scavenger birds. *Lecture Notes in Computer Science*, 5957 (2010), 182-195.

3. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Implementing P systems parallelism by means of GPUs. *Lecture Notes in Computer Science*, 5957 (2010), 227-241.

4. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulation of P systems with active membranes on CUDA. *Briefings in Bioinformatics* **JCR 4.627**, versión online (`http://dx.doi.org/10.1093/bib/bbp064`).

5. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulating a P system based efficient solution to SAT by using GPUs. *Journal of Logic and Algebraic Programming* **JCR 1.018**, versión online (`http://dx.doi.org/10.1016/j.jlap.2010.03.008`).

6. D. Díaz, C. Graciani, M.A. Gutiérrez, I. Pérez–Hurtado, M.J. Pérez-Jiménez. Software for P systems. In Gh. Păun, G. Rozenberg, A. Salomaa (eds.) *The Oxford Handbook of Membrane Computing*, Oxford University Press, Oxford (U.K.), 2009, Chapter 17, pp. 437-454.

7. D. Díaz, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos. A P-lingua programming environment for Membrane Computing. *Lecture Notes in Computer Science*, 5391 (2009), 187-203.

8. M. García-Quismondo, R. Gutiérrez-Escudero, M.A. Martínez del Amor, E. Orejuela, I. Pérez-Hurtado. P-Lingua 2.0: A software framework for cell–like P systems. *International Journal of Computers, Communications and Control*, IV, 3 (2009), 234-243.

9. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. An overview of P-Lingua 2.0. *Lecture Notes in Computer Science*, 5957 (2010), 264-288.

10. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. A P-Lingua based simulator for Tissue P systems. *Journal of Logic and Algebraic Programming* **JCR 1.018**, versión online (`http://dx.doi.org/10.1016/j.jlap.2010.03.009`).

Parte I

Preliminares

Capítulo 1

Computación Bioinspirada

En este primer capítulo vamos a realizar, en primer lugar, un breve recorrido histórico por el desarrollo y evolución de diversos conceptos relacionados directamente con la Teoría de la Computación y la Teoría de la Complejidad Computacional. Desde las primeras formulaciones de cuestiones relevantes que marcarían el devenir de las Ciencias por parte de G. Leibnitz y D. Hilbert, hasta el análisis de algunas consecuencias que tendría la resolución de la conjetura $\mathbf{P} \neq \mathbf{NP}$, pasando por la materialización de los primeros modelos formales de computación o la problemática relativa a la definición de las clases de complejidad.

En la segunda sección, se analizan los procesos de cálculo que están inspirados en la Naturaleza y que se enmarcan dentro del campo denominado *Computación Natural*. Se describirán algunos paradigmas y modelos de Computación Natural; en particular, se hará una breve introducción de los *algoritmos genéticos* (inspirados en un mecanismo de selección natural por el cual los individuos mejor dotados, respecto de una cierta medida de *evaluación* de la bondad, son los que sobreviven a lo largo del tiempo); las *redes neuronales artificiales* (inspiradas en la estructura de conexiones que se producen en el cerebro), y la *computación molecular basada en ADN* (cuyo objetivo es el desarrollo de procedimientos sistemáticos a partir de las propiedades computacionales de las moléculas orgánicas).

La última sección de este capítulo está dedicada a la presentación de las primeras ideas y conceptos de la *Computación celular con membranas*, *Membrane Computing*, (modelo de computación paralelo de tipo distribuido y no determinista inspirado en la observación de los procesos que se producen a nivel celular) en cuyo marco se van a desarrollar los trabajos que se recogen en la presente memoria.

1.1. Computabilidad versus Complejidad

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

Es usual encontrar en un diccionario el término *algoritmo* como sinónimo de *cualquier método especial de resolución de un cierto tipo de problemas*. La palabra *algoritmo* debe su nombre al autor persa Abú Jáfar Mohammed ibn al–Khowarizmi, que escribió un texto en el año 825 d.C. en el que recogía una serie de procedimientos mecánicos para el álgebra (en particular una serie de reglas que, secuenciadas en un determinado orden, permitían realizar las operaciones con números decimales). Históricamente, el primer algoritmo no trivial es debido a Euclides que, entre el año 400 y el 300 a.C., describió un procedimiento mecánico para hallar el máximo común divisor de dos números enteros arbitrarios.

La aparición de los primeros métodos formales de razonamiento (posiblemente en Mesopotamia, y mejorados sustancialmente en la civilización griega) proporcionó una herramienta interesante para poder expresar de forma algo

más precisa el concepto intuitivo de *procedimiento sistemático, procedimiento mecánico* o *algoritmo*. Se puede afirmar que el primero en plantear la necesidad de formalizar qué se entiende por procesos (o razonamientos) mecánicos (o automáticos) fue G.W. Leibniz, cuando a finales del siglo XVII formula la necesidad de disponer de un lenguaje universal (*lingua characteristica*) en el que poder expresar cualquier idea, y la necesidad de mecanizar cualquier tipo de razonamiento (*calculus ratiocinator*).

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

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

A principios del siglo XX, los matemáticos estaban a punto de realizar una serie de descubrimientos que marcarían el futuro de las ciencias, en general, y

de las Matemáticas, en particular. Había un convencimiento generalizado de que las Matemáticas podían ser descritas a través de un sistema axiomático, de tal manera que bastaba encontrar el lenguaje, los axiomas y las reglas de inferencia adecuadas para deducir en él los enunciados. Es decir, las Matemáticas vendrían a ser una especie de sistema computacional en el que podría establecerse, mediante un procedimiento mecánico, la veracidad o falsedad de cualquier aserto matemático.

D. Hilbert estaba convencido de que no existía límite para la capacidad de la inteligencia humana. Quizás por ello, hizo pública su ya famosa lista de 23 problemas que propuso en el Congreso Internacional de Matemáticos celebrado en París, en 1900 y, posteriormente, en el Congreso Internacional de Matemáticos celebrado en Bolonia en 1928, formuló tres cuestiones acerca de las Matemáticas que marcarían el devenir de la misma:

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

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

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

Las ideas de Hilbert que subyacen en las cuestiones planteadas, representan la culminación de dos mil años de tradición matemática en donde las referencias obligadas eran el *método axiomático* de Euclides, el proyecto de Leibnitz y Boole de crear una *lógica simbólica* y, más recientemente, los *Principia Matematica* de Russell y Whitehead. Hilbert pretendía clarificar de una vez por todas los métodos del razonamiento matemático: quería diseñar un sistema axiomático formal que abarcara todas las Matemáticas, desde la aritmética elemental hasta el álgebra, pasando por la geometría, el cálculo, etc. Dicho sistema debería cumplir una serie de requisitos indispensables para

los matemáticos, empezando por la *consistencia* y la *completitud* del mismo, y que viene a significar que el sistema sea capaz de probar *la verdad, toda la verdad y nada más que la verdad.*

Para Hilbert, resolver el problema de *decisión* (*Entscheidungsproblem*) relativo a un sistema axiomático o teoría T consistía en hallar un *procedimiento mecánico* que permita decidir, para cualquier aserto formulado en el sistema, si éste puede probarlo o no. Cualquier solución del problema citado se denomina *procedimiento de decisión* del sistema axiomático T. Es interesante observar que si existiera un procedimiento de decisión para las Matemáticas, entonces se podría decidir mecánicamente si cualquier aserto matemático es o no demostrable y, con la consistencia y completitud, si es o no verdadero.

En 1931, el matemático K. Gödel dio una respuesta negativa a la primera de las tres cuestiones formuladas por Hilbert en 1928, considerando un sistema axiomático que contenía la Aritmética elemental y construyendo una proposición verdadera que no se podía probar en dicho sistema. Además, justificó que no era posible dar respuesta a la segunda cuestión en el marco de las propias Matemáticas, demostrando que, si se supone que un sistema sea consistente y que se verifiquen ciertas propiedades *básicas*, entonces el sistema no es capaz de probar una fórmula que exprese su propia consistencia.

1.1.1. Teoría de la Computabilidad

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

de función computable. En 1936, A. Turing utiliza por primera vez el concepto abstracto de *máquina* para formalizar la noción de algoritmo y, por tanto, el concepto de función computable.

En 1936, A. Church formula su famosa tesis acerca de la equivalencia entre la clase de funciones computables, en sentido intuitivo, y la clase de funciones λ-calculables. Esta tesis (también asumida por A. Turing) relaciona dos conceptos de naturaleza completamente distinta: existencia de un procedimiento mecánico que resuelve un problema (concepto informal) y existencia de una máquina de Turing que resuelve un problema (concepto formal). Por tanto, no tiene sentido buscar una prueba matemática de dicha tesis. En dicho año A. Church proporciona el primer ejemplo de un problema para el que no existe procedimiento mecánico (en su modelo) que lo resuelva: el problema de la decidibilidad de la lógica de primer orden, respondiendo negativamente a la tercera cuestión planteada por Hilbert.

Pocos meses después del resultado de Church, A. Turing [116] establece de manera independiente el mismo resultado pero en su modelo de computación, probando que no existía una máquina de Turing que proporcione un procedimiento de decisión para la lógica de primer orden. Lo esencial de la prueba de Turing no radica en la estructura de su modelo computacional, sino en cómo utilizó dicho modelo: de la irresolubilidad algorítmica del *problema de la parada* (dada una máquina de Turing y una configuración inicial, determinar si la máquina para o no), dedujo la indecidibilidad de la lógica de primer orden, usando una técnica de *reducibilidad*.

En el trabajo antes citado, Turing establece la equivalencia de su modelo y el de las funciones λ-calculables, y anuncia la equivalencia entre la clase de funciones computables por máquinas de Turing y la clase de funciones recursivas, que probaría en 1937 [117]. De esta manera se tiene que los tres modelos de computación introducidos formalmente son equivalentes en el sentido de que todo aquello que es calculable en uno de esos modelos lo es en cualquiera de los otros dos. Dado que, según la tesis de Church-Turing, en dichos modelos se pueden hallar todas las funciones que sean computables en un sentido intuitivo, a todos los modelos equivalentes a las máquinas de Turing

se les llama *universales* (o computacionalmente completos).

Con el desarrollo de máquinas computacionales teóricas (antes aún de que la tecnología permitiera su construcción), los investigadores comienzan a centrarse en el estudio de la potencia y limitaciones de dichas máquinas. Estas máquinas teóricas ayudarían de forma decisiva a la construcción de los actuales ordenadores, basados en el modelo conceptual de John von Neumann que, a su vez, está inspirado en los trabajos de Turing acerca de una máquina universal, programable y de propósito general.

Podemos decir que este momento marca, propiamente, el inicio de la *Teoría de la Computación*, cuyo objetivo principal es la clasificación de problemas abstractos de acuerdo con us resolubilidad algorítmica. Como toda teoría interesante, aborda a la vez aspectos positivos (¿qué se puede calcular con los procedimientos mecánicos introducidos formalmente?) y negativos (¿existe algún problema que no se puede resolver de manera mecánica?) y trata cuestiones del siguiente tipo:

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

1.1.2. Teoría de la Complejidad Computacional

Dar un modelo de computación consiste, básicamente en:

- *Definir sintácticamente* cuáles van a ser los procedimientos que serán considerados como mecánicos en el modelo; es decir, dar una

formalización del concepto de algoritmo.

- *Precisar semánticamente* cómo se van a ejecutar dichos procedimientos.

- *Definir* qué se entiende por *resolver* un problema en el modelo (es decir, cuál es el *modo* de computación)

A este respecto, nos referiremos a dos modos de computación especialmente interesantes. El *modo determinista* está caracterizado por el hecho de que cada configuración tiene *a lo sumo* una configuración siguiente; en particular, cada configuración que no sea de parada tiene una *única* configuración siguiente. El *modo no determinista* está caracterizado por el hecho de que cada configuración que no sea de parada tiene *al menos* una configuración siguiente.

Dado cualquier modelo de computación es posible diseñar dispositivos (máquinas) que permitan implementar la ejecución de los procedimientos del modelo. Dichas máquinas pueden ser simples dispositivos teóricos/abstractos o bien dispositivos reales.

Durante la década de los cincuenta se construyen los primeros ordenadores de propósito general al materializarse las primeras implementaciones prácticas de las ideas de J. von Neumann. De esta manera surge la posibilidad de usar estos dispositivos para ayudar a los humanos a resolver problemas.

Desde esta perspectiva, la aparición de nuevos modelos de computación lleva implícito la necesidad de desarrollar dispositivos que, de la manera más fidedigna posible, implemente el modelo a través de máquinas *reales*. Entonces surge la siguiente cuestión: ¿qué problemas pueden ser resueltos en esas máquinas reales? Para responder a esta pregunta es necesario disponer de unas herramientas que permitan cuantificar la mínima cantidad de *recursos* que necesita cualquier solución algorítmica del modelo. De acuerdo con el *principio de invariancia* (el tiempo de ejecución de dos implementaciones de un mismo algoritmo en máquinas distintas difieren en una constante multiplicativa) resulta que el cálculo teórico de los recursos proporciona una medida de la *computabilidad práctica* del problema.

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 explorar el 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 Computacional* con la clasificación de lenguajes y funciones (debidas a J. Hartmanis, P.M. Lewis y R.E. Stearns [59][70]) en función del tiempo y del espacio necesario para su generación o cálculo. Asímismo, 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.

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

El tiempo y el espacio (o memoria) son instancias de recursos computacionales y existe una relación entre ambas medidas. Así por ejemplo, si admitimos que cada paso de una computación permite el acceso a una nueva unidad de espacio/memoria, entonces la cantidad de espacio/memoria utilizada por un algoritmo nunca superará al tiempo de ejecución. No obstante, hay que destacar una diferencia fundamental entre ambas medidas de complejidad: mientras el espacio es reutilizable, el tiempo no lo es.

Problemas de optimización y de decisión

Todos tenemos una idea informal de qué es un *problema*. Ahora bien, ¿qué entendemos por *problema* desde un punto de vista computacional? Para que la resolución de un problema pueda ser abordada por una máquina de cálculo (ordenador) es fundamental que tanto los datos de entrada como los datos de salida puedan ser codificados mediante sucesiones finitas de símbolos (es decir, a través de cadenas o palabras sobre un alfabeto finito).

Informalmente hablando, cuando se trabaja con *problemas de optimización* se trata de encontrar la *mejor* solución (de acuerdo a un cierto criterio) entre una clase de soluciones posibles (candidatos). Es decir, en este tipo de problemas pueden existir muchas eventuales soluciones pero cada una de ellas tiene asociado un valor (un número racional positivo), y se trata de encontrar una solución con valor óptimo (máximo o mínimo).

De manera más formal, diremos que un problema de optimización, X, es una tupla (I_X, s_X, f_X) en donde: (a) I_X es un lenguaje sobre un alfabeto finito; (b) s_X es una función cuyo dominio es I_X y para cada $u \in I_X$, $s_X(u)$ es un subconjunto finito de I_X; y (c) f_X es una función (función objetivo) que asigna a cada instancia $u \in I_X$ y cada $c_u \in s_X(u)$, un número racional positivo $f_X(u, c_u)$.

Los elementos de I_X se denominan *instancias* del problema X. Para cada instancia $u \in I_X$, los elementos del conjunto finito $s_X(u)$ se denominan *soluciones candidatas* asociada a la instancia u del problema. Para cada instancia $u \in I_X$ y cada $c_u \in s_X(u)$, el número racional positivo $f_X(u, c_u)$ es el *valor de la solución* para c_u. La función f_X proporciona el criterio para determinar la *mejor* solución. Una *solución óptima* para una instancia $u \in I_X$ es una solución candidata $c \in s_X(u)$ asociada a dicha instancia tal que, o bien para cada $c' \in s_X(u)$ se tiene que $f_X(u, c) \leq f_X(u, c')$ (c es una *solución minimal* para u), o bien para cada $c' \in s_X(u)$ se tiene que $f_X(u, c) \geq f_X(u, c')$ (c es una *solución maximal* para u).

Una clase importante de problemas de optimización son los *problemas de decisión* que, informalmente, son aquellos que sólo admiten dos respuestas: *sí*

o *no*. Formalmente, un problema de decisión, X, es un par (I_X, θ_X) tal que I_X es un lenguaje sobre un alfabeto finito (cuyos elementos se denominan *instancias*) y θ_X es un predicado (función total booleana) sobre I_X. Por tanto, todo problema de decisión $X = (I_X, \theta_X)$ puede ser considerado como un problema de optimización $X = (I_X, s_X, f_X)$, en donde para cada $u \in I_X$ se tiene que $s_X(u) = \{\theta_X(u)\}$ (la única solución candidata asociada a esa instancia es *sí* o *no*, dependiendo de la respuesta del problema a esa instancia) y $f_X(u, \theta_X(u)) = 1$.

En la vida real, la mayoría de los problemas que surgen son problemas de optimización, Es importante hacer notar que a cada problema de optimización X se le puede asociar un problema de decisión X_D de tal manera que: (a) la construcción de X_D a partir de X se puede realizar *consumiendo* pocos recursos; y (b) conociendo una solución del problema de decisión X_D es posible diseñar otra solución del correspondiente problema de optimización X *consumiendo* pocos recursos en ese proceso.

Por tanto, en toda teoría de la complejidad computacional se trabajará con problemas de decisión, lo que, de acuerdo con lo que acabamos de indicar, no representa una restricción demasiado importante.

Por otra parte, es conviente resaltar que todo problema de decisión $X = (I_X, \theta_X)$ tiene asociado de manera natural un lenguaje L_X sobre el alfabeto de I_X, como sigue: $L_X = \{u \in I_X \mid \theta_X(u) = 1\}$. Recíprocamente, todo lenguaje L sobre un alfabeto Σ tiene asociado un problema de decisión, $X_L = (I_{X_L}, \theta_{X_L})$, en donde $I_{X_L} = \Sigma^*$ y, además, $\theta_{X_L}(u) = 1$ si y sólo si $u \in L$.

El concepto de *resolubilidad de un problema de decisión* se va a definir a través del concepto de *reconocimiento de un lenguaje*. Para ello, recordemos que dado un lenguaje L sobre un alfabeto Σ se dice que una máquina de Turing, M, *reconoce* L si para cada $u \in \Sigma^*$ se verifica que $u \in L$ si y sólo si la máquina M con entrada u *acepta* dicho dato. Así pues, la clave de la definición anterior radica en el concepto de aceptación y rechazo. Si M es una máquina *determinista* (con Σ como alfabeto de trabajo), entonces diremos que M *acepta* $u \in \Sigma^*$ si y sólo si la computación de M con entrada u para y

devuelve *sí*. Si M es *no-determinista*, entonces diremos que M *acepta* $u \in \Sigma^*$ si y sólo si existe, al menos, una computación de M con entrada u que para y devuelve *sí*. Obsérvese que, en el caso no determinista, no se dispone de un método mecánica para **no** aceptar un dato de entrada, ya que pueden existir computaciones de la máquina con entrada u que no paren.

Diremos que una máquina de Turing M resuelve un problema de decisión X si M *reconoce* el lenguaje asociado a X; es decir, para cada instancia u del problema: (1) en el caso determinista, la respuesta del problema es *sí* si y sólo si la máquina (con entrada u) devuelve *sí*; y (2) en el caso no determinista, la respuesta del problema es *sí* si y sólo si existe alguna computación de la máquina (con entrada u) que devuelve *sí*.

Algoritmos óptimos

Si queremos resolver un problema mediante un ordenador electrónico será muy importante disponer de unas herramientas que permitan de alguna manera cuantificar, *a priori*, la cantidad de recursos necesarios para ejecutar una *buena solución* en la máquina. No obstante, podría suceder que el mejor algoritmo conocido que resuelve un cierto problema consuma una importante cantidad de recursos (por ejemplo, de espacio/memoria y/o tiempo). En estas condiciones parece natural plantearse la búsqueda de otros algoritmos que usen estríctamente menos recursos que el mejor conocido y que también resuelvan el problema. De esta manera se plantea una nueva cuestión: *dado un problema resoluble algorítmicamente, hallar el mejor algoritmo que lo resuelva*. El concepto de *mejor solución* estará referido a una cierta *medida de complejidad* que cuantifique los recursos.

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

- Determinar una cota inferior asintótica 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 exacto 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 una cierta analogía con la obtención de problemas irresolubles algorítmicamente: se trata de hallar un algoritmo que satisfaga una propiedad que implica a todos los algoritmos que resuelven dicho problema.

Como es fácilmente imaginable, la tarea de calcular un *algoritmo óptimo* que resuelva un problema suele ser ardua y complicada. Lo más sorprendente es que dicha tarea, a veces, es *imposible* llevarla a cabo. En efecto: si las medidas de complejidad (tiempo, espacio, etc.) que se consideran para cuantificar los recursos satisfacen unos requisitos mínimos (por ejemplo, los *axiomas de Blum* [15]), entonces existe al menos un problema resoluble algorítmicamente que carece de algoritmo óptimo, respecto a dichas medidas, que lo resuelve (*teorema de aceleración*).

Clases de complejidad

La primera consecuencia que se deduce del teorema de aceleración de Blum es la imposibilidad de definir de manera *individual* el concepto de complejidad computacional de un problema, a través de la cantidad de recursos que usa un algoritmo óptimo que lo resuelve. Y ello se debe a que de acuerdo con el teorema citado, existiría al menos un problema que carecería de algoritmo óptimo (respecto de la medida considerada) y, por tanto, no se le podría asignar complejidad alguna de acuerdo con dicha 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 que agrupará a todos aquellos que usen una cantidad de recursos *similar*, en cierto sentido. De esta manera surgen las denominadas *clases de complejidad*.

Los ingredientes necesarios para definir una clase de complejidad son los siguientes:

(a) Un *modelo de computación* que proporcione los dispositivos sobre los que se van a resolver los problemas.

(b) Un *modo de computación* que precise el concepto de *aceptación* de un dato de entrada (y, por tanto, fije el significado de resolución de un problema).

(c) Una *medida de complejidad* que permita cuantificar los recursos usados por los dispositivos computacionales en la resolución de problemas.

(d) Una *función* total computable entre números naturales que sirva de *cota superior* de los recursos usados.

Así por ejemplo, podríamos considerar como modelo de computación las máquinas de Turing, como modo de computación el ordinario o determinista (que precisa la definición de *aceptación* de un dato de entrada, dada anteriormente), como medida de complejidad el tiempo y como cota una cierta familia de funciones distinguidas entre números naturales (por ejemplo, las logarítmicas, las polinomiales o las exponenciales) que actuarían de cotas superiores de los tiempos de ejecución que necesitan los procedimientos mecánicos, en relación con el tamaño de las instancias que describen el problema.

Partiendo del universo formado por todos los problemas resolubles mediante procedimientos mecánicos (que de acuerdo con la tesis de Church–Turing coincidiría con la clase de problemas para los que existe una máquina de Turing que lo resuelve) podríamos obtener las clases **L**, **P** y **EXP** de los problemas para los que existe algún algoritmo que lo resuelve realizando un número de pasos computacionales o *transiciones* que está acotado por alguna

función logarítmica, polinomial o exponencial, respectivamente, en la *longitud* o *tamaño* del dato de entrada.

Puede ocurrir que algún problema sea asignado inicialmente a una clase de complejidad pero que se desconozca si puede o no ser incluido en otra clase contenida estríctamente en la anterior. Por ejemplo, el problema del camino hamiltoniano es un problema que pertenece a la clase **EXP**, pero se desconoce si pertenece o no a la clase **P**.

La *Teoría de la Complejidad Computacional* proporciona herramientas para medir la dificultad de problemas abstractos, a la vez, en términos absolutos (*complejidad intrínseca* de un problema) y en términos comparativos con otros problemas (*clases de complejidad*). El objetivo fundamental de dicha teoría es la clasificación de problemas en función de la *resolubilidad algorítmica práctica* de los mismos. Para ello, se define un concepto de *eficiencia* o resolubilidad práctica que trata de capturar la idea intuitiva de resolubilidad a través de ordenadores que existen actualmente. Un *algoritmo* se dirá *eficiente* si la cantidad de recursos necesarios para su ejecución, en el caso peor, está acotada por un polinomio en el tamaño del dato de entrada. De esta manera se fija una frontera entre la resolubilidad algorítmica práctica (*tratabilidad*) y la no resolubilidad algorítmica práctica (*intratabilidad*).

¿Por qué se consideran a las funciones polinómicas para establecer dicha línea? Básicamente porque esa clase de funciones es estable por ciertas operaciones importantes (suma, producto y composición de funciones) y porque el crecimiento de las funciones polinómicas suele ser relativamente *lento*.

Así pues, los problemas se clasifican en *tratables* e *intratables*, según sean o no resolubles de forma eficiente. De acuerdo con lo expresado anteriormente, la clase de complejidad de los problemas tratables es, precisamente, la clase **P**. Los problemas *computacionalmente intratables* serán aquellos que no se pueden resolver algorítmicamente en tiempo polinomial; es decir, que no pueden ser resueltos por máquinas reales para instancias de tamaño razonablemente grandes (por ejemplo, existen problemas de la clase **EXP** que son intratables).

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.

La clase NP

Podríamos extender el concepto de procedimiento mecánico admitiendo la posibilidad de que su ejecución sea *no determinista* en el siguiente sentido: en cada instante de una computación existe un conjunto de instrucciones que son *ejecutables* de tal manera que el procedimiento puede seleccionar cualquiera de ellas para proseguir su ejecución. Así pues, en un cierto paso computacional no determinista puede suceder que una configuración posea más de una configuración siguiente y, en consecuencia, a partir de un cierto dato de entrada de un problema, un procedimiento mecánico no determinista puede proporcionar, de manera independiente, muchas computaciones distintas. El coste en tiempo de un procedimiento no determinista se define como sigue: a cada instancia de tamaño n se le asocia la longitud de la menor computación de aceptación (si no existe, se le asocia el valor 0); entonces, el coste en tiempo es una función que a cada n le asocia el máximo de los valores asignados a cada instancia de tamaño n de acuerdo con el criterio anterior.

La clase **NP** está formada por todos aquellos problemas que se pueden resolver por algoritmos no deterministas cuyo tiempo de ejecución está acotado por un polinomio; es decir, problemas cuyas posibles soluciones pueden ser chequeadas en tiempo polinomial a fin de decidir si realmente son o no soluciones correctas. Así pues, los problemas de la clase **NP** serían resolubles en tiempo polinomial mediante máquinas que tuvieran la capacidad de realizar en paralelo y de manera independiente, un número no acotado de computaciones. Por tanto, esta clase jugaría el papel de clase de problemas *tratables en modo no determinista*.

Es obvio que todo algoritmo *ordinario* puede ser considerado como no determinista y, por tanto, se verifica la inclusión $\mathbf{P} \subseteq \mathbf{NP}$. Ahora bien, desde el punto de vista de la eficiencia computacional ¿añade algo realmente nuevo el

modo no determinista respecto del modo ordinario, que podríamos denominar determinista? Es decir, ¿es estricta la inclusión $\mathbf{P} \subseteq \mathbf{NP}$? La respuesta no se conoce hoy día y, sin lugar a dudas, es una de las cuestiones abiertas más importantes a las que se enfrenta la ciencia del siglo XXI. El último apartado de esta sección está dedicado a analizar algunas consecuencias que se deducirían de una respuesta (afirmativa o negativa) a dicha cuestión.

El Instituto de Matemáticas Clay, CMI, de Cambridge, Massachusetts, ha seleccionado siete problemas que son considerados como especialmente relevantes, y la resolución de cada uno de ellos tiene asignado un premio de un millón de dólares. El primero de ellos es el problema **P** *versus* **NP**; es decir, determinar si las clases **P** y **NP** coinciden.

Dentro de la clase **NP** podemos destacar una subclase de problemas que tienen especial interés: los problemas que son los más *difíciles* de la clase, en el sentido de que cualquier otro problema de la clase **NP** puede ser resuelto a través de él con un coste en tiempo adicional de tipo polinomial (mediante una *reducción en tiempo polinomial*). Es la clase de los problemas denominados **NP**–*completos*.

El interés de la clase de los problemas **NP**–completos (que notaremos **NPC**) radica principalmente en el hecho de que dichos problemas son candidatos idóneos para atacar la cuestión $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$. En efecto, es fácil probar que si existe *un* problema **NP**-completo que es tratable (es decir, que pertenece a la clase **P**), entonces la respuesta a la cuestión es afirmativa (es decir, *todo* problema de la clase **NP** pertenece a la clase **P**); y si existe *un* problema **NP**-completo que no es tratable, entonces la respuesta es negativa (y, además, resulta que *ningún* problema **NP**–completo pertenece a la clase **P**). Con otras palabras, las clases de complejidad **P** y **NPC** o bien son iguales o son disjuntas.

En 1971, S.A. Cook [31] 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 reducibilidad en tiempo polinomial permite generar nuevos problemas **NP**-completos a partir de otros conocidos, R.M. Karp [68] da 24 ejemplos nuevos de problemas **NP**-completos. Entre

ellos, 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. Karp introdujo las notaciones **P** y **NP** que ahora son consideradas estándares, y redefinió el concepto de **NP**–completitud en los términos antes descritos.

En la actualidad se conocen muchos problemas **NP**-completos de disciplinas tan diversas como lógica, teoría de números, diseño de circuitos, telecomunicaciones, teoría de grafos, economía, investigación operativa, etc. El libro [43] de M.R. Garey y D.S. Johnson constituye un catálogo exhaustivo de más de trescientos problemas **NP**-completos.

¿Y si P = NP?

La *Teoría de la Complejidad Computacional* juega un papel fundamental en la *criptografía* moderna (*public–key cryptography*). Actualmente en Internet existe una gran información de tipo confidencial, se realizan numerosas transacciones comerciales que mueven una cantidad ingente de dinero, etc. La seguridad en la red depende básicamente de la complejidad computacional inherente a problemas como el de la factorización entera o la desencriptación de cadenas codificadas por el sistema DES (Data Encryption Standard), sistema que encripta textos planos a través de 64 bits usando 56 símbolos claves.

Un ataque convencional sobre un texto encriptado por DES realizado mediante búsqueda exhaustiva, a través de un ordenador que es capaz de realizar un millón de operaciones por segundo, tardaría unos mil años aproximadamente. Recientemente, D. Boneh, Ch. Dunworth y R.J. Lipton [16] estiman que una encriptación DES puede ser decodificada por un ordenador molecular en unos cuatro meses.

Si resultase que **P** fuese igual a **NP**, entonces resultaría que un algoritmo cuadrático que resolviera una variante simple del problema **SAT** se podría usar para factorizar los números de 200 dígitos en algunos minutos.

Así pues, una respuesta afirmativa a la cuestión $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ tendría unas

consecuencias funestas para la criptografía. No obstante, también tendría sus consecuencias positivas. Por ejemplo, sería posible diseñar programas que permitieran a un ordenador electrónico convencional encontrar demostraciones de teoremas que tengan pruebas de *longitud razonable* (ya que las pruebas formales pueden ser *reconocidas* en tiempo polinomial). Desgraciadamente, ocurriría que muchas de las pruebas no sería entendidas por los humanos. Pero bueno, entre otras consecuencias *positivas*, conseguiríamos los siete millones de dólares que garantizan los premios CMI (con tal de guardar cuidadosamente la prueba de que $\mathbf{P} = \mathbf{NP}$ durante el tiempo necesario para obtener las soluciones de los siete problemas). Lo que no estaría nada mal.

En el caso en que la respuesta a la cuestión citada fuese negativa, podría suceder que todo problema **NP**–completo tuviese un algoritmo determinista de coste polinomial que trabaje *correctamente* sobre *muchas* entradas del problema. Con ello, el mundo de la criptografía no se tambalearía y, en cambio, se conseguiría algunos beneficios parciales del caso en que la respuesta fuese afirmativa.

En esta línea, L. Levin [69] y R. Impagliazzo [64] desarrollan una teoría de la completitud del caso promedio, en donde la cuestión $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ es sustituida por esta otra: determinar si cada problema **NP**–completo puede ser resuelto, con una *razonable* distribución de probabilidad sobre sus entradas en tiempo polinomial en el caso promedio.

Bajo el supuesto de que $\mathbf{P} \neq \mathbf{NP}$, R.E. Ladner (1975) ha demostrado la existencia de *problemas intermedios*; es decir, de problemas intratables que no son **NP**–completos.

A la hora de enfrentarnos a un problema computacionalmente difícil/duro, es interesante tener presente algunas consideraciones:

- Preguntarnos en qué aspecto del problema radica la razón de la dificultad.
- Intentar buscar una solución aproximada más simple en lugar de una solución exacta del problema.
- Tener presente que algunos problemas sólo son difíciles en el caso peor

(que se podría dar poco) y, en cambio, son fáciles en los restantes casos. Así se podría obtener un procedimiento mecánico ocasionalmente lento pero que muchas veces es rápido.

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

1.2. 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 provoca un crecimiento exponencial en la otra. Por ello, surge la necesidad de buscar nuevos modelos que sean capaces de reducir ambos parámetros o, al menos, de incluir procedimientos en los que un coste alto en una de las medidas sea *asimilado*, en cierto sentido, por el propio modelo en beneficio de una reducción considerable sobre la otra.

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

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

La *Computación Natural* surge como una de las posibles alternativas a la computación que podríamos denominar clásica, en la búsqueda de nuevos paradigmas que puedan proporcionar una solución *efectiva* a las limitaciones que poseen los modelos convencionales. Actualmente, dentro del campo de Computación Natural se engloba un conjunto de modelos que tienen como característica común la simulación del modo en que la naturaleza actúa/opera

sobre la materia (hay quien extiende este concepto hasta abarcar modelos tales como la *computación cuántica*, que no se ajusta fielmente a la interpretación anterior). Es decir, esos modelos estudian 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. Esta simulación que aborda la Computación Natural puede tener distintas interpretaciones a la hora de describir los nuevos modelos: que se utilice para el diseño de nuevos esquemas algorítmicos usando técnicas inspiradas en la naturaleza, o bien que sugiera la creación física de nuevos modelos experimentales en los que el medio electrónico de los ordenadores convencionales se sustituya por otro sustrato que pueda implementar ciertos procesos que aparecen en el modo de operar de la naturaleza.

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

Como ejemplo de la segunda interpretación, a finales de la década de los cincuenta el premio nobel R.P. Feynman [39] 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 [60] propone explícitamente el primer modelo teórico molecular basándose en las propiedades de la molécula de ADN. En noviembre de 1994, L. Adleman [2] 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. Entre las áreas que se enmarcan dentro de la Computación Natural, destacamos las siguientes:

1. Los *algoritmos genéticos* (o más en general, la *computación evolutiva*), introducidos por J. Holland [63] en 1975, que hacen uso de algunas operaciones inspiradas en la evolución y en la selección natural a fin de encontrar una *buena* solución a partir de una gran cantidad de posibles soluciones candidatas.

2. Las *redes neuronales artificiales*, introducidas por W.S. McCulloch y W. Pitts [78] en 1943, que están inspiradas en las interconexiones y en el funcionamiento de las neuronas en el cerebro.

3. La *computación molecular*, cuyo objetivo consiste en usar moléculas orgánicas (ADN, ARN, proteínas, etc.) como hardware biológico que permite realizar computaciones. Esta disciplina nace a finales de 1994 con los trabajos de L. Adleman [2], si bien tiene precedentes en un trabajo de T. Head [60] en el que formula el *sistema splicing* (modelo teórico de procesamiento de moléculas de ADN con la participación de enzimas).

4. La *Computación celular con membranas*, introducida por Gh. Păun [85] en 1998, que está inspirada en la estructura y el funcionamiento de las células de los organismos vivos, en cuanto a su capacidad para procesar y generar información.

Los algoritmos genéticos y las redes neuronales artificiales han sido implementadas a través de programas en ordenadores electrónicos convencionales. La computación molecular basada en ADN ha sido implementada en medios bioquímicos (el experimento de L. Adleman permitió resolver una instancia concreta del problema del camino hamiltoniano, en su versión dirigida y con dos nodos distinguidos, a través de la manipulación de moléculas de ADN en el laboratorio). En cambio, la Computación celular con membranas aún no ha sido implementada ni en medios electrónicos ni bioquímicos.

En octubre de 2003, el Institute for Scientific Information (ISI, USA), ha designado a la *Computación celular con membranas* como *Fast Emerging Research Front* en el área de *Computer Science*.

A continuación pasamos a estudiar brevemente algunas cuestiones relativas a los principales modelos de Computación Natural.

1.2.1. Algoritmos Genéticos

Los algoritmo genéticos fueron introducidos por John Holland en 1975 inspirándose en el proceso observado en la evolución natural de los seres vivos.

Se puede interpretar que, por medio de las características propias del material genético de que disponen los seres vivos, cada individuo resuelve cada día el problema de la supervivencia. Para ello, a nivel genético, la solución consiste en buscar aquellas adaptaciones beneficiosas en un medio hostil y cambiante. Debido en parte a la selección natural, cada especie adquiere un cierto *conocimiento* que es incorporado a la información de sus cromosomas.

Por tanto, podemos localizar la evolución de los seres vivos en los cromosomas, donde está codificada la información relevante del mismo. Esta información es modificada de unas generaciones a otras, combinando la información presente en sus progenitores.

Aunque el método que se sigue en esta selección no es todavía bien conocido, los siguientes principios son aceptados unánimemente:

- La evolución se da a nivel cromosómico, no a nivel de individuo.
- Llamamos *selección natural* al proceso por el que los individuos con cromosomas *mejores* tienen mayor probabilidad de reproducirse.
- La evolución tiene lugar por medio de la combinación (*recombinación*) de los cromosomas de los progenitores.
- En la evolución biológica se pierde la *historia* de la evolución, en el sentido de que en cada nueva generación tan sólo puede considerarse la información presente en sus progenitores, pero no de ascendentes anteriores.

Los algoritmos genéticos establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena a modo de cromosoma. En palabras del propio Holland: *se pueden encontrar soluciones aproximadas a problemas de gran complejidad computacional mediante un proceso de evolución simulada.*

A tal efecto se introduce una medida de evaluación (que se denomina *fitness*) basada en la función objetivo del problema. Además, se introduce un mecanismo de selección de manera que los individuos con mejor evaluación tendrán mayor probabilidad de recombinarse para producir la siguiente generación.

Por tanto, para aproximar la solución de un problema por medio de un algoritmo genético es necesario definir los siguientes elementos:

- Una representación cromosómica.
- Una población inicial.
- Una medida de evaluación.
- Un criterio de selección y eliminación de cromosomas.
- Una (o varias) operaciones de recombinación.
- Una (o varias) operaciones de mutación.

Originalmente, las soluciones fueron representadas por medio de cadenas binarias, es decir, listas de 1's y 0's. Este tipo de representaciones ha sido ampliamente utilizada incluso en problemas donde no es muy natural. En 1985, De Jong plantea la siguiente cuestión: ¿Qué se debe hacer cuando los elementos del espacio de búsqueda se representan de modo natural por estructuras complejas como matrices, árboles o grafos?, ¿Se debe intentar linealizar en una cadena o trabajar directamente con estas estructuras?

Conviene tener presente que, debido a que las operaciones genéticas dependen del tipo de representación elegida, en función de la respuesta que se ofrezca a estas cuestiones, todo el mecanismo diseñado ha de ser modificado.

La ventaja de la representación lineal radica en que la definición de las operaciones de recombinación es sencilla, y los resultados sobre convergencia están probados para el caso de cadenas binarias. Sin embargo en algunos problemas puede ser poco natural y eficiente el utilizarlas (por ejemplo, problemas relacionados con búsquedas de soluciones sobre grafos).

La popularidad de los algoritmos genéticos se debe en parte a que la evolución es un método robusto y bien probado dentro de los sistemas biológicos naturales. Además son fácilmente paralelizables, lo que supone una ventaja gracias al abaratamiento actual de los costes en hardware. Por otra parte, se pueden realizar búsquedas en espacios de hipótesis que contienen complejas interacciones entre las distintas partes, donde el impacto de cada parte sobre la función de evaluación es difícil de especificar.

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

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

1.2.2. Redes Neuronales Artificiales

El cerebro animal está continuamente recibiendo información de muchas fuentes por medio de señales de entrada y las procesa para generar una respuesta de salida. Está formado por millones de neuronas interconectadas formando lo que podría denominarse una *Red Neuronal*.

Las neuronas son las células que forman la corteza cerebral de los animales, cada una está compuesta por un cuerpo neuronal, del que parten dos tipos de prolongaciones: las *dendritas*, que son pequeñas y ramificadas, y el *axón*, que es un tubo muy largo que se ramifica en su extremo distal dando pequeños bulbos finales que casi tocan las dendritas de las neuronas adyacentes. La pequeña separación entre los bulbos finales y las dendritas se denomina *sinapsis*.

El funcionamiento de las neuronas se realiza a través de impulsos eléctricos y reacciones químicas. Los impulsos eléctricos, que utiliza una neurona para intercambiar información con las demás, viajan por el axón que hace contacto con las dendritas de la neurona vecina mediante las sinapsis.

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

El primer modelo de red neuronal fue propuesto en 1943 por W.S. McCulloch y W. Pitts [78] como simulación de la actividad nerviosa, y, poco después, sirvió de ejemplo para los modelos posteriores de M. Minsky [81] y F. Rosenblatt [107], entre otros.

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

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

Una vez construida la red neuronal, se obtiene un modelo *a la medida*

que actúa en función de lo que percibe. Si existe una correspondencia de causa–efecto entre las descripciones introducidas y los valores a prever, el modelo extraerá dicha relación para aplicarla en los casos sucesivos. Además, el modelo obtenido es robusto, en el siguiente sentido: la aparición de ejemplos no coherentes en la base de ejemplos es reconocida por la red y no influyen en las conclusiones extraídas.

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

- *Modelos de tipo biológico*, que tratan de simular los sistemas neuronales biológicos (y, más recientemente, funciones auditivas y algunas funciones básicas de la visión). Su objetivo principal es desarrollar un modelo para verificar hipótesis relativas a sistemas biológicos.
- *Modelos dirigidos a la aplicación*, que, en general, no presentan similitud con los sistemas biológicos sino que se diseñan atendiendo a las necesidades del problema abstracto que pretenden resolver.

La gran diferencia que hace singulares a las redes neuronales en relación con otras ramas de la Computación radica en que no son algorítmicas, esto es no se programan haciéndoles seguir una secuencia predefinida de instrucciones. Las redes neuronales generan ellas mismas sus propias *reglas*, para asociar la respuesta a su entrada; es decir, aprenden por ejemplos y de sus propios errores.

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

1.2.3. Computación Molecular

Como ya se ha comentado, a finales de la década de los cincuenta, el premio nobel R.P. Feynman [39] describe los ordenadores *sub–microscópicos* e

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

En 1987, T. Head [60] propone el primer modelo computacional abstracto basado en la manipulación de las moléculas de ADN, *el modelo splicing*. En este modelo la información es almacenada en cadenas de caracteres al modo en que lo hacen las moléculas de ADN, y las operaciones que se pueden realizar sobre dichas cadenas son similares a las que realizan ciertas enzimas sobre el ADN.

L.M. Adleman materializa esta similitud en noviembre de 1994 [2] mostrando que es posible usar procesos biológicos para atacar la resolubilidad de instancias de ciertos problemas matemáticos especialmente *difíciles*: mediante un experimento realizado en el laboratorio consiguió resolver una instancia concreta de un problema *computacionalmente intratable* usando técnicas de biología molecular para la manipulación del ADN. Aunque el experimento de Adleman no es propiamente una implementación práctica del modelo que diseñó T. Head, el tipo de sustrato utilizado (moléculas de ADN) así como las operaciones que usa sobre dicho sustrato son similares a las propuestas por el modelo splicing.

En julio de 2000, un equipo de científicos de la Universidad de California desarrolló un interruptor del tamaño de una millonésima de milímetro (un nanómetro), a partir de una molécula. Todo parece indicar que este interruptor

puede representar una alternativa revolucionaria en relación con los actuales chips de silicio.

- En su funcionamiento sustituye la electricidad por una reacción química, lo que representa un importante ahorro en el consumo de energía.

- Estos nuevos interruptores podrían disponer de más de mil procesadores en el espacio ocupado hoy día por un sólo procesador (los actuales chips de silicio tienen una altura aproximada de cinco mil nanómetros).

- Se estima que estos interruptores podrían aumentar la velocidad de procesamiento de la información cien mil millones de veces la de un ordenador convencional, y podrían reproducir la capacidad equivalente a cien ordenadores convencionales en el tamaño de un grano de sal fina.

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

Los modelos moleculares constituyen una rama de la Computación en la que intervienen áreas del conocimiento humano muy distintas y, tradicionalmente, disjuntas, que van desde las disciplinas más abstractas de las matemáticas hasta las aplicaciones más novedosas que se puedan obtener en los laboratorios de biología molecular. Esta característica hace que muchos de los avances en la creación de modelos eficientes pase ineludiblemente por la consecución de nuevas, eficientes y robustas técnicas de manipulación de las moléculas con las que se trabaja en el laboratorio.

1.3. Computación celular con membranas

Hasta ahora hemos hablado de modelos de Computación Natural en los que se simula el modo en que la naturaleza *calcula* a un nivel genético (algoritmos genéticos y computación molecular basada en ADN), o a un nivel neuronal (redes neuronales). Un estudio más detallado del funcionamiento de los organismos vivos nos sugiere un nuevo nivel de computación no analizado hasta ahora: *el nivel celular*.

La célula es la unidad fundamental de todo organismo vivo. Posee una estructura compleja y, a la vez, muy organizada que permite la ejecución simultánea de un gran número de reacciones químicas.

Existen dos tipos de células: las *procarióticas* (propias de ciertos organismos unicelulares, como las bacterias y las cianofíceas) que carecen de membrana nuclear y por tanto no hay separación entre núcleo y citoplasma, y las células *eucarióticas* (propias de los animales y plantas) cuyo núcleo está separado del citoplasma por una doble membrana. En ambos tipos de células se realizan de manera similar una serie de procesos que son esenciales para la vida, tales como la replicación del ADN, la captación y utilización de energía, la síntesis de proteínas y otros muchos procesos metabólicos.

En un primer análisis se pueden distinguir en una célula tres partes claramente diferenciadas: una especie de película muy fina (*membrana plásmática*) que delimita a la célula de su entorno; un corpúsculo central (*núcleo*), que contiene y almacena la información genética en moléculas de ADN y de ARN; y el *citoplasma*, que es la parte comprendida entre el núcleo y la membrana plasmática.

En el citoplasma de las células eucarióticas existen diferentes componentes fundamentales: *las mitocondrias*, encargadas de la generación de moléculas que encierran energía utilizable en los procesos metabólicos; el *aparato de Golgi*, encargado del transporte intracelular de distintas sustancias; los *ribosomas*, que vienen a ser una fábrica de proteinas y desempeñan un papel esencial en el metabolismo celular; el *retículo endoplásmico*, que es una red de

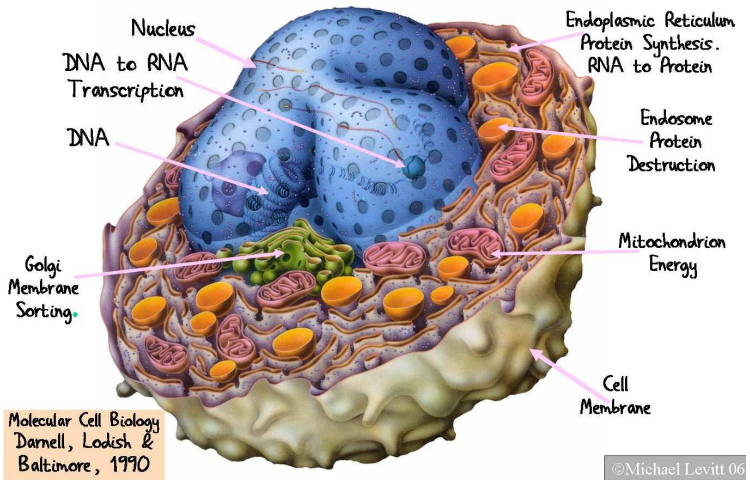


Figura 1.1: La célula eucariótica

membranas interconectadas estructurada en dos partes: una que forma parte de la membrana nuclear y que que facilita el paso del ARN mensajero del núcleo hacia el citoplasma, y otra que se encarga de la comunicación entre los distintos componentes de la célula; y, finalmente, los *lisosomas*, que son vesículas rodeadas de una única membrana que contiene enzimas y se encargan de digerir sustancias que proceden del exterior así como degradar restos de los componentes que ya no son útiles para la célula.

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 sustancias químicas que la propia célula procesa.

Debido a que los procesos que se producen en una célula son de una gran complejidad, resulta 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. Se pretende crear un modelo de computación abstracto que simule, de forma simplificada aunque

lo 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, algunas veces de forma selectiva e incluso en una sola de las direcciones.

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

1.3.1. Las membranas biológicas

Las *membranas biológicas* son estructuras dinámicas básicas para la célula y juegan un papel esencial a la hora de definir el fenómeno que usualmente denominamos como *vida*.

Una membrana (denominada *plasmática*) separa el espacio interno de la célula protegiéndolo de su entorno. Las restantes membranas (denominadas *inter-*

nas) proporcionan la estructura jerarquizada propia de la célula (que puede ser formalizada a través de un *árbol enraizado*) y ofrecen la adecuada protección al núcleo, que es el depositario de la información genética. Las membranas están involucradas de manera decisiva en la mayoría de reacciones químicas que tienen lugar en los compartimentos celulares y pueden considerarse como barreras semipermeables que, o bien permiten el paso de algunas sustancias químicas en cualquiera de los dos sentidos (dentro–fuera o fuera–dentro), o bien impiden el paso de sustancias. Como ya se ha comentado, las membranas se comportan como canales selectivos de comunicación para la transferencia de compuestos químicos entre distintos compartimentos, así como entre la propia célula y su entorno, controlando así un cierto flujo de datos (es decir, de información).

El premio Nobel de química del año 2003 ha sido otorgado a los científicos P. Agre y R. MacKinnon por sus descubrimientos relacionados con los canales de las membranas celulares, que han supuesto un gran avance en los estudios de la química celular e ilustran su importancia en los procesos vitales.

Cada compartimento celular (delimitado y separado del resto por una membrana) puede considerarse como una unidad computacional (un *procesador*) con sus propios datos (compuestos químicos) y su propio programa local (reacciones químicas), de tal manera que el conjunto de compartimentos, considerado como una unidad global (la célula), puede ser interpretado como un *modelo de computación no convencional*.

A lo largo de la breve historia de la teoría de la computación, muchos avances, tanto en el plano teórico como en el de aplicaciones prácticas, se han producido bajo la inspiración de los procesos que se dan en la naturaleza. ¿Qué puede proporcionar la célula como fuente de inspiración computacional? ¿Se puede abstraer una especie de modelo de computación a partir de las células vivas? ¿Es posible implementar computaciones a través de las células, con la esperanza de disponer algún día de un *computador celular*, de propósito general?

1.3.2. Sistemas P

En las ideas originales de Gh. Păun, los sistemas celulares con membranas no fueron introducidos propiamente para modelizar completamente la estructura y funcionamiento de una célula, sino más bien para analizar algunos hechos computacionalmente relevantes que pueden ser abstraídos de las mismas.

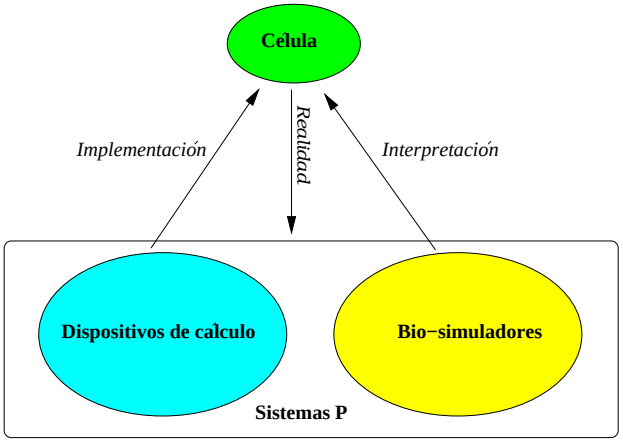


Figura 1.2: Sistemas celulares

Los *sistemas P* (dispositivos computacionales de los modelos celulares) constituyen un modelo teórico de computación inspirado en la realidad de las células vivas. No obstante, existe la esperanza de que la computación celular con membranas proporcione un marco de simulaciones de fenómenos moleculares. En este contexto, se está trabajando en el desarrollo del hardware y simulaciones biológicas. Así el proyecto E-CELL, iniciado en 1996, tiene como objetivo la modelización y simulación de procesos celulares tales como trayectorias metabólicas, síntesis de proteínas y transporte a través de las membranas, con el fin de predecir el comportamiento de las células vivas.

Uno de los retos más importantes planteado en este nuevo modelo consiste

en encontrar una posible implementación biológica del mismo.

La *Computación celular con membranas* ha sido hasta ahora el último modelo de Computación Natural. Fue introducido por Gh. Păun en octubre 1998 [85] como un modelo de tipo distribuido y paralelo, y está inspirado en el funcionamiento de la célula como organismo vivo capaz de procesar y generar información.

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

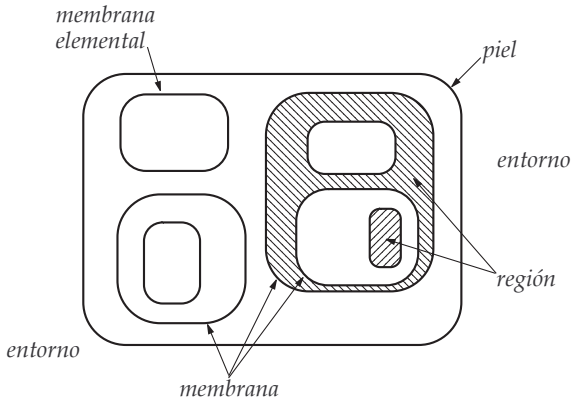


Figura 1.3: Estructura de membranas

En su trabajo fundacional [85], Gh. Păun introdujo los sistemas P de transición de grado $q \geq 1$ de la siguiente manera: es una tupla

$$\Pi = \big(\Gamma, \mu_{\Pi}, \mathcal{M}_1, \ldots, \mathcal{M}_q, (R_1, \rho_1), \ldots, (R_q, \rho_q), i_0\big)$$

en donde:

- Γ es un alfabeto finito (*alfabeto de trabajo*).
- μ_{Π} es una *estructura de membranas* (que consta de q membranas). Las membranas de μ_{Π} están etiquetadas de forma unívoca usando los números naturales desde 1 hasta p.
- $\mathcal{M}_i$ $(1 \leq i \leq q)$ es un multiconjunto finito sobre Γ asociado a la membrana i del sistema, representando el contenido inicial.
- R_i $(1 \leq i \leq q)$ es un conjunto finito de *reglas de evolución* de transición asociado a la membrana i del sistema. Una regla de evolución de transición es un par (u, v), habitualmente representado $u \to v$, donde u es una cadena sobre Γ y $v = v'$ o $v = v'\delta$, siendo v' una cadena sobre $\Gamma \times \big(\{here, out\} \cup \{in_i : i = 1, \ldots, q\}\big)$.
- ρ_i $(1 \leq i \leq q)$ es un *orden parcial estricto* sobre R_i, que establece una relación de prioridad entre las reglas de R_i.
- i_0 es un número natural entre 1 y q (*membrana de salida* del sistema).

Para describir de manera informal la semántica del modelo se introduce el concepto de configuración, de donde seguirá, una vez establecida la forma en que se aplican las reglas de transición, la noción de computación del sistema.

Una *configuración* de un sistema P consta de una estructura de membranas y una familia de multiconjuntos de objetos asociados a cada región. En la descripción de un tal sistema, aparece siempre una configuración denominada *inicial*. En algunas variantes existe una *membrana de entrada* que permite añadir un multiconjunto de objetos antes de que se produzcan evoluciones. Es decir, una *configuración* de Π es una tupla $(\mu, M_{i_1}, \ldots, M_{i_t})$ tal que μ es

la estructura de membranas que se obtiene de μ_Π al eliminar las membranas distintas de i_1 a i_t (la membrana piel no se puede eliminar, por lo que coincide en ambas estructuras); y M_{i_j} es un multiconjunto finito sobre Γ, para cada $j = 1, \ldots, t$.

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

La *ejecución de una regla* $u \to v$ asociada a una membrana i presente en una configuración se realiza como sigue: los objetos en u se eliminan de la membrana i (ésta debe contener, por tanto, suficientes objetos para que la regla se pueda aplicar); entonces, para cada $(a, out) \in v$ un objeto $a \in \Gamma$ se incluye en la membrana padre de la membrana i (o abandona el sistema si la membrana i es la piel); para cada $(a, here) \in v$ un objeto $a \in \Gamma$ se añade a la membrana i; para cada $(a, in_j) \in v$ un objeto $a \in \Gamma$ se introduce en la membrana j (teniendo en cuenta que si la membrana j no es hija de la membrana i, entonces la regla no se puede aplicar); finalmente, si $\delta \in v$, entonces la membrana i se disuelve; es decir, se elimina de la estructura de membranas, pasando sus objetos a su membrana padre y desapareciendo las reglas de evolución y relaciones de prioridad asociadas a ella (la membrana piel no se puede disolver).

Las reglas serán aplicadas de manera *no determinista, paralela y maximal*; es decir, las reglas a ser usadas y los objetos involucrados en la misma serán escogidos de tal manera que en cada paso de computación todos los objetos que *puedan* evolucionar *tienen* que evolucionar. Los objetos pueden pasar de una región a otra a través de las membranas y éstas pueden ser disueltas, creadas o divididas. Así se produce una *transición* de una configuración a otra.

Dada una regla $u \to v$, la longitud de la cadena u se denomina *radio* de la regla. Se dice que un sistema celular es *cooperativo* si posee, al menos, una regla de evolución de radio mayor que uno.

Las *relaciones de prioridad* dadas sobre las reglas se interpretarán como sigue: si la regla r_1 tiene mayor prioridad que la regla r_2 y se puede aplicar r_1, entonces no se aplicará r_2, aunque fuera factible. 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.

Dadas dos configuraciones, C y C', de Π, diremos que C' se obtiene de C en un *paso de transición*, y notaremos $C \Rightarrow_{\Pi} C'$, si C' es el resultado de aplicar a C, en *paralelo* y para todas las membranas al mismo tiempo, las reglas de evolución contenidas en un determinado (multi)conjunto de reglas asociadas a las membranas que aparecen en la estructura de membranas de C. En dicho (multi)conjunto se indica el número de veces que se aplica cada una de las reglas y debe ser *maximal*, en el sentido de que, tras la aplicación de las reglas, en ninguna membrana permanezcan objetos sin evolucionar que puedan activar alguna de las reglas asociada a la membrana. Puesto que para una configuración usualmente existe más de un (multi)conjunto de reglas aplicable, los pasos de transición se realizan de manera no determinista.

Una *computación* en un sistema P es una sucesión (finita o no) de configuraciones tal que la primera de ellas es una configuración inicial y toda configuración de la sucesión que no sea la inicial se obtiene de la anterior mediante un paso de transición. Es decir, una *computación* $\mathcal{C}$ de Π es una sucesión (finita o no) de configuraciones, $\{C^i\}_{i<r}$, tal que C^0 es la configuración inicial de Π; $C^i \Rightarrow_{\Pi} C^{i+1}$, para todo $i < r-1$; y, o bien r es un número natural no nulo y no existe una regla que se pueda aplicar en ninguna de las membranas de C^{r-1} (en cuyo caso se dice que $\mathcal{C}$ es de parada, ha realizado $r-1$ pasos y C^{r-1} es su *configuración de parada*), o bien $r = \infty$ (en cuyo caso se dice que $\mathcal{C}$ no es de parada). Diremos que una *computación* $\mathcal{C}$ es *exitosa* si es una computación de parada y, además, la membrana de salida i_0 aparece en C^{r-1} como membrana elemental. A cada computación exitosa de Π, se le puede asociar una salida (codificada por el multiconjunto asociado a la membrana de salida del sistema en su configuración de parada). Entonces, un sistema P de transición es una máquina que genera el conjunto de números naturales formado por las salidas de todas las computaciones exitosas de dicho sistema.

Así pues, una computación en un sistema P parte de una configuración y mediante sucesivas transiciones, o bien se llega a una situación de parada (en tanto que el sistema no puede seguir evolucionando), o bien, el sistema nunca

se detiene (en tanto que sigue evolucionando de manera indefinida).

Admitiremos que toda computación ejecuta un proceso sincronizado; es decir, se supone que hay una especie de *reloj universal* que marca las actuaciones de todos los elementos que integran el sistema celular. Hay que destacar la existencia de dos niveles de paralelismo en la ejecución: por una parte, las reglas asociadas a una membrana son aplicadas de manera simultánea; y, por otra, estas operaciones son realizadas en el mismo instante en todas las membranas que vertebran el sistema. Es decir, en un paso de reloj se ejecutarán todas las reglas que pueden ser aplicadas (de manera maximal), y en todas las membranas.

Los sistemas celulares con membranas constituyen un modelo de computación que conjuga la elegancia y sencillez del modelo, con su proximidad a la realidad de la célula, desde un punto de vista biológico, abstrayendo de la estructura y funcionamiento de las células una serie de ideas computacionalmente relevantes. Son modelos matemáticos con propiedades atractivas desde el punto de vista computacional: algunos sistemas celulares son *completos* (tienen la misma potencia computacional que las máquinas de Turing) y *eficientes* (capaces de proporcionar soluciones polinomiales de problemas **NP**–completos).

Aunque los sistemas celulares tienen una inspiración biológica, se debe resaltar el hecho de que constituyen 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 de ordenadores como Internet puede ser representada como una estructura de membranas, en la que los nodos de la red se interpretan como las membranas que forman el sistema, y el flujo de información entre nodos conectados se interpreta como las componentes químicas que dichas membranas generan e intercambian.

Asimismo, sistemas dinámicos complejos, como aquellos que surgen en el estudio de la dinámica de poblaciones, pueden ser también interpretados como sistemas celulares en los que los individuos de las distintas especies

interaccionan entre sí dentro de hábitats que permiten el traspaso de los mismos.

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, puede resaltarse el hecho de que constituye igualmente un buen modelo teórico de computación distribuida, en donde distintas unidades de cálculo trabajan independientemente pero estructuradas en una cierta jerarquía vertical; por ejemplo, la jerarquización usada en las conexiones establecidas en redes como internet puede ser representada como una estructura de membranas.

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 [87]. Entre las diversas variantes que se consideran en los P sistemas destacan las siguientes:

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

Además, se prueba que muchos de los modelos que se obtienen a partir de los

nuevos conceptos son computacionalmente completos; es decir, que el modelo que se obtiene posee la misma potencia computacional que una máquina de Turing.

A diferencia de los modelos de Computación Natural reseñados anteriormente, la Computación celular con membranas no dispone en la actualidad de ninguna implementación real, ya sea en el laboratorio (como en el caso de la computación molecular) o bien a través de una adaptación en ordenadores convencionales (como en el caso de los algoritmos genéticos y las redes neuronales). Hasta el momento, todo lo realizado en esta dirección se reduce a una interpretación como P sistemas de las redes de ordenadores convencionales (por ejemplo, *internet*), o de los procesos de ciertas reacciones químicas que se producen en medio acuoso [61], y a una simple simulación de ejecuciones de los P sistemas a través de lenguajes de programación convencionales.

Otra nota diferenciadora con respecto a los modelos de computación molecular basados en ADN, consiste en que este modelo no está descrito, propiamente, a través de un lenguaje de programación; es decir, no proporciona una serie de operaciones básicas susceptibles de ser secuenciadas sobre un dato de entrada para obtener un resultado final (solución del problema que se pretende resolver). En este modelo se generan *máquinas* (al modo de las máquinas de Turing) cuya ejecución modifica el contenido de las distintas componentes que la integran hasta llegar, en su caso, a un estado de parada (en el que la máquina deja de funcionar). En este sentido, la ejecución de los P sistemas se podría decir que es independiente del usuario, puesto que una vez construido el sistema no es necesario, en principio, intervenir para *dirigir* la ejecución.

Capítulo 2

Un marco de modelización basado en sistemas P

En este capítulo se aborda la problemática general que existe para modelizar fenómenos (procesos o sistemas complejos) de la realidad a fin de obtener nuevo conocimiento acerca de los mismos.

La primera sección está dedicada a analizar la necesidad de usar modelos formales para poder extraer información cualitativa y cuantitativa de los procesos que se tratan de estudiar.

En la sección 2 se describen brevemente algunas aproximaciones para el diseño de modelos formales de fenómenos biológicos, desde la pionera orientación continua basada en sistemas de ecuaciones diferenciales ordinarias (ODEs), a las orientaciones discretas basadas en distintos paradigmas de computación (sistemas basados en agentes, redes de Petri y álgebra de procesos).

La sección 3 está dedicada a la presentación de dos tipos de paradigmas computacionales que capturan la aleatoriedad inherente a los procesos que se producen en la naturaleza. En primer lugar, una semántica estocástica basada en el algoritmo clásico de Gillespie y, en segundo lugar, una semántica de tipo probabilístico. Ambas semánticas tiene como característica común el hecho

de asociar constantes o funciones numéricas a las reglas o ecuaciones que aparecen en el modelo, de tal manera que los valores numéricos asociados a cada regla o ecuación en un instante determinado juegan un papel relevante para determinar la siguiente configuración del sistema, a través de sendos algoritmos que tratan de describir las semánticas formales de los modelos.

El capítulo finaliza presentando un marco de especificación formal para la modelización de fenómenos biológicos basado en el paradigma de *Membrane Computing*.

2.1. Modelización de procesos reales

Informalmente, un modelo de un cierto fenómeno, sistema o proceso de la vida real es una representación concreta, abstracta, conceptual, gráfica o formal, que permite analizar, describir, explicar y, en general, profundizar en el conocimiento acerca del mismo. El uso de modelos es intrínseco a cualquier actividad científica. Los científicos usan regularmente abstracciones de la realidad tales como diagramas, grafos, leyes, relaciones, etc. con el fin de tratar de entender mejor la realidad que examinan. Los médicos, biólogos y ecólogos han estado siempre familiarizados con unos mecanismos de modelización, generalmente informales, estrechamente relacionados con los experimentos que realizaban en el laboratorio o sobre el terreno. En este contexto, las Matemáticas y la Informática han sido utilizadas por dichos colectivos simplemente como herramientas auxiliares para el mejor desarrollo cuantitativo de esos experimentos.

Sin embargo, durante las últimas décadas se ha producido un extraordinario avance en las técnicas usadas en los experimentos, lo cual ha permitido recopilar una masiva cantidad de información sobre los fenómenos objetos de estudio (búsqueda de las moléculas involucradas en ciertas funciones y en el análisis de su estructura como fue el caso de la insulina, la secuenciación del genoma de distintas especies, incluida la humana, y la búsqueda de genes, la recolección de una gran variedad de datos relativos a especies

en peligro de extinción, especies exóticas invasoras, etc.). Así se llegó a la conclusión que el avance en las técnicas de laboratorio/campo solamente nos había proporcionado la *partitura* de la complejidad molecular de los procesos celulares, sin darnos información directa acerca de cómo interpretar la música o de cómo componer nuestras propias melodías.

Esos progresos obtenidos durante finales del siglo pasado, tanto en Biología celular y molecular como en Ecología (y, por supuesto, en Ciencias de la Computación), así como la necesidad de realizar avances cualitativos acerca de los resultados obtenidos experimentalmente, han propiciado la convergencia de dichas disciplinas a través del uso de modelos formales.

Un modelo formal es una abstracción del mundo real dentro de un marco matemático-computacional que trata de resaltar algunos aspectos relevantes del sistema objeto de estudio a través del uso de sistemas formales; es decir, es una traducción de la realidad a un nuevo sistema expresado en términos matemático-computacionales.

Está ampliamente aceptado [101] que un buen modelo formal debe satisfacer, al menos, las cuatro siguientes propiedades: *relevancia*, *comprensibilidad*, *extensiblidad* y *tratablidad matemático–computacional*.

- Un modelo debe ser *relevante*; es decir, debe capturar las propiedades esenciales del fenómeno investigado de una manera unificada, tanto en su estructura como en su conducta dinámica.

- Un modelo debe facilitar una mejor *comprensibilidad* del sistema que se estudia; es decir, los formalismos abstractos usados en el modelo deberían corresponderse bien con los conceptos informales e ideas subyacentes del sistema, de tal manera que pueda ser fácilmente interpretado por los expertos en el tema objeto de estudio.

- Un modelo debe ser fácilmente *extensible* a otros niveles de organización y fácilmente modificable para incluir nuevo conocimiento o eliminar hipótesis falsas. Generalmente, el conocimiento que se tiene del sistema es dinámico y continuamente se está generando nueva información sobre

el mismo; por ello, el modelo debería ser lo suficientemente flexible a fin de poder incorporar, con facilidad, esa nueva información.

- Un modelo debe ser *tratable computacionalmente*, en el sentido de permitir su implementación en dispositivos electrónicos a fin de poder ejecutar simulaciones que permitan el estudio de la dinámica del sistema en diferentes escenarios, a través de la manipulación de las condiciones experimentales en el modelo, sin necesidad de realizar experimentos complejos y costosos en el laboratorio, posibilitando, de esta manera, su análisis matemático y computacional.

Uno de los objetivos fundamentales de cualquier modelo formal es su capacidad de *predicción*; es decir, la posibilidad de realizar conjeturas o hipótesis plausibles acerca de la dinámica del sistema modelizado que es objeto de investigación, en distintos escenarios.

El desarrollo de modelos formales es un proceso arduo en donde, con frecuencia, habrá que reconsiderar los supuestos, las simplificaciones, etc. realizadas en su diseño inicial. El primer paso consiste en establecer la parte específica del sistema que uno desea modelar, los objetivos a alcanzar, las cuestiones y preguntas que se trata de responder, así como la forma en que el modelo deberá ser validado y analizado. Seguidamente hay que especificar o traducir en un lenguaje formal la descripción intuitiva de las componentes del sistema y las cuestiones a tratar.

A continuación hay que elegir la forma de implementar o, en su caso, simular el modelo en ordenadores electrónicos. A este respecto, es fundamental la corrección de los algoritmos usados en relación con el modelo considerado debido, básicamente, a la gran cantidad de parámetros, variables y estructuras involucradas en las funciones celulares e interacciones entre individuos de las poblaciones que se investigan (tarea que entra, propiamente, dentro de la ingeniería del software). Hemos indicado que en el estudio y análisis de procesos biológicos intervienen una gran cantidad de parámetros, algunos de los cuales se obtienen experimentalmente mientras que otros no pueden ser obtenidos en el laboratorio/campo o conllevan la realización de experimentos

muy costosos. Por ello, antes de realizar simulaciones hemos de *calibrar* nuestro modelo; es decir, hemos de obtener estimaciones acerca de esos parámetros, y validarlos en función de la conducta esperada del sistema. Una vez que se han encontrado un conjunto de parámetros fiables, podemos centrarnos en algunas de las cuestiones que han motivado la introducción de nuestro modelo. Existen diferentes métodos de análisis en función del tipo de modelo, que van desde la simple generación de simulaciones sobre ordenadores electrónicos a sofisticados métodos estadísticos y/o de model checking.

Por otra parte, existen diversas formas de enfocar el proceso de modelización formal de sistemas complejos. A continuación, enumeramos algunas aproximaciones.

1. En relación con el escalado del espacio, se puede distinguir entre modelo *macroscópico*, modelo *microscópico* y *mesoscópico*. En el primero de ellos, el sistema se observa como un todo, sus componentes se representan con poco detalle y no se proporcionan mecanismos de interacción entre ellas. En el modelo *microscópico*, cada parte del sistema se representa con bastante detalle; por ejemplo, en el caso de fenómenos moleculares, se tiene en cuenta cada molécula y se especifica su posición y momento. Esta aproximación es intratable desde el punto de vista computacional en la mayoría de casos. Por último, el modelo *mesoscópico* se centra en el número de componentes individuales que integran el sistema, sólo tiene en cuenta las partes del sistema que se consideran especialmente relevantes, despreciando parámetros como, por ejemplo, la posición y el momento, en el caso de las moléculas. Esta aproximación es más tratable que la microscópica y, a la vez, conserva información más interesante que la macroscópica.

2. De acuerdo con el tipo de análisis realizado, un modelo formal puede ser *cuantitativo* (proporciona información y datos cuantitativos acerca del sistema que se estudia), o *cualitativo* (proporciona información cualitativa acerca del sistema y de su dinámica).

   Según el tipo de datos cuantitativos generados y el carácter de la

especificación del sistema, un modelo puede ser *discreto* (las componentes del sistema modelizado son representadas mediante individuos o entidades discretas y los datos generados son, también, discretos), o *continuo* (las componentes del sistema son representadas mediante variables continuas y los datos generados son continuos).

3. Respecto de su dinámica o evolución, los modelos se dividen en: *deterministas* (existe una única posible evolución del modelo a partir de unos parámetros iniciales) y *estocásticos* (existen distintas posibles evoluciones del modelo a partir de unos parámetros iniciales). En la dinámica estocástica, el paso de una configuración a una configuración siguiente se realiza de acuerdo con un criterio que captura la aleatoriedad; por tanto, a la hora de obtener resultados mínimamente fiables a partir de un determinado escenario, es necesario realizar un número elevado de simulaciones.

4. Y, finalmente, en relación con el origen de la información utilizada para su diseño, los modelos formales pueden ser *empíricos* (construidos a partir de observaciones directas o resultados experimentales) y *heurísticos* (diseñados a través de los mecanismos conocidos del sistema que se estudia).

2.2. Diferentes marcos de modelización

En esta sección se va a hacer un pequeño recorrido sobre los distintos marcos de modelización formal de fenómenos biológicos. Los sistemas de ecuaciones diferenciales ordinarias (ODEs) constituyeron el primer marco para esa modelización, tanto a nivel celular y molecular como a nivel de dinámica de poblaciones. Ello se debe, posiblemente, al hecho de que las ODEs habían sido utilizado décadas antes para el estudio y análisis de procesos dinámicos complejos. Sin embargo, la aproximación macroscópica, continua y determinista inherente a las ODEs es cuestionable, en particular, para el estudio de sistemas celulares con un bajo número de moléculas, en los que se

producen reacciones lentas o en estructuras no homogéneas [4, 50]. Además, esta aproximación no puede utilizar un número de variables excesivamente grande, lo cual restringe bastante el marco de actuación, en particular, a la hora de modelizar dinámica de poblaciones.

La complejidad de los procesos biológicos hacen necesario el uso de aplicaciones informáticas para poder extraer información del modelo que ayude a desvelar algunos mecanismos y funcionalidades subyacentes a dichos procesos. En el caso de modelos basados en ODEs habría que realizar una aproximación numérica, sin embargo, en la mayoría de los modelos computacionales es posible, al menos teóricamente, realizar una implementación *ad hoc* de dichas aplicaciones, aunque se carezca de un marco definido, consistente y formal. Recientemente, se han propuesto varios formalismos bien fundamentados para el diseño de modelos computacionales. En esta sección se presentan brevemente las aproximaciones computacionales más extendidas: los sistemas basados en agentes [62], las redes de Petri [54], y el álgebra de procesos (π-cálculo [101]). No obstante, en estas aproximaciones no se tiene en cuenta el papel crucial que juegan las membranas biológicas y la estructura jerarquizada y compartimentalizada en el funcionamiento de las células. Estas ideas serán estudiadas en la sección 2.4 a través del paradigma computacional de Membrane Computing.

2.2.1. Modelos basados en ODEs

Clásicamente, los sistemas de ecuaciones diferenciales ordinarias han sido utilizados para el estudio de sistemas dinámicos y, quizás, por esta razón, constituyeron el primer marco de modelización formal para el análisis de procesos biológicos. En particular, hoy día, las ODEs representan, sin lugar a dudas, la aproximación más utilizada para modelizar funciones celulares, dinámica de poblaciones, en general, y ecosistemas reales, en particular. Ahora bien, conviene hacer hincapié en la existencia de una serie de restricciones importantes cuando se usan las ODEs como marco de modelización; por ejemplo, en el caso de redes moleculares los modelos basados en ODEs parten

de dos supuestos o hipótesis básicas:

- *Las células tienen volúmenes homogéneos y las concentraciones no cambian con respecto al espacio*; es decir, se presupone que las células son volúmenes homogéneos y bien mezclados (el número de moléculas está uniformemente distribuido en el volumen). Desde luego, este supuesto depende de las unidades de tiempo y espacio con las que se trabaje. En el caso de las bacterias se discute la validez de esta hipótesis ya que se ha probado que la difusión es suficiente para que una proteína/molécula recorra todo el volumen de la bacteria en segundos. La hipótesis de homogeneidad espacial no suele cumplirse, por lo general, en células eucariotas, ni en muchos fenómenos biológicos; por ejemplo, aquellos en los que el tiempo para que ciertas moléculas se desplacen a través del sistema sea grande en comparación con el tiempo (promedio) que tardan las reacciones químicas.

- *Las concentraciones químicas varían continuamente a lo largo del tiempo y de manera determinista.* Este supuesto es válido si el número de moléculas en el volumen que reacciona es suficientemente grande (al menos, miles) y las reacciones químicas son rápidas. Sin embargo, en sistemas con pocas moléculas (centenares de moléculas es un número considerado pequeño) las interacciones moleculares tienen lugar de forma discreta y está separada por intervalos de tiempo no constantes.

Esas condiciones no se satisfacen en sistemas celulares con un número pequeño de moléculas, reacciones lentas y/o estructuras organizadas en diferentes compartimentos. No obstante, conviene resaltar el hecho de que las ODEs han sido usadas con gran éxito como marco de modelización en diferentes sistemas celulares dentro de la Biología de Sistemas, así como para el análisis de poblaciones de individuos en Ecología.

Más concretamente, los sistemas de ecuaciones diferenciales ordinarias han sido usados con éxito para modelizar la cinética de reacciones químicas *macroscópicas* cuyo objetivo es el análisis de la evolución media de la

concentración de las sustancias químicas a través de todo el sistema. En esta aproximación, el cambio de concentraciones a lo largo del tiempo es descrita para cada especie química, admitiendo implícitamente que la fluctuación en torno al valor medio de la concentración es pequeño en relación con la concentración. Este supuesto de homogeneidad puede ser razonable en determinadas circunstancias, pero deja de serlo en muchos casos (por ejemplo, cuando el número de moléculas es bajo o la distribución de ciertas moléculas relevantes en la célula no es uniforme). Así pues, aunque el modelo basado en ODEs puede producir resultados interesantes y útiles bajo ciertas restricciones, también puede proporcionar una visión distorsionada de lo que está ocurriendo en la célula [12].

Ahora bien, teniendo presente la complejidad de, por ejemplo, algunas rutas señalizadoras o la gran cantidad de especies que interaccionan en un ecosistema, con frecuencia es necesario utilizar un sistema de ecuaciones diferenciales con un número muy elevado de variables a fin de modelizar esos procesos, e incluso la interdependencia entre muchas ecuaciones diferenciales pueden hacer muy sensible al modelo en relación con sus condiciones iniciales. En este marco es difícil modelizar cuestiones como tiempo de retraso y efecto espacial [98] y pequeños cambios en la topología de la red molecular o en las interacciones de los individuos pueden exigir cambios sustanciales en muchas de las ecuaciones diferenciales básicas [14].

Para estos casos específicos, se está analizando la posibilidad de sustituir la aproximación basada en ODEs por otra que utilice sistemas de ecuaciones en derivadas parciales, lo cual puede resultar útil para el análisis de la dinámica de los sistemas espacialmente extendidos sobre escalas que son grandes comparadas con las longitudes de escalas de los objetos del proceso que se estudia. El estado del sistema se puede expresar en términos de funciones que dependen del espacio y del tiempo.

2.2.2. Aproximación basada en agentes

La modelización basada en agentes trata cada componente individual del sistema como una entidad simple (un agente) que tiene asociado su propio conjunto de reglas, teniendo la capacidad de interaccionar con el entorno y con agentes vecinos, de acuerdo con determinados protocolos.

Teniendo presente que los agentes pueden representar cualquier componente de un sistema, resulta que para una ruta señalizadora, por ejemplo, es posible representar mediante un agente desde una simple molécula (receptor, ligando, etc.) hasta una cadena de interacciones moleculares, pasando por los individuos de una cierta población en el caso de un ecosistema. Esto proporciona un marco de modelización extensible y modular.

Un modelo bioquímico basado en agentes no tiene las restricciones a las que está sujeto el modelado basado en ODEs: cualquier número y distribución de moléculas puede ser modelizado, las cuestiones espaciales y los tiempos de retraso en los procesos celulares pueden ser fácilmente incorporados en el modelo, y las interacciones individuales entre agentes no producen la misma *volatilidad* que un sistema de ecuaciones diferenciales interdependientes.

Recientemente, los sistemas basados en agentes han sido aplicados al estudio y análisis de distintos sistemas biológicos: comunidades de insectos (M. Holcombe y otros, 2003 [62], M. Gheorghe y otros [47], D. Jackson y otros, 2004 [65], [66]), tejido epitelial (D. Walker y otros, 2004 [119]), rutas señalizadoras (M. Pogson y otros, 2006 [98]), migración de células tumorales (L. Dib y otros, 2005 [37]), etc. El *Computational Systems Biology Group* del Dpt. of Computer Science de la Universidad de Sheffield ha sido pionero en este campo.

2.2.3. Redes de Petri

Un sistema bioquímico y una población de individuos puede ser prepresentada a través de un sistema de eventos discretos cuyas propiedades estructurales pueden resultar de utilidad para la obtención de conclusiones acerca del

comportamiento y de la estructura del sistema o de la población original [100].

Las redes de Petri (K.A. Petri, 1962) constituyen una herramienta matemático–computacional para la modelización y el análisis de sistemas de eventos discretos con un comportamiento concurrente, propiciando la representación formal de su estructura así como la simulación de su comportamiento y la demostración de ciertas propiedades del mismo.

Una red de Petri consiste en un grafo formado por dos tipos de nodos llamados *lugares* y *transiciones*, respectivamente. Los lugares y transiciones se conectan mediante arcos dirigidos con un peso asociado. Dada una transición se distingue entre lugares de entrada, que son aquellos nodos con arcos que llegan hasta la transición, y lugares de salidas que son aquellos nodos con arcos que salen de la transición y llegan a ellos. Normalmente una red de Petri se representa gráficamente dibujando los lugares como círculos y las transiciones como rectángulos.

Recientemente, el marco de modelización de las redes de Petri ha sido aplicado en diferentes campos de ingeniería de sistemas y en ciencias de la computación. La variante específica de redes de Petri, denominada *place-transition net* (PT-net) ha sido utilizada por P.J.E. Goss y otros [54], así como por V.N. Reddy y otros [100], para modelizar sistemas de interacciones moleculares. Para ello, se representa cada especie molecular como un *lugar* y cada transformación bioquímica como una *transición*. Los *tokens* dentro de un lugar pueden representar el presente de una molécula en ciertas proporciones.

Bioquímica	**PT-net**
Molécula	Lugar
Población Molecular	Marking
Transformación Bioquímica	Transición
Reactante	Lugar de entrada
Producto	Lugar de salida

En este marco computacional sólo se puede realizar un estudio cualitativo de los sistemas moleculares. Con la finalidad de obtener un marco cuantitativo

se desarrollaron las *redes de Petri estocásticas* en las que el tiempo de espera para el disparo de las transiciones se determina según una distribución exponencial cuyo parámetro se calcula, a su vez, de acuerdo con una constante asociada a cada transición y al número de *tokens* en los lugares de entrada.

2.2.4. Álgebra de procesos, π-cálculo

El π-cálculo fue introducido por R. Milner, J. Parrow y D. Walker como un lenguaje formal para describir procesos móviles que se comunican a través de canales de comunicación [80]. Está considerado como un modelo para sistemas que interaccionan y que poseen una topología dinámica de comunicación. El π-cálculo permite que los canales pasen, como si fueran datos, a través de otros canales y este hecho proporciona una movilidad a los canales, lo cual es un hecho importante que aumenta su potencia expresiva. La semántica del π-cálculo es relativamente simple y está basada en una teoría algebraica tratable. Partiendo de acciones atómicas y procesos más simples, se pueden construir procesos más complejos de muy diversas formas. La evolución de un proceso es descrita en π-cálculo mediante una relación de reducción entre procesos que contienen a aquellas transiciones que pueden ser inferidas a partir de un conjunto de reglas.

Diferentes variantes han sido usadas para modelar interacciones moleculares [102], redes de genes y para integrar redes moleculares y de genes [97].

En los últimos años se han producido diversos intentos de establecer puentes entre el π-cálculo y los sistemas P. En [71], J.B. Lingrel y T. Kuntzweiler han descrito paso por paso los mecanismos de transferencia de la bomba de sodio-potasio (*Na–K pump*) y se han usado herramientas de verificación formal para chequear la validez de esta aproximación. En [11], D. Besozzi y G. Ciobanu describen y analizan el funcionamiento de la misma bomba en el marco de los sistemas P. Para ello, se han definido nuevos ingredientes, como una variable de etiquetado de membranas, condiciones de activación para las reglas, membranas con doble capa y reglas de comunicación específicas.

Al igual que sucede con las redes de Petri ordinarias, este nuevo marco es sólo cualitativo. Por ello, para introducir información cuantitativa se asocia una constante a cada canal de comunicación y, entonces, el tiempo de espera para realizar una comunicación se determina mediante una distribución exponencial cuyo parámetro se calcula utilizando la constante citada y el número de procesos que intentan comunicarse a través de dicho canal.

2.3. Modelos estocásticos versus modelos probabilísticos

La aleatoriedad inherente a los procesos biológicos, así como el ruido externo y la incertidumbre se captura en los modelos computacionales considerados a través del uso de estrategias estocásticas basadas en el algoritmo de Gillespi o de estrategias probabilísticas con algoritmos de simulación *ad hoc*.

2.3.1. Modelos estocásticos

Las descripciones continuas y deterministas son adecuadas únicamente si es *grande* el número de individuos que intervienen en el proceso.

A nivel microscópico, el funcionamiento de procesos celulares sigue las leyes de la física. Un resultado fundamental de la física teórica es la famosa *ley* $\sqrt{n}$ que afirma lo siguiente: el nivel de aleatoriedad o fluctuación de un sistema es inversamente proporcional a la raíz cuadrada del número de individuos que intervienen en el sistema. En consecuencia, en un sistema bioquímico de células vivas con un número pequeño de moléculas de un cierto reactante, se exhibe un comportamiento estocástico y discreto más que continuo y determinista.

El primer paso para el análisis de descripciones estocásticas de reacciones químicas consiste en definir un conjunto de variables de estados suficientemente completo tal que los cambios sólo dependan del estado actual. Supondremos que el sistema objeto de estudio ocupa un volumen fijo, constante, V y que

se encuentra en equilibrio a una cierta temperatura. Supondremos, además, que el sistema consta de moléculas de n tipos o especies $\{s_1, \ldots, s_n\}$ tales que interactúan a través de una serie de reacciones químicas $\{r_1, \ldots, r_q\}$

De esta manera, es posible representar el estado instantáneo de un tal sistema a partir del número de moléculas de cada especie reactante; es decir, el estado del sistema en un instante t se representa mediante un vector $\mathbf{X}(t) = (X_1(t), \ldots, X_n(t))$, en donde $X_i(t)$ representa el número de moléculas de la especie molecular i en ese instante. Se trata de estudiar la evolución del vector de estado $\mathbf{X}(t)$ a partir de un estado inicial conocido $\mathbf{X}(t_0) = \mathbf{x}_0$.

Cada interacción molecular r_j está caracterizada por un *vector de cambio de estado* $\mathbf{v}_j = (v_{1j}, \ldots, v_{nj})$ y una constante de *propensidad* $p_j(\mathbf{X}(t))$ asociada a cada estado en un instante determinado. Las componentes v_{ij} representan el cambio que se produce en la población molecular de la especie s_i por la ejecución de la reacción r_j. Así pues, si en un instante determinado el estado del sistema es $\mathbf{X}(t)$ y se ejecuta una regla r_j, entonces el nuevo estado será $\mathbf{X}(t) + \mathbf{v}_j$.

La propensidad $p_j(\mathbf{X}(t))$ de una regla r_j en un instante t se define de tal manera que $p_j(X(t))dt$ sea la probabilidad de que tenga lugar una interacción molecular del tipo r_j en el intervalo de tiempo $[t, t + dt)$.

Para calular la propensidad $p_j(\mathbf{X}(t))$ de una regla r_j, se parte de una *constante cinética* k_j usada en la teoría cinética convencional, como ODE's, la cual depende de las propiedades físicas de las moléculas involucradas así como de otros parámetros físicos y que se suele calcular experimentalmente. A partir de ella, se calcula una *constante estocástica* c_j que permite hallar la propensidad, a través un proceso que depende del tipo de la reacción química r_j. Esta aproximación se denomina *modelización mesoscópica*.

El algoritmo de Gillespie permite explorar de forma exacta el espacio de estados asociado a un sistema de ecuaciones diferenciales con tantas ecuaciones como posibles estados (la *ecuación química maestra*). Se trata de un algoritmo de Monte Carlo para la simulación estocástica de interaciones moleculares que tienen lugar en un determinado volumen [50] (ver también

[48] para algunas mejoras recientes) y proporciona un método exacto para la simulación estocástica de sistemas de reacciones bioquímicas. La validez del método ha sido demostrada y, además, este algoritmo se ha usado con éxito en la simulación de una amplia gama de procesos bioquímicos [79]. Además, el algoritmo de Gillespie ha sido usado en la implementación del π-cálculo estocástico [97, 135], y en su aplicación a la modelización de procesos biológicos [99].

Supongamos que en un determinado medio acotado m (con un volumen determinado) existe un cierto multiconjunto de sustancias químicas que están sometidas a un conjunto de posibles reacciones químicas $r_1, \ldots, r_q$, de las que se conocen los valores $h_1, \ldots, h_q$ (número de posibles combinaciones de los reactantes), así como los valores $k_1, \ldots, k_q$ (constantes cinéticas de las reacciones). Entonces, el algoritmo (que, a partir de ahora, denominaremos clásico) de Gillespie nos permite seleccionar la reacción química a ejecutar y su correspondiente *tiempo de espera*. El agoritmo puede ser descrito como sigue:

1. Calcular $a_0 = \sum_{j=1}^{q} p_j$, siendo $p_j = h_j \cdot k_j$ las propensidades de las reglas.
2. Generar dos números aleatorios b_1 y b_2 uniformemente distribuido sobre el intervalo unidad $(0, 1)$.
3. Calcular el tiempo de espera para la siguiente reacción: $\tau_m = \frac{1}{a_0} ln(\frac{1}{b_1})$.
4. Elegir el índice j_0, de la siguiente reacción química, que verifica:
$$\sum_{k=1}^{j_0-1} p_k < r_2 \cdot a_0 \leq \sum_{k=1}^{j_0} p_k .$$
5. Devolver la terna (τ_m, j_0, m).

Entonces se aplicará *una vez* la regla r_{j_0} y se actualizará el número de moléculas de la membrana m.

El tiempo de cada paso de transición en la simulación/evolución del sistema se pueden considerar no constante, y será determinado en cada iteración a partir de la configuración del sistema.

2.3.2. Modelos probabilísticos

En la especificación sintáctica de un modelo computacional de tipo probabilístico se asocia una constante o una función probabilística una de cuyas variables independientes es el tiempo, a cada uno de los elementos básicos que controlan la dinámica del sistema (transiciones en el caso de las redes de Petri, canales de comunicación en el π–cálculo, o reglas de evolución en el caso de los sistemas basados en agentes o en los sistemas P).

Los valores numéricos asociados a esos elementos del modelo en un instante determinado, representan la probabilidad que cada uno de ellos tiene de ser utilizados para que el sistema formal evolucione en ese momento. Dichas constantes o funciones son obtenidas experimentalmente o bien son calculadas a través de una cierta distribución de probabilidad o, incluso, en algunos casos pueden aproximarse a través de un proceso de calibración. Posteriormente, se diseña un algoritmo *ad hoc* que trata de describir la semántica del modelo computacional.

2.4. Modelos basados en sistemas P

La Computación celular con membranas (*Membrane Computing*) trata de capturar ideas computacionales a partir de la estructura y el funcionamiento de las células vivas, así como de la forma en que éstas están organizadas en tejidos o en otras estructuras de orden superior. Los sistemas P proporcionan un nuevo marco de modelización computacional que integra los aspectos estructurales y dinámicos de los sistemas celulares en una forma comprensiva y relevante, a la vez que facilita la formalización necesaria para realizar análisis matemáticos y computacionales [88]. La versatilidad y flexibilidad de los modelos, el trabajar con elementos de matemática discreta (el uso de la reescritura en multiconjuntos está próximo a la notación biológica), la consideración explícita del papel relevante de las membranas en los sistemas biológicos (difusión, filtro selectivo de compuestos químicos del entorno, recepción de señales externas, distribución espacial de especies, etc.), el carácter algorítmico (que implica

una fácil programabilidad), la escalabilidad y el paralelismo masivo, así como la modularidad del diseño, hacen especialmente atractivo este nuevo paradigma de modelización computacional.

Este marco parte del supuesto de que los procesos que tienen lugar en la estructura compartimentalizada de una célula viva, tanto a nivel de reacciones químicas como a nivel de flujo de sustancias entre los distintos compartimentos (membranas biológicas) que la componen, pueden ser interpretados como procedimientos de cálculo; es decir, como computaciones. La Computación celular con membranas parte de la observación de que las membranas biológicas juegan un papel relevante en el funcionamiento de las células. La membrana plasmática separa y, por tanto, protege el espacio interno celular del entorno externo. Las membranas interiores definen la estructura de la célula identificando un número de compartimentos internos que permiten proteger al núcleo que contiene toda la información genética. Las membranas están involucradas en muchas reacciones químicas que tienen lugar dentro de los compartimentos y, además, actúan como canales selectivos de comunicación entre las células y su entorno.

Los sistemas P tienen en consideración el carácter discreto de la cantidad de componentes del sistema usando reglas de reescritura sobre multiconjuntos de objetos que representan a las sustancias químicas, especies de individuos u otros ingredientes de una población. En la versión original de los sistemas P, las reglas se aplican de manera *paralela maximal* y cada una de ellas se ejecuta en una unidad de tiempo. Sin embargo, pronto se descubrió que esta aproximación provocaba, básicamente, dos importantes distorsiones respecto de la realidad biológica: en primer lugar, que las reglas aplicables no se ejecutaban todas ellas con la misma frecuencia que las reacciones químicas que representaban y, en segundo lugar, que no se capturaba la dinámica del sistema modelizado, debido a que todo paso computacional consumía la misma cantidad de tiempo.

Es importante resaltar el hecho de que, de acuerdo con la motivación original, la introducción de los sistemas P no tenía como finalidad proporcionar un modelo comprensivo y completo de las células vivas sino, más bien, explorar la naturaleza computacional de una serie de hechos relativos a las

membranas biológicas. Por ello, se ha estudiado exhaustivamente la potencia computacional de distintas variantes de sistemas P, así como su eficiencia computacional, entendiendo como tal la capacidad de dichos dispositivos para resolver problemas computacionalmente duros en tiempo polinomial, por supuesto, intercambiando recursos de tiempo por espacio. Recientemente, los sistemas P están siendo utilizados para modelizar fenómenos biológicos dentro del marco de la Biología de Sistemas, proporcionando modelos de sistemas oscilatorios [40], de traducción de señales [93], de control de la regulación de genes [105], del quorum sensing [106], de metapoblaciones [95], y de ecosistemas reales [19, 20].

2.4.1. Especificación sintáctica

Seguidamente, se presenta un marco de especificación para la modelización de un amplio espectro de fenómenos biológicos que van desde el nivel micro para procesos moleculares y celulares, hasta el nivel macro para el estudio de la dinámica de poblaciones.

Definición 2.1. *Un esqueleto de un sistema P extendido con membranas activas de grado* $q \geq 1$ *es una tupla* $\Pi = (\Gamma, \mu, R)$*, en donde:*

- Γ *es un alfabeto (el alfabeto de trabajo);*
- μ *es una estructura de membranas (es decir, un árbol enraizado) que consta de* q *membranas etiquetadas de manera inyectiva por* $0, 1, \ldots, q-1$*. La etiqueta de la membrana piel es 0. Cada membrana tiene asociada una carga eléctrica del conjunto* $\{0, +, -\}$*.*
- R *es un conjunto finito de reglas de evolución del tipo* $u[\ v\ ]_i^{\alpha} \rightarrow u'[\ v'\ ]_i^{\alpha'}$*, en donde* u, v, u', v' *son multiconjuntos sobre* Γ*,* $i \in \{0, \ldots, q-1\}$*, y* $\alpha, \alpha' \in \{0, +, -\}$*;*

Un esqueleto, $\Pi = (\Gamma, \mu, R)$, de un sistema P extendido con membranas activas de grado $q \geq 1$, se puede considerar como un conjunto de membranas

etiquetadas por $0, \ldots, q-1$ y jerarquizadas según una estructura μ. Se supone que todas las membranas tienen carga neutra y, además, cada una de ellas tiene asociado un conjunto finito de reglas de R.

Definición 2.2. *Un sistema P funcional con membranas activas de grado $q \geq 1$, usando T unidades de tiempo, $T \geq 1$, es una tupla*

$$\Pi = (\Gamma, \mu, R, T, \{f_r : \ r \in R\}, \mathcal{M}_0, \ldots, \mathcal{M}_{q-1})$$

en donde:

- (Γ, μ, R) *es el esqueleto de un sistema P extendido con membranas activas de grado q.*
- T *es un número natural, $T \geq 1$;*
- *Para cada $r \in R$, f_r es una función real computable cuyo dominio es $\{1, \ldots, T\}$.*
- $\mathcal{M}_0, \ldots, \mathcal{M}_{q-1}$ *son multiconjuntos de objetos sobre* Γ *inicialmente colocados en las q membranas de μ etiquetadas por $0, \ldots, q-1$, respectivamente.*

Un sistema P funcional con membranas activas de grado $q \geq 1$, usando T unidades de tiempo, $\Pi = (\Gamma, \mu, R, T, \{f_r : \ r \in R\}, \mathcal{M}_0, \ldots, \mathcal{M}_{q-1})$, se puede considerar como un conjunto de q membranas etiquetadas inyectivamente por $0, \ldots, q-1$ y jerarquizadas por la estructura μ. El número natural $T \geq 1$ representa el tiempo de simulación del sistema. Para cada regla $r \in R$ y cada instante $t = 1, 2, \ldots, T$, el número $f_r(t)$ representa una constante asociada a la regla r en el instante t. Notaremos de manera genérica $r : u[v]_i^\alpha \xrightarrow{f_r(t)} u'[v']_i^{\alpha'}$. Si $f_r(t) = 1$, entonces omitiremos la expresión $f_r(t)$ y escribiremos más brevemente $r : u[v]_i^\alpha \longrightarrow u'[v']_i^{\alpha'}$.

La tupla de multiconjuntos que están presentes en un instante dado en las q membranas del sistema, junto con las polarizaciones de las mismas, constituye la *configuración* del sistema en ese instante. La tupla $(\mathcal{M}_0, \ldots, \mathcal{M}_{q-1})$, junto

con las polarizaciones neutras en cada membrana, determinan la configuración inicial del sistema Π.

El sistema P puede pasar de una configuración a otra mediante la aplicación de las reglas del conjunto R de acuerdo con el siguiente criterio:

- Una regla $u[v]_i^{\alpha} \rightarrow u'[v']_i^{\alpha'}$ es aplicable a una membrana etiquetada por i cuya carga eléctrica es α si el multiconjunto v está contenido en esa membrana y el multiconjunto u está contenido en su padre. Si una tal regla es aplicada, entonces los multiconjuntos u y v son eliminados de las citadas membranas y son reemplazados por los multiconjuntos u' y v', respectivamente. Además, la polarización pasa a ser α'.

- La familia $\{f_r : r \in R\}$ de funciones computables asociadas a las reglas sirve de base para definir la dinámica del sistema, entre cuyas orientaciones más relevantes destacan la semántica estocástica y la semántica probabilística.

Definición 2.3. *Un sistema P multientorno funcional con membranas activas de grado* (q, m, n)*, con* $q \geq 1$*,* $m \geq 1$ $n \geq 1$*, usando* T *unidades de tiempo,* $T \geq 1$*, es una tupla*

$$(G, \Gamma, \Sigma, \mu, R_\Pi, R_E, \{f_{r,k} : r \in R_\Pi, 1 \leq k \leq n\}, \{\mathcal{M}_{i,k} : 0 \leq i \leq q-1, 1 \leq k \leq n\})$$

En donde:

- $G = (V, S)$ *es un grafo dirigido tal que* $(e, e) \in S$*, para cada* $e \in V$*. Los elementos del conjunto de nodos de* G*,* $V = \{e_1, \ldots, e_m\}$*, se denominan entornos;*

- Γ *es el alfabeto de trabajo y* $\Sigma \subsetneq \Gamma$ *es un alfabeto que representa el conjunto de objetos que pueden estar presentes en los entornos;*

- *Para cada regla* $r \in R_\Pi$ *y cada* k*,* $1 \leq k \leq n$*,* $f_{r,k}$ *es una función real computable cuyo dominio es* $\{1, \ldots, T\}$*;*

- $(\Gamma, \mu, R_\Pi) = \Pi$ *es el esqueleto de un sistema P extendido con membranas activas de grado* q*;*

- *Para cada* $1 \leq k \leq n$, $(\Gamma, \mu, R_\Pi, T, \{f_{r,k} : \ r \in R_\Pi\}, \mathcal{M}_{0,k}, \ldots, \mathcal{M}_{q-1,k})$, *que notaremos* Π_k *es un sistema P funcional con membranas activas de grado* q *usando* T *unidades de tiempo. Denotaremos por* R_{Π_k} *el conjunto* R_Π *en donde cada regla* $r \in R_\Pi$ *tiene asociada la función* $f_{r,k}$*;*

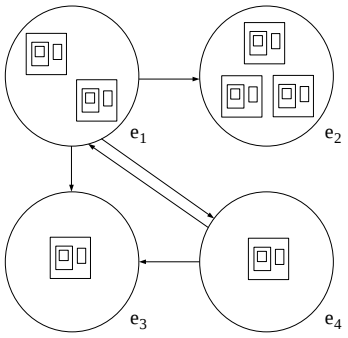
- R_E *es un conjunto finito de reglas de comunicación entre entornos, de la forma*

$$(x)_{e_j} \xrightarrow{p_{(x,j,j')}} (y)_{e_{j'}} \qquad y \qquad (\Pi_k)_{e_j} \xrightarrow{p_{(k,j,j')}} (\Pi_k)_{e_{j'}}$$

en donde $x, y \in \Sigma$, $(e_j, e_{j'}) \in S$, $1 \leq k \leq n$, *y* $p_{(x,j,j')}$ *y* $p_{(k,j,j')}$ *son funciones reales computables cuyo dominio es* $\{1, \ldots, T\}$.

Un sistema P multientorno funcional con membranas activas de grado (q, m, n), con $q \geq 1$, $m \geq 1$, $n \geq 1$, usando T unidades de tiempo, $T \geq 1$, se puede considerar como un conjunto de m entornos conectados a través de los arcos de un grafo dirigido G en el que existen n sistemas P eventualmente distintos pero con el mismo esqueleto.

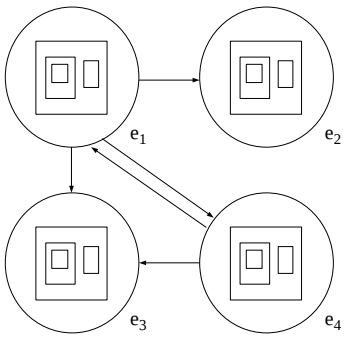
En la *orientación estocástica*, las funciones $p_{(x,j,j')}$ y $p_{(k,j,j')}$ son constantes y todos los sistemas Π_k son idénticos. Además, los n sistemas Π_k son distribuidos aleatoriamente, en el instante inicial, entre los m entornos como, por ejemplo, muestra la siguiente figura.



Al aplicar una regla de comunicación entre entornos $(x)_{e_j} \xrightarrow{c} (y)_{e_{j'}}$, el objeto x pasa del entorno e_j al entorno $e_{j'}$, posiblemente transformado en

otro objeto y. Del mismo modo, al aplicar una regla de comunicación entre entornos $(\Pi_k)_{e_j} \xrightarrow{c'} (\Pi_k)_{e_{j'}}$, el sistema Π_k pasa del entorno e_j al entorno $e_{j'}$. El papel que juegan las constantes c y c' en la aplicación de estas reglas viene determinado por el algoritmo de simulación que se considere para capturar la semántica del sistema.

En la *orientación probabilística*, el número de entornos coincide con el de sistemas P; es decir, $m = n$ y cada entorno e_j contiene, inicialmente, el sistema Π_j de tal manera que todos esos sistemas poseen el *mismo* esqueleto Π y, además, $\mathcal{M}_{0,j}, \ldots, \mathcal{M}_{q-1,j}$ describen los correspondientes multiconjuntos iniciales de Π_j, como indica la siguiente figura. Además, las funciones $p_{(k,j,j')}$ son constantes e iguales a cero (es decir, no existen reglas del entorno del tipo $(\Pi_k)_{e_j} \xrightarrow{p_{(k,j,j')}} (\Pi_k)_{e_{j'}}$).



En ambas orientaciones y al igual que sucede en los sistemas P clásicos, supondremos la existencia de un reloj universal que marca las unidades de tiempo para todo el sistema; es decir, la aplicación de todas las reglas en todas las membranas está sincronizada.

La tupla de multiconjuntos presentes en cada instante en los m entornos y en cada una de las regiones de los sistemas P colocados en esos entornos, junto con las polarizaciones de las membranas, constituye la configuración del sistema en ese momento. En la configuración inicial del sistema se consideran los correspondientes multiconjuntos iniciales de todos los sistemas P que

intervienen y, además, se admite que todos los entornos están vacíos en ese instante inicial y que todas las membranas tienen carga neutra.

El sistema puede pasar de una configuración a otra mediante la aplicación de las reglas del conjunto $R = R_E \cup \bigcup_{k=1}^{n} R_{\Pi_k}$ como sigue: en cada paso de computación, el número de veces que las reglas serán aplicadas se elige de acuerdo con las constantes asociadas a esas reglas en ese instante, y todas las reglas aplicables serán simultáneamente aplicadas de acuerdo con el criterio de maximalidad.

2.4.2. Un algoritmo de simulación para sistemas P estocásticos

Finalmente, en esta sección vamos a describir un algoritmo de simulación para sistemas P estocásticos, inspirado en el algoritmo clásico de Gillespie. En el capítulo 5 se describirá un algoritmo de simulación para sistemas P probabilísiticos.

Supongamos que tenemos un sistema P estocástico (más concretamente, de tipo mesoscópico) que modeliza un cierto proceso biológico. Recordemos que el algoritmo original de Gillespie trabaja con un único volumen, mientras que en los sistemas P se dispone de una estructura jerarquizada en la que cada membrana del sistema delimita una región o compartimento que incluye, propiamente, un volumen específico. Cada uno de esos compartimentos contiene su propio conjunto de reglas (que son abstracciones de las interacciones moleculares) y multiconjuntos de objetos (que son abstracciones de las moléculas). Por ello, si deseamos *adaptar* el algoritmo de Gillespie al marco de los sistemas P será necesario desarrollar una extensión del mismo, de tal manera que cada membrana del sistema pueda evolucionar de acuerdo con el algoritmo de Gillespie clásico, el cual nos devolverá la regla que se va a aplicar en esa membrana, así como su correspondiente *tiempo de espera*.

Una extensión del citado algoritmo para ser usado como algoritmo de simulación de sistemas P estocásticos ha sido desarrollado por F.J. Romero y

M.J. Pérez en [106] y se denomina *algoritmo de Gillespie multicompartimental* cuyo pseudocódigo es el siguiente:

- **Inicialización**
  - Poner el tiempo de simulación a $t = 0$.
  - Para cada membrana m en la estructura μ hacer:
    1. Cargar las reglas que pueden ser aplicadas en la membrana así como sus constantes estocásticas.
    2. Cargar el número de moléculas determinado por el multiconjunto inicial asociado a la membrana.
    3. Ejecutar el algoritmo de Gillespie clásico en esa membrana, devolviendo una terna (τ_m, j, m).
  - Ordenar la lista de ternas (τ_m, j, m) con relación a τ_m (en orden ascendente), formando una pila de reglas a aplicar;
- **Iteración**
  - Elegir la primera terna de la pila, (τ_m, j, m).
  - Poner el tiempo de simulación a $t = t + \tau_m$.
  - Actualizar el tiempo de espera para el resto de las ternas, restando el valor τ_m.
  - Aplicar una vez la regla r_j modificando el número de objetos en las membranas afectadas.
  - Para cada membrana m' afectada por esa aplicación, eliminar la terna correspondiente $(\tau'_{m'}, j', m')$ de la pila.
  - Para cada membrana m' afectada por la aplicación de la regla j volver a ejecutar el algoritmo de Gillespie clásico en el nuevo contexto, obteniendo una nueva terna $(\tau''_{m'}, j'', m')$.
  - Añadir la terna $(\tau''_{m'}, j'', m')$ a la lista y ordenarlas de nuevo de acuerdo con los tiempos de espera, formando una nueva pila.
  - Iterar el proceso.
- **Finalización**
  - Si el tiempo de la simulación t alcanza o excede a un tiempo maximal prefijado, finalizar el proceso.

En esta aproximación el tiempo de espera calculado por el algoritmo de Gillespie clásico se utiliza para elegir la(s) membrana(s) que evolucionará(n) en el siguiente paso; concretamente, aquellas que tienen menor tiempo de espera. Además, teniendo presente que la aplicación de una regla en una membrana puede afectar a otras (ya que algunos objetos pueden atravesar una membrana), es necesario volver a ejecutar el algoritmo clásico de Gillespie sobre esas membranas teniendo presente la nueva distribución.

Este nuevo algoritmo se puede aplicar a sistemas biológicos compartimentalizados y con distintos volúmenes, en donde el número de moléculas o individuos que intervienen es arbitrario (grande o pequeño).

Parte II

Aplicaciones informáticas en *Membrane Computing*

Capítulo 3

Simuladores de sistemas P

Debido a la naturaleza bio-inspirada, masivamente paralela y no determinista de los sistemas P, su implementación real constituye un gran reto para la ciencia y la tecnología actual. Si bien es cierto que existen estudios preliminares analizando la problemática relativa a dicha implementación (e.g. [46]), aún queda un largo camino hasta alcanzar el objetivo final, respecto al que se muestra especialmente escéptico el colectivo de los biólogos, en lo que a dispositivos celulares se refiere.

Es por ello que la *simulación* de sistemas P utilizando dispositivos electrónicos convencionales se convierte en una necesidad de vital importancia para el progreso de las actividades científicas en *Membrane Computing*.

Recordemos que todo modelo de computación consta de una especificación sintáctica y su dinámica se rige por medio de una semántica formal. En esta memoria, usaremos el término de *simulador* de un modelo formal de computación a una aplicación software/hardware que describe la especificación a través de un cierto lenguaje de programación y captura la semántica mediante la implementación de un algoritmo de simulación que debe reproducir la dinámica con fidelidad; es decir, cada paso de computación del modelo formal es reproducido en el simulador a través un número finito de pasos, de tal manera que el simulador es capaz de determinar los elementos básicos del modelo que han intervenido de forma relevante en ese paso.

Los simuladores suelen seguir una política de *caja negra*; es decir, producen la misma salida que la máquina simulada, pero no informan al usuario sobre cómo se ha obtenido. En otras palabras, no muestran el funcionamiento del algoritmo de simulación.

En los últimos años, un gran número de aplicaciones de *software* y *hardware* para *Membrane Computing* han sido presentadas. La mayoría de ellas son simuladores que replican una o varias de las posibles computaciones del sistema P simulado, siendo posible en algunos casos obtener todas las computaciones. Estos simuladores reciben como dato de entrada la configuración inicial de un sistema P junto con su conjunto de reglas, y producen un conjunto de computaciones como dato de salida, o cualquier otra información relevante para el usuario.

El crecimiento del número de estas aplicaciones se ha producido simultáneamente con el desarrollo teórico. En este capítulo se proporciona una descripción general del estado del arte de las aplicaciones informáticas para *Membrane Computing*. En la sección 3.1 se analiza la estructura general de un simulador para sistemas P; y, en la siguiente sección, se clasifican las aplicaciones informáticas existentes atendiendo a diferentes criterios, recogiendo un listado cronológico de las aplicaciones más relevantes desarrolladas hasta la fecha, así como explicando brevemente la finalidad y la funcionalidad de cada una.

3.1. Estructura general de un simulador para sistemas P

En esta sección se exponen los elementos comunes que, habitualmente, son incluidos en la mayoría de los simuladores de sistemas P (independientemente de su finalidad), tomando como referencia las características técnicas de las aplicaciones enumeradas en la sección 3.2.

En la figura 3.1 se presenta el esquema general de un simulador de sistemas P, donde se pueden observar los siguientes elementos comunes:

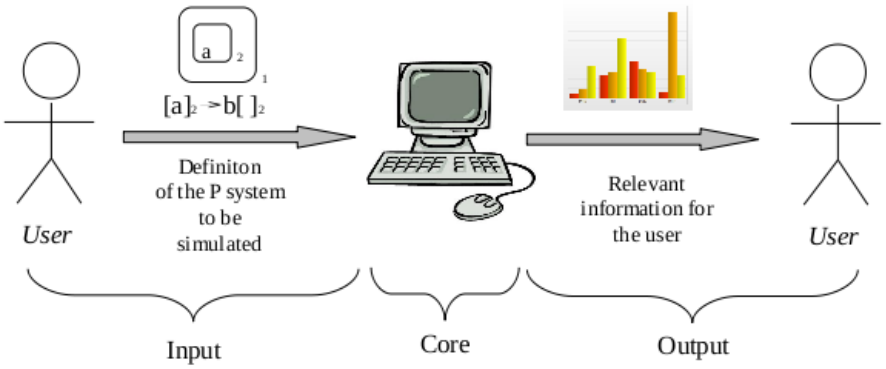


Figura 3.1: Elementos comunes en los simuladores de sistemas P

- Definición del sistema P que va a ser simulado.
- Núcleo de simulación.
- Presentación de resultados al usuario.

3.1.1. Definición del sistema P

Antes de poder simular un sistema P es necesario establecer una especificación que permita definirlo completamente. Esto implica que al simulador se le ha de suministrar la siguiente información:

- El modelo de sistema P a simular.
- La estructura inicial de membranas.
- Los multiconjuntos iniciales.
- El conjunto de reglas.

Esta tarea de definición se complica cuando es necesario especificar familias de sistemas P (como es el caso cuando se diseñan soluciones a problemas

de decisión) en donde el conjunto de reglas, el alfabeto, los multiconjuntos iniciales y, eventualmente, la estructura de membranas dependen de los valores asignados a unos parámetros iniciales.

Las principales soluciones para la definición de sistemas P empleadas por las aplicaciones descritas en la Sección 3.2 son los siguientes:

- Definición del sistema P dentro del código fuente.
- Definición del sistema P mediante interfaces de usuario.
- Definición del sistema P mediante ficheros externos.

Con la intención de diseñar una estrategia general para la definición de sistemas P, vamos a analizar las soluciones mencionadas atendiendo a una serie de criterios bien conocidos en *Ingeniería del Software.* [1]

Los criterios considerados son: (a) acoplamiento; (b) reutilización; y (c) usabilidad.

Se entiende por *acoplamiento* el grado de interdependencia que hay entre los distintos módulos de un programa. Lo deseable es que esta interdependencia sea lo menor posible, es decir, que exista un bajo acoplamiento.

La *reutilización* de código se refiere al comportamiento y a las técnicas que garantizan que una parte o la totalidad de un programa informático existente se pueda emplear en la construcción de otro programa. De esta forma se aprovecha el trabajo anterior, se economiza tiempo y se reduce la redundancia.

La *usabilidad* es la facilidad que tienen los usuarios para utilizar una herramienta de software con el fin de alcanzar un objetivo concreto. La usabilidad está relacionada directamente con la interacción persona-ordenador.

Los principios básicos que miden la usabilidad son:

- *Facilidad de Aprendizaje:* se refiere al esfuerzo requerido por parte de los nuevos usuarios hasta que logran alcanzar una interacción efectiva con el

[1] disciplina de la Informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad.

sistema o producto. Está relacionada con la predicibilidad, sintetización, familiaridad, generalización de conocimientos previos y consistencia.

- *Flexibilidad:* está relacionada con la variedad de posibilidades con las que el usuario y el sistema pueden intercambiar información. También incluye la posibilidad de diálogo, la multiplicidad de vías para realizar la tarea, similitud con tareas anteriores y la optimización entre el usuario y el sistema.

- *Robustez:* es el nivel de apoyo al usuario dirigido a facilitarle el cumplimiento de sus objetivos. Está relacionada con la capacidad de observación y supervisión de las acciones introducidas por el usuario, de recuperación de información y de ajuste de la tarea del usuario a las necesidades del mismo.

Definición del sistema P dentro del código fuente

Se entiende por *código fuente* de un programa el conjunto de líneas de texto que representan las instrucciones que debe seguir la máquina para ejecutar dicho programa.

Una primera propuesta para definir el sistema P que va a ser simulado consiste en integrar dicha definición dentro del propio código fuente del programa. Ello implica diseñar una representación adecuada utilizando el mismo lenguaje de programación que se utiliza para el simulador.

Podemos afirmar que ésta es la solución más rápida de implementar ya que el programador conoce bien el lenguaje de programación empleado y sólo necesita diseñar la estructura de datos adecuada. Por tanto, se trata de una solución rápida y eficaz para diseñar un prototipo o versión preliminar de un simulador posterior, facilitando de esta manera las tareas de depuración del núcleo de simulación y la realización de pruebas unitarias.

En cambio, esta solución presenta bastantes problemas si se pretende utilizar como parte de la versión estable de un determinado simulador debido, principalmente, a que:

- Es una solución *fuertemente acoplada*: la definición del sistema P a simular depende completamente del código fuente del simulador. Si se necesita hacer algún cambio en dicha definición, por mínimo que sea, es necesario modificar el código fuente del programa, lo cual puede ocasionar errores difíciles de depurar.

- Es una solución *muy poco reutilizable*: sólo es posible simular el sistema P definido en el código fuente, empleando únicamente el lenguaje de programación y las estructuras de datos utilizadas inicialmente en el desarrollo del simulador. Si se desea exportar la definición del sistema P a otro software, sería necesario emplear el mismo lenguaje y la misma representación de datos.

- La *usabilidad es escasa*: la definición del sistema P queda encapsulada dentro del código fuente, provocando que la interacción del usuario con dicha definición sea muy difícil y en absoluto flexible. Si se desea cambiar algún aspecto de la definición, es necesario acceder al código fuente y conocer la estructura del programa y el lenguaje de programación, lo cual implica un aprendizaje. Además, existe poca robustez, por la dificultad para que el programa supervise los posibles cambios que el usuario pudiera hacer.

Si bien esta solución es adecuada para desarrollos rápidos de prototipos, también podría ser válida para aquellos simuladores que siempre se ejecutan sobre el mismo sistema P. No obstante, esto último no suele ocurrir, ya que lo normal es que el usuario desee realizar algún cambio sobre el sistema P definido, o bien que se necesite introducir otros sistemas P distintos.

Definición del sistema P mediante interfaz de usuario

Otra propuesta, tal vez más flexible, para definir el sistema P que va a ser simulado consiste en el desarrollo de una *interfaz de usuario* que, en general, es la parte de un programa informático destinada al usuario y que le permite comunicarse con la máquina. Este desarrollo comprende todos los puntos de

contacto entre el usuario y el equipo. Los interfaces de usuario deben ser fáciles de entender y utilizar.

De especial interés son las interfaces gráficas de usuario, conocidos también como GUI (del inglés *Graphical User Interface*), que utilizan conjuntos de imágenes y objetos gráficos para representar la información y las acciones disponibles en la interfaz.

La ventaja fundamental de recurrir a interfaces gráficas consiste en que es posible representar de manera amigable el sistema P que va a ser simulado. Por otra parte, uno de los puntos más importantes es la dificultad que entraña su desarrollo. El diseño de interfaces de usuario eficaces es una tarea difícil de ingeniería ya que se trata directamente con una de las componentes más complejas de todo sistema informático: el usuario. Más aún, para toda aplicación suelen existir diversas categorías de usuarios, con diferentes conocimientos, gustos e intereses, y precisamente esta disparidad es la que dificulta implementar una interfaz amigable orientada al abanico completo de posibles usuarios.

Atendiendo a los criterios considerados y suponiendo que la interfaz de usuario está diseñada de manera óptima, se puede afirmar que:

- Es una solución de *poco acoplamiento*: en un buen diseño de software, se debe implementar la interfaz de usuario de manera independiente a la lógica del programa; es decir, con un acoplamiento mínimo. Por tanto, debería ser posible modificar o intercambiar el código fuente de la interfaz de usuario sin que esto afecte al resto del programa.

- Es una solución *moderadamente reutilizable*: al implementar la interfaz de usuario de manera independiente, es posible reutilizarla en otros programas con la restricción de que se deben utilizar las mismas estructuras de datos y, muy posiblemente, el mismo lenguaje de programación con el cual está implementada la interfaz. Para ello, se necesitan conocimientos técnicos a nivel de programación.

- Es una solución con *buena usabilidad*: la interacción entre el usuario

y la máquina debe ser fácil de aprender y de utilizar con una interfaz de usuario correctamente diseñada. Lo ideal es facilitar que se pueda definir el sistema P a simular con cierta flexibilidad (por ejemplo, debería ser posible definir sistemas P que dependen de una serie de parámetros iniciales, dando libertad al usuario para que asigne sus valores). Una interfaz correctamente diseñada proporcionaría una gran robustez, ya que la propia interfaz podría hacer llamadas a métodos que supervisen los datos introducidos por el usuario, permitiendo sólo aquellos que fueran coherentes con el sistema P que se está definiendo.

En consecuencia, la definición de sistemas P a través de interfaces de usuario es una buena solución siempre que se haga un correcto diseño de la interfaz. Una de las principales dificultades radica en la complejidad de programar las interfaces de usuario (especialmente si son gráficas) y, además, esta tarea suele ser larga debido a la necesaria interacción con los usuarios durante el proceso de diseño con el fin de garantizar una solución bien adaptada a sus gustos y necesidades. En caso de prescindir de este proceso de interacción, se corre el riesgo de obtener un producto que no responda a las expectativas de los usuarios.

Conviene hacer notar la existencia de soluciones híbridas entre la definición del sistema P en el código fuente y la utilización de interfaces de usuario. Por ejemplo, se podría programar en el código fuente la definición de una familia de sistemas P que dependan de unos parámetros iniciales y diseñar una interfaz apropiada que permita al usuario la introducción de valores asignados a dichos parámetros. Este tipo de solución ganaría algo más de usabilidad con respecto a realizar toda la definición dentro del código fuente, pero seguiría teniendo un alto grado de acoplamiento y escasa reutilización.

Definición del sistema P mediante ficheros externos

Otra aproximación consiste en utilizar ficheros externos a la aplicación para definir el sistema P que va a ser simulado. Estos ficheros deben seguir un formato determinado y pueden ser editados por terceras aplicaciones.

Un fichero es un conjunto de bits que codifica algún tipo de información y se encuentra almacenado en un dispositivo periférico de almacenamiento como, por ejemplo, un disco duro, un pen-drive o un CD-ROM, DVD, etc. Dependiendo de la forma en que se codifique la información, los ficheros se pueden clasificar en ficheros de texto y ficheros binarios.

En los ficheros de texto la información se almacena mediante secuencias de caracteres codificados según un código determinado. Por ejemplo, ASCII es un código de caracteres de 7 bits + 1 bit de control basado en el alfabeto latino, el estandar ISO-8859-1 es una extensión que utiliza 8 bits para proporcionar caracteres adicionales usados en idiomas distintos al inglés, como es el español. En este tipo de ficheros, cada byte (secuencia de 8 bits) representa un carácter y, debido a esta codificación, suelen ser eficaces a la hora de guardar textos, sin olvidar el hecho de que existan muchos y variados programas informáticos para su edición.

Los ficheros binarios están compuestos por secuencias de bytes que pueden codificar cualquier tipo de información, pudiendo agruparse los bytes de diversas maneras para representar cualquier estructura de datos, siendo necesario conocer el formato del fichero para decodificar su información. Por ejemplo, un fichero binario podría contener una secuencia de números enteros, en donde cada número se representa en binario con 4 bytes. La ventaja de los ficheros binarios con respecto a los ficheros de texto es que, en igualdad de condiciones, necesitan menos espacio para codificar la misma información.En cambio, tienen la desventaja de ser muy dependientes del formato empleado, necesitando programas informáticos específicos de cada formato para editarlos y manipularlos.

Es necesario establecer un formato de fichero para dar significado a las secuencias almacenadas con independencia de cómo se guarde la información. Es decir, hay que determinar la codificación de la información en el fichero.

Por ejemplo, unos formatos de fichero de texto muy utilizados son los que codifican la información según algún lenguaje basado en XML (eXtensible Markup Language). XML es un metalenguaje extensible de etiquetas que

permite definir la gramática de lenguajes específicos. Por lo tanto XML no es un lenguaje en particular, sino más bien una manera de definir lenguajes para diferentes necesidades, pudiendo usar programas de edición de XML o edición de texto para editarlos. Un caso concreto de un lenguaje basado en XML es SBML (Systems Biology Markup Language) que es empleado para representar modelos de procesos biológicos.

Una buena solución podría ser aportar la definición de un sistema P a través de uno o varios ficheros (ya sean de texto o binarios) debido, principalmente, a los siguientes puntos.

- Es una solución de *poco acoplamiento*: la definición de un sistema P a través de ficheros puede ser utilizada por diferentes aplicaciones informáticas, independientemente de la plataforma utilizada o el lenguaje de programación en el que fueron desarrolladas. El único requisito es conocer el formato con el cual se codifica el sistema P.

- Es una solución *reutilizable*: la misma definición de un sistema P puede ser utilizada en diferentes entornos de software ya que al quedar guardada la definición del sistema P en un soporte físico, no es necesario volver a crearla en sucesivas simulaciones.

- Es una solución con *buena usabilidad*: en la mayoría de los casos, es posible implementar interfaces de usuario para editar, guardar y cargar los ficheros con los cuales se definen los sistemas P a simular. De esta manera, se implementa una solución mixta entre la utilización de ficheros y la aplicación de interfaces de usuario, posibilitando disfrutar de las ventajas de usabilidad de estos últimos.

  También es posible delegar la edición de ficheros a terceras aplicaciones bien conocidas por los usuarios, especialmente en el caso de los formatos de ficheros de texto, tales como los basados en XML.

  En cualquier caso, la facilidad de aprendizaje del formato de codificación empleado será un factor determinante para conseguir una buena usabilidad.

3.1.2. Núcleo de simulación

El núcleo de simulación es la parte de la aplicación que se encarga de simular una o varias computaciones del sistema P definido.

En este módulo, se parte del supuesto de que el sistema P definido no contiene errores y es coherente con un determinado modelo. Para la consecución de este objetivo se delega en el módulo previo.

El núcleo de simulación ejecuta un determinado algoritmo de simulación que obtiene una o varias computaciones del sistema P simulado. En esta sección se realiza un análisis y clasificación de las estrategias usadas frecuentemente a la hora de diseñar el núcleo de simulación.

Atendiendo al nivel de paralelismo real, el núcleo de simulación podría ser:

- *Secuencial*: se ejecuta un algoritmo de simulación que está diseñado para correr sobre una única CPU, habitualmente consiste en un bucle que selecciona y ejecuta reglas en cada iteración.

- *Multi-hilo*: en programación, un hilo corresponde a una secuencia de código que puede ser ejecutada en paralelo. Dependiendo de la arquitectura del sistema, se producirá un paralelismo real o un paralelismo virtual. Las CPUs que no poseen ningún nivel de paralelismo real, ejecutan los hilos de manera secuencial por intervalos de tiempo; en cambio, las CPUs que poseen un cierto nivel de paralelismo real en su arquitectura permiten la ejecución de hilos en diferentes procesadores o núcleos.

- *Paralelo*: existen determinadas plataformas totalmente paralelas, como es el caso de los *clusters* de ordenadores, en donde cada ordenador conectado corresponde a un nodo de ejecución. Por otra parte, existe hardware paralelo que se puede programar utilizando determinados lenguajes específicos, entre otros, las GPUs (Graphics Proccesor Units) o el hardware reconfigurable basado en FPGAs (Field Programmable Gate Arrays). En cualquier caso, la programación de hardware paralelo no es

una tarea sencilla, ya que es necesario conocer bien las restricciones de la arquitectura con el fin de poder alcanzar, siempre que sea posible, un buen grado de paralelismo.

Según la diversidad de sistemas P que se pueden simular, el núcleo de simulación puede ser:

- Diseño para un sistema P concreto: se trata un simulador *ad hoc*. Esta solución suele ser empleada junto con la definición del sistema P en el código fuente y, además, en aquellos casos en los que únicamente interesa simular un sistema P en particular. Se trata de una solución poco flexible pero es la más sencilla y rápida de realizar.

- Diseño para un modelo de sistemas P: esta solución puede simular aquellos sistemas P que sigan la semántica de un modelo determinado. En este caso, el simulador recibe un sistema P definido por el módulo anterior y ejecuta la simulación de una o varias posibles computaciones siguiendo la semántica del modelo.

- Diseño para varios modelos de sistemas P. Es posible encontrar puntos en común entre diversos modelos; por ejemplo, el modelo de transición y el modelo symport/antiport sólo se diferencian en que el segundo es más restrictivo en las reglas permitidas, pues sólo se permiten reglas de comunicación con cooperación dependiente en cierto sentido. No obstante, desde el punto de vista de la simulación se podría utilizar el mismo núcleo de simulación.

3.1.3. Presentación de resultados al usuario

El núcleo de simulación reproduce, paso a paso, una de las posibles computaciones del sistema P definido, para lo cual se establecen estructuras de datos en la memoria de la máquina para almacenar las sucesivas configuraciones alcanzadas. Es posible almacenar todas las configuraciones por

las que pasa el simulador, o bien se puede optar por almacenar exclusivamente la última configuración generada.

En cualquier caso, es necesario extraer información de estas configuraciones con el fin de mostrarlas al usuario. La información mostrada dependerá principalmente de la finalidad del simulador.

- En el caso de los simuladores que fueron diseñados con motivos pedagógicos, será interesante mostrar la máxima información posible: la estructura de membranas, los multiconjuntos de objetos, las reglas ejecutadas en el último paso de computación simulado, etc. Esta información se podrá presentar al usuario de varias maneras, siendo la utilización de interfaces gráficas una de las más apropiadas.

- En el caso de los simuladores cuyo objetivo es la simulación de procesos de la vida real a través de sistemas P, deja de tener sentido el recopilar los detalles pormenorizados de la computación simulada. De hecho, es muy probable que el usuario final no esté familiarizado con el paradigma de *Membrane Computing*, sino que sea un experto en el proceso objeto de estudio a través de sistemas P. Para poder presentar los resultados de manera adecuada a este tipo de usuario, es necesario realizar un proceso de conversión entre la computación generada por el simulador y la información relevante que el usuario necesita.

La presentación de resultados al usuario depende mucho de la finalidad del simulador. En el caso de los simuladores con fines pedagógicos, son relevantes las propias configuraciones generadas por el simulador; en el caso de los simuladores que modelizan procesos de la vida real, la información relevante depende del proceso modelizado y deberá ser mostrada de manera comprensible para el tipo de usuario objetivo.

		Finalidad		Paralelismo			Plataforma			Modelo		
		Pedagógicos	Simulación de procesos reales	Secuenciales	Multi-hilo	Paralelismo real	CPU	Hardware específico	Redes de ordenadores	Sistemas P de transición	Membranas activas	Otros
2000	Simulador de Malita	X		X			X			X		
	Simulador de Suzuki y Tanaka	X	X	X			X			X		
2002	Simulador de Balbontín y otros	X		X			X			X		
	Simulador de Baranda y otros	X		X			X			X		
	Simulador de Ciobanu y Paraschiv	X			X		X				X	X
2003	Simulador de Ardelean y Cavaliere		X	X			X			X		
	Simulador de Ciobanu y Wenyuan	X			X	X			X	X		
	Simulador de Georgiou (*SubLP-Studio*)	X		X			X					
	Simulador de Syropoulos y otros	X			X	X			X			X
2004	Simulador de Nepomuceno (*SimCM*)	X			X		X				X	
	Simuladores del GCN	X		X			X				X	
	Simulador del GMNC (*PSim*)	X	X		X		X					X
2005	Simulador de Nishida	X		X			X					X
2006	Simuladores de Cazzaniga y Pescini		X	X			X					X
	Cyto-Sim: Biological compartment simulator		X	X			X					X
	Simulador de Frisco	X	X	X			X					X
	Simuladores de Romero-Campero y M. Gheorghe		X	X			X					X
2007	Simulador de Acampora y Loia	X			X		X			X		
	Simulador de Borrego-Ropero y otros	X			X		X					X
	Simulador de Petreska y Teuscher	X				X		X				X
	Simulador de Ramírez-Martínez y Gutiérrez-Naranjo	X			X		X					X
2008	Simulador de Ribero y otros (*JPlant*)	X			X		X				X	
2009	Simulador de Martínez-del-Amor y otros	X				X		X			X	
	Simulador de Nguyen y otros	X				X		X		X		
	Simulador de Castellini y otros (*Meta-Plab*)	X	X	X			X					X

Tabla 3.1: Listado de simuladores de sistemas P

3.2. Clasificación de los simuladores existentes

En esta sección se trata de analizar brevemente los simuladores de sistemas P desarrollados desde el año 2000 hasta la fecha.

La mayoría de estos simuladores se pueden descargar de la sección de software de la página web de los *P systems* `http://ppage.psystems.eu/`.

En el Cuadro 3.1 se muestra un listado de simuladores clasificados según su finalidad, nivel de paralelismo real, plataforma para la cual fueron desarrollados y modelo de sistemas P que simulan.

3.2.1. Año 2000

Simulador de Maliţa

El primer simulador de sistemas P fue presentado por Mihaela Maliţa [72]. Es un programa escrito en LPA-Prolog para la simulación de sistemas P de transición.

El simulador recibe como entrada la configuración inicial de un sistema P, su conjunto de reglas y un parámetro que especifica el número de pasos a simular. En cada paso, por cada membrana, se selecciona solamente una regla que es aplicada tantas veces como es posible. La salida muestra la secuencia de configuraciones de una de las posibles computaciones hasta alcanzar el número deseado de pasos.

Simulador de Suzuki y Tanaka

En el mismo año, Yasuhiro Suzuki e Hiroshi Tanaka presentaron en [111] un programa escrito en Lisp para la simulación de sistemas P de transición.

Para controlar la cantidad de recursos necesarios, se impuso una importante restricción: El tamaño máximo de los multiconjuntos utilizados está limitado.

A pesar de que este simulador no fue inicialmente desarrollado con la

intención de reproducir problemas de la vida real, ha sido exitosamente utilizado para simular el modelo Brusselator o en la modelización y análisis de sistemas ecológicos (ambos y más en [112]).

3.2.2. Año 2002

Simulador de Balbontín y otros

Dos años después, Delia Balbontín, Mario J. Pérez y Fernando Sancho presentaron durante el *Workshop on Membrane Computing 2002* un simulador [8] para sistemas P de transición escrito en MzScheme.

Este simulador, como el de Maliţa, recibe como entrada la configuración inicial de un sistema incluyendo el conjunto de reglas y una serie de parámetros especificando el número deseado de pasos de evolución, pero presenta como novedad importante que proporciona como salida el árbol de computación del sistema P, paso a paso, hasta alcanzar el número deseado de pasos de evolución. Evidentemente, cuanto mayor sea la ramificación del árbol de computación, menor será el número de pasos que se puedan simular.

Simulador de Baranda y otros

En los primeros años del desarrollo de la teoría de los sistemas P, algunos miembros del Grupo de Computación Natural de la Universidad Politécnica de Madrid [122] propusieron algunos *frameworks* y estructuras de datos para sistemas P ([5, 6, 10, 9]). En [7], basándose en dicha formalización previa, se presentó un simulador escrito en Haskell para sistemas P de transición implementado por A. Baranda.

El simulador recibe como entrada un fichero que codifica un sistema (configuración inicial y reglas en cada región) y produce otro fichero codificando el sistema obtenido por la aplicación de un paso de computación (a través de un multiconjunto maximal de reglas seleccionadas aleatoriamente).

Simulador de Ciobanu y Paraschiv

En [29] Gabriel Ciobanu y Dorin Paraschiv presentaron una aplicación de software, desarrollada en Visual C++ con MFC, que proporcionó una simulación de la versión inicial de los sistemas celulares catalíticos jerarquizados, así como para los sistemas P con membranas activas (ver [85, 86]).

El sistema se presenta al usuario mediante una interfaz gráfica donde la pantalla principal se divide en dos ventanas: la ventana de la izquierda ofrece una representación en árbol del sistema de membranas, incluyendo objetos y membranas; y la ventana de la derecha proporciona una representación gráfica del sistema de membranas mediante diagramas de Venn. Un menú permite añadir nuevos objetos, membranas, reglas y prioridades. Usando las funcionalidades de *Start*, *Next* y *Stop*, los usuarios pueden observar la evolución del sistema paso a paso.

3.2.3. Año 2003

Simulador de Ardelean y Cavaliere

Una de las primeras herramientas para la modelización de procesos biológicos con sistemas P fue presentada en [3]. En este trabajo, se describe una aplicación informática que se puede entender como un simulador para sistemas P de transición sin disolución ya que el número de membranas no varía durante la computación. Siendo más precisos, el software trabaja con una variante especial de sistemas P (propuesta en [23]), en donde las reglas permitidas son de reescritura y de tipo *symport/antiport*.

Los autores intentan enlazar el modelo matemático con la realidad biológica, indicando cómo se puede usar el marco de los sistemas P para modelizar procesos que tienen lugar dentro de las células. En este sentido, varios procesos biológicos han sido simulados con este software (ver [24]).

Simulador de Ciobanu y Wenyuan

Gabriel Ciobanu y Guo Wenyuan presentaron en [30] una implementación paralela de sistemas P de transición diseñada para un cluster de ordenadores. Fue escrita en C++ y hacía uso del *Interfaz de Paso de Mensajes (MPI)* [133] como mecanismo de comunicación.

El programa está implementado y probado en un cluster de 64 nodos duales de procesamiento en la National University of Singapore.

Simulador de Georgiou (*SubLP-Studio*)

En septiembre de 2003, Alexandros Georgiou de la Universidad de Sheffield presentó un simulador llamado *SubLP-Studio* [44, 45] que es un software para el modelo de sistemas Sub LP, una variante de sistemas L y sistemas P. De manera opcional, proporcionaba una interfaz a *cpfg*, por lo que era posible producir dibujos de plantas usando el intérprete *turtle*. Se trata de la primera herramienta capaz de generar gráficos por ordenador que aparece en el ámbito de *Membrane Computing*.

Simulador de Syropoulos y otros

En el año 2003, Apostolos Syropoulos *et al.* presentaron en [114] una simulación puramente distribuida de sistemas P. Está implementada usando RMI (*Invocación Remota de Métodos*) de Java para conectar un número de ordenadores que intercambian datos. Como los autores apuntan, la idea de diseñar un simulador distribuido para una red de ordenadores –en vez de realizarlo sobre una arquitectura cluster– evita problemáticas con la compatibilidad de hardware.

3.2.4. Año 2004

Simulador de Nepomuceno (*SimCM*)

Siguiendo el propósito pedagógico, en [82], se pueden encontrar la descripción de una aplicación software, *SimCM*, escrita en Java. La clave de este simulador es la manera de mostrar la simulación al usuario: esta herramienta software permite seguir la evolución de un sistema P de transición de forma visual.

Simuladores del GCN

El Grupo de Investigación en Computación Natural de la Universidad de Sevilla (GCN) [123] presentó en el año 2004 dos simuladores para sistemas P con membranas activas que fueron concebidos como herramientas de asistencia al diseño y la verificación formal de soluciones celulares uniformes de problemas **NP**-completos [90, 91] mediante sistemas P reconocedores [89, 103]. En este caso, como sólo se consideran sistemas P confluentes, es suficiente simular una rama del árbol de computaciones.

El primero de estos simuladores que trabajan con sistemas P de membranas activas está escrito en CLIPS y fue presentado en [89].

El segundo simulador fue presentado por Cordón-Franco *et al.* (ver [32] y [33]). Fue escrito en Prolog y usado con éxito como asistente en el diseño de sistemas P para resolver problemas **NP**-completos (ver [34, 32, 33, 57, 103]).

Simulador del GMNC

El GMNC (Group for Models of Natural Computing [121]) en Verona ha desarrollado Psim, un simulador desarrollado en Java para sistemas P Metabólicos, introducido por primera vez en [13].

Las características principales son las siguientes: (a) la definición de la estructura de membranas y el conjunto de reglas mediante ficheros XML; (b)

una interfaz de usuario amigable; y (c) la posibilidad de salvar resultados intermedios que puedan ser cargados de nuevo. La salida del simulador consiste en una serie de grafos representando la multiplicidad de los objetos del sistema a través del tiempo.

3.2.5. Año 2005

Simulador de Nishida

En [84], Taishin Y. Nishida inició una nueva línea de investigación presentando una simulación de algoritmos de aproximación para resolver problemas de optimización **NP**-completos. Dichos algoritmos lo denomina *algoritmo de membranas* y una implementación del mismo para resolver el problema del *viajante de comercio (TSP)* fue realizada en el lenguaje Java.

3.2.6. Año 2006

Simuladores de Cazzaniga y Pescini

Paolo Cazzaniga y Dario Pescini han utilizado la *GNU Scientific Library* [124] y el lenguaje de programación C para desarrollar dos simuladores que modelizan procesos reales.

El primer simulador reproduce el sistema de regulación de genes de la bacteria Vibrio Fischeri (Quorum Sensing) usando el algoritmo multi-compartimental de Gillespie, desarrollado por F.J. Romero-Campero y otros en [106],

El segundo simulador es muy similar y permite simular computaciones de sistemas P probabilísticos dinámicos, definidos de una manera muy diferente a la que se realiza en esta memoria.

El código fuente de ambos simuladores está disponible en la sección de software de la página de los sistemas P [126].

Cyto-Sim: Biological compartment simulator

En [109] fue presentado un nuevo software para simulación de procesos biológicos a nivel micro y nivel macro. Se trata de un simulador estocástico que utiliza una estructura jerarquizada de membranas para modelizar ciertos procesos biológicos, donde las membranas se componen de una capa interna, otra externa y una intermedia. Cyto-Sim soporta redes de Petri y puede recibir la entrada de datos en formato SBML.

Simulador de Frisco

En el año 2002, P. Frisco y S. Ji presentaron un modelo de sistemas P denominado *conformon P systems* [42]. En el año 2006, se presentó el primer simulador para *conformon P systems* [41]. La herramienta ha sido aplicada a varios procesos biológicos, tales como la dinámica del virus HIV [35].

Simuladores de Romero-Campero y M. Gheorghe

F.J Romero-Campero y M. Gheorghe presentaron dos simuladores desarrollados en la Universidad de Sheffield para la simulación de procesos biológicos con sistemas P. En ambos, los autores implementan el algoritmo multi-compartimental de Gillespie en Scilab y C, respectivamente (ver [127]), incluyendo la traducción de sistemas P a PRISM para realizar la validación de los modelos.

Los simuladores han sido utilizados con éxito para estudiar varios procesos biológicos, tales como la simulación de rutas asociadas al factor de crecimiento epidérmico (EGFR) [92], simulación del mecanismo de apoptosis mediatizado por FAS [28], modelización del control de expresión de genes [105], o el desarrollo de un primer modelo computacional del fenómeno del *quorum sensing*; en particular, relativo a colonias de bacterias Vibrio fischeri [50].

3.2.7. Año 2007

Simulador de Acampora y Loia

G. Acampora y V. Loia presentan en [1] una aplicación paralela y distribuida para la simulación basada en multi-agentes de sistemas P de transición.

Simulador de Borrego-Ropero y otros

En [17] se presentó un simulador llamado *Tissue Simulator* para sistemas P de tejido con división de células que sean reconocedores. La herramienta incluye una interfaz gráfica de usuario para la especificación del sistema P de tejido a simular, y permite la simulación de una posible computación, mostrando gráficamente la computación generada con las reglas aplicadas en cada paso.

Ha sido utilizado como asistente en el diseño de soluciones a problemas **NP**-completos utilizando sistemas P de tejido reconocedores (ver, por ejemplo, [36]).

Simulador de Petreska y Teuscher

Petreska y Teuscher presentan en [96] una implementación basada en hardware paralelo, que permite ejecutar simulaciones de ciertas clases de sistemas P de forma relativamente eficiente. El código fuente de la implementación y más información están disponibles en [120].

Simulador de Ramírez-Martínez y Gutiérrez-Naranjo

En [55, 56] fue presentado el primer simulador para sistemas P neuronales de impulsos. Su objetivo es dar una información exhaustiva al usuario sobre el proceso computacional. De esta manera, se puede considerar como un asistente para la verificación de tales sistemas. La herramienta devuelve el diagrama de transición, paso a paso, de un sistema dado. El código es

modular y suficientemente flexible con la finalidad de ser adaptado para futuras aplicaciones.

3.2.8. Año 2008

Simulador de Ribero y otros (*JPlant*)

En sistemas P con creación de membranas, las nuevas membranas pueden ser creadas dentro de membranas existentes y esto produce una expansión de la estructura con nuevas ramas (el árbol que determina la estructura de membranas puede incrementar su profundidad). De esta manera, la estructura de membranas puede ser representada como un árbol que evoluciona en el tiempo con una longitud y anchura de ramas que puede crecer de forma similar a las plantas reales.

En [104], los autores presentan una aplicación que procesa las primeras configuraciones de una computación y dibuja el gráfico correspondiente. Este software es útil para la investigación experimental de la representación gráfica de sistemas P y establece un nuevo puente entre los sistemas L y los sistemas P.

3.2.9. Año 2009

Simulador de Martínez-del-Amor y otros

En [25, 26, 27] se presentan los primeros simuladores de sistemas P basados en GPUs (*Graphic Processor Units*). La arquitectura de las tarjetas gráficas ha evolucionado de tal manera que presentan un paralelismo real comparable a un cluster de 240 procesadores. Utilizando el lenguaje de programación CUDA y la tarjeta Nvidia Tesla C1060, los autores presentan un simulador para sistemas P reconocedores con membranas activas y reglas de división, consiguiendo ejecutar simulaciones de sistemas P que resuelven instancias de problemas **NP**-completos de forma estrictamente más eficiente que en simuladores

secuenciales. El simulador implementa paralelismo real y reproduce una de las posibles computaciones del sistema P utilizando exclusivamente la tarjeta gráfica como co-procesador paralelo. Cabe decir que estos simuladores definen el sistema P que va a ser simulado mediante la utilización del lenguaje de programación P-Lingua, desarrollado en esta memoria.

Simulador de Nguyen y otros

En [83] se presenta una prometedora investigación sobre la simulación de sistemas P implementando paralelismo real sobre hardware reconfigurable (FPGAs). Con la finalidad de producir el código que debe ser cargado en las FPGAs, dos herramientas de software son proporcionadas: *Reconfig-P* y *P-Builder*. El trabajo presenta la primera herramienta para la configuración automática de hardware con la finalidad de simular computaciones de sistemas P de transición.

Simulador de Castellini y otros (*MetaPlab*)

En [22] se presenta una actualización del simulador PSim, añadiendo flexibilidad y capacidad de integración con otras herramientas gracias a una arquitectura basada en pluggins.

La aplicación presenta una amigable interfaz gráfica de usuario que permite definir el sistema P metabólico que va a ser simulado mediante objetos gráficos. Para más información, se puede descargar junto con su código fuente de la página web de los sistemas P [126].

Capítulo 4

Un entorno de programación para *Membrane Computing*

La mayoría de las aplicaciones informáticas usadas en *Membrane Computing* son simuladores que comparten una serie de elementos en común y, por tanto, sus programadores han tenido que enfrentarse a problemas comunes de diseño, tales como

- Estrategias para definir los sistemas P a simular.
- Algoritmos de simulación que reproduzcan computaciones de sistemas P.
- Mecanismos para procesar las computaciones simuladas y presentar los datos relevantes al usuario.

Habitualmente, cada simulador aporta soluciones específicas a estos problemas, lo cual provoca una escasa reutilización de código y dificulta la adaptación de los usuarios a diferentes entornos de software.

En este capítulo se proponen soluciones generales a estos problemas de diseño, con la intención de facilitar el desarrollo de futuras aplicaciones y su utilización por parte de los usuarios.

En la primera sección se presenta un lenguaje de programación, que denominamos P-Lingua, como mecanismo estándar para la definición de

sistemas P en ficheros de texto. De esta manera, se pueden reutilizar esos mismos ficheros en diferentes aplicaciones informáticas, mejorando el tiempo de adaptación del usuario a una nueva aplicación, así como el coste de desarrollo de nuevos simuladores. La sintaxis de P-Lingua se ilustra a traves de algunos ejemplos de código. Más concretamente, en la sección 4.2 se muestra la definición de un sistema P de transición y en la siguiente sección se describe la implementación en P-Lingua de dos soluciones eficientes de un problema **NP**-completo: `SAT`.

El lenguaje P-Lingua se acompaña de una serie de bibliotecas y herramientas que constituyen un entorno de programación para *Membrane Computing*, En la sección 4.4 se presenta una biblioteca desarrollada en el lenguaje de programación Java [128], *pLinguaCore*, para el procesamiento y simulación de ficheros que definen sistemas P. Dicha biblioteca permite realizar

- Lectura y análisis de ficheros de texto que definen sistemas P, ya sea en formato P-Lingua o en otros. La biblioteca es capaz de detectar y localizar los errores léxico/sintácticos y semánticos.

- Simulación paso a paso de computaciones de los sistemas P definidos. Para cada modelo de sistemas P soportado se diseñan uno o más algoritmos de simulación.

- Exportación de los sistemas P definidos a otros formatos de fichero, permitiendo interoperabilidad entre distintos simuladores.

En la siguiente sección se explican los algoritmos de simulación utilizados en la biblioteca. Y, finalmente, en la sección 4.6 se presentan dos aplicaciones basadas en pLinguaCore para ser ejecutadas desde la línea de comandos: (a) un compilador que puede traducir la definición de un sistema P de un formato de fichero a otro distinto (incluyendo P-Lingua como formato de entrada); y (b) un simulador que reproduce computaciones de los sistemas P definidos.

4.1. P-Lingua: un estándar para la definición de sistemas P

Tras el análisis realizado en el capítulo anterior, se deduce que la definición de sistemas P mediante ficheros de texto es una buena solución que proporciona

- bajo acoplamiento, ya que la especificación en ficheros de texto es independiente del programa que los gestiona;
- reutilización, debido a que una misma definición de un sistema P puede ser utilizada por diferentes aplicaciones; y
- usabilidad, gracias a la diversidad de programas de edición de texto existentes.

Los ficheros de texto que definen sistemas P deben seguir algún formato; la mayoría de ellos están diseñados para simuladores concretos, lo cual repercute negativamente en el tiempo y el esfuerzo que el programador requiere para desarrollar nuevos simuladores, así como en el tiempo y el esfuerzo que el usuario final necesita para asimilar los formatos de fichero de diferentes simuladores.

En esta memoria se presenta un lenguaje de programación, *P-Lingua*, como estándar para la codificación de sistemas P en ficheros de texto. De esta manera, se pueden reutilizar los mismos ficheros de texto que definen sistemas P en diferentes aplicaciones informáticas, mejorando el tiempo de adaptación del usuario a una nueva aplicación y, mediante el desarrollo de bibliotecas de programación que procesen el formato estándar, se consigue mejorar el coste de desarrollo de nuevos simuladores.

La comunidad científica en Computación celular con membranas está formada por un grupo heterogéneo de investigadores, desde matemáticos e informáticos hasta biólogos, ingenieros, físicos y ecólogos. Este grupo interdisciplinar comparte el lenguaje científico en el que se especifica los sistemas P y, por tanto, se hace necesario el desarrollo de un estándar

orientado a esta comunidad que debería tener ciertas similitudes con el lenguaje utilizado tradicionalmente para especificar sistemas P. De esta manera, se reduciría la dificultad de aprendizaje y se mejoraría la usabilidad, permitiendo a los usuarios escribir en un lenguaje que les resulte familiar, dentro de las limitaciones inherentes a escribir en texto plano.

Por otra parte, la existencia de elementos comunes en soluciones propuestas a diferentes problemas numéricos **NP**-completos utilizando familias de sistemas P reconocedores con membranas activas ha permitido realizar una primera aproximación al desarrollo de un lenguaje de programación celular basado en subrutinas o módulos [58]. La especificación de sistemas P de manera modular presenta una serie de ventajas tales como la mejor comprensión de los programas escritos, la elegancia de código y la estructuración en módulos funcionales que se corresponden con secciones o conjuntos de reglas que se pueden utilizar repetidas veces en el la ejecución del programa.

El lenguaje de programación P-Lingua permite definir sistemas P pertenecientes a diferentes modelos o variantes de manera sencilla, pues su sintaxis está basada en la notación científica usada por los investigadores: por una parte, paramétrica, permitiendo definir familias de sistemas P mediante el uso de índices y parámetros; y por otra, modular, atendiendo a las ideas expuestas en [58].

Finalmente, los programas escritos en P-Lingua pueden servir de entrada para diversas aplicaciones informáticas o, mediante el uso de compiladores, pueden ser traducidos a otros formatos de especificación, aportando interoperabilidad entre simuladores, tal como se ilustra en la figura 4.1.

4.1.1. Sintaxis del lenguaje P-Lingua

La sintaxis de P-Lingua es suficientemente amplia como para definir una gran variedad de tipos/modelos/variantes de sistemas P. Sin embargo, al principio de cada fichero P-Lingua se debe especificar el modelo de sistemas P que se está utilizando, de tal manera que el compilador puede detectar errores

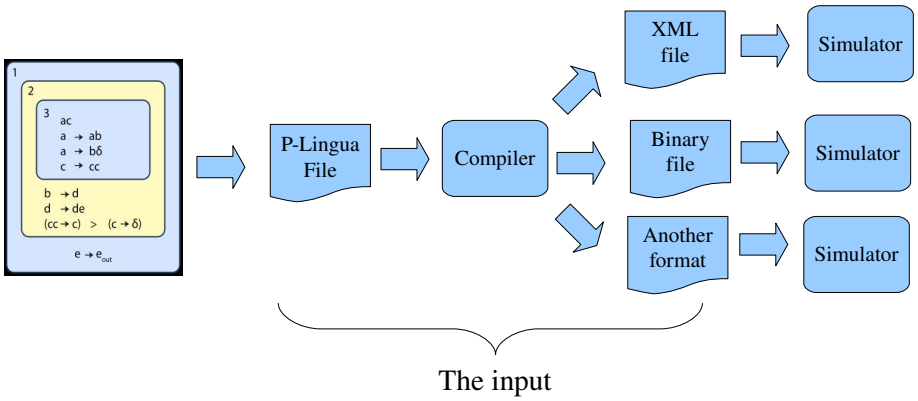


Figura 4.1: Interoperabilidad utilizando P-Lingua

semánticos (desde el punto de vista de la programación) cuando el sistema P escrito no satisface las restricciones del modelo especificado.

En la actualidad, en el marco de P-Lingua actualmente se pueden definir sistemas P pertenecientes a los siguientes modelos:

- Sistemas P que trabajan a modo de célula (*cell-like*):
  - Sistemas P de transición.
  - Sistemas P symport/antiport.
  - Sistemas P con membranas activas y reglas de división.
  - Sistemas P con membranas activas y reglas de creación.
  - Sistemas P estocásticos.
- Sistemas P que trabajan a modo tejido (*tissue-like*):
  - Sistemas P de tejido con reglas symport/antiport y reglas de división.
  - Sistemas P probabilísticos multientorno.

Ampliar el lenguaje para soportar más modelos es uno de los retos que tenemos planteados y que proponemos como una de las líneas futuras de investigación en el capítulo 9.

A continuación se presenta la sintaxis de P-Lingua.

Identificadores válidos

Se dice que una sucesión finita de caracteres forma un *identificador válido* si no comienza por un carácter numérico, no es una palabra reservada y está compuesta por caracteres de entre los siguientes:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _
```

Los identificadores válidos son ampliamente utilizados en el lenguaje: para definir nombres de módulos, parámetros, índices, etiquetas de membranas, objetos del alfabeto y cadenas.

Las siguientes cadenas de texto `call`, `@ceil`, `@d`, `@debug`, `def`, `@floor`, `let`, `@log`, `main`, `@model`, `@ms`, `@mu`, `@round`, `-->` , `<-->` , `#` se denominan *palabras reservadas* del lenguaje.

Variables

En P-Lingua se permiten cuatro tipos de variables: *Variables globales*, *Variables locales*, *Índices* y *Parámetros*.

Las variables se usan para almacenar valores numéricos y sus nombres deben ser identificadores válidos. Se usan 64 bits con signo en doble precisión.

Variables globales

Las variables globales deben ser declaradas fuera de cualquier módulo (ver definición de módulos más adelante) y a ellas se puede acceder desde cualquiera

de los módulos del programa P-Lingua. El nombre de una variable global `global_variable_name` debe ser un identificador válido. La sintaxis para definir una variable global es la siguiente:

```
global_variable_name = numeric_expression;
```

Variables locales

Las variables locales siempre se definen dentro de módulos, y sólo pueden ser usadas en el ámbito del módulo en el que están definidas. El nombre de una variable local `local_variable_name` debe ser un identificador válido. La sintaxis para definir una variable local es la siguiente:

```
let local_variable_name = numeric_expression;
```

Los índices y parámetros se consideran variables locales.

Variables con índices

Se pueden usar índices numéricos en la definición de variables, ya sean globales o locales. Los índices se representan por expresiones numéricas que se escriben entre llaves a la derecha del nombre de la variable, por ejemplo:

- `b{1} = 12;`
- `i = 1;`
- `x{b{i},7}=10;`

Identificadores para cargas eléctricas

En P-Lingua, se pueden considerar cargas eléctricas usando los símbolos `+` y `-` para cargas positivas y negativas, respectivamente. La carga neutra será representada por defecto mediante la omisión de los símbolos de cargas.

Etiquetas de membranas

Hay tres maneras de escribir etiquetas de membranas en P-Lingua: la primera es utilizar un número natural; la segunda es determinar la etiqueta como un identificador válido y la tercera es utilizar expresiones numéricas entre paréntesis que representen números naturales.

Identificadores de entornos

Cuando se definen sistemas P multientorno, se pueden escribir identificadores de entornos de la misma manera que se definen etiquetas de membranas.

Expresiones numéricas

Se pueden escribir expresiones numéricas usando los operadores `*` (multiplicación), `/` (división), `%` (módulo), `+` (adición), `-` (substracción), `^` (potencia), `@log` (logaritmo en base 2), `@ceil` (mayor entero más cercano) y `@floor` (menor entero más cercano) sobre números enteros o reales y variables, permitiendo el uso de paréntesis. Es posible escribir números usando notación exponencial.

Ejemplos de expresiones numéricas:

- $3 * 10^{-5}$ se escribe `3e-5`.
- $\frac{3^{4-x}+17}{23}$ se escribe `(3^(4-x)+17)/23`.
- $7\lceil log_2 x \rceil$ se escribe `7*(@ceil @log x)`.

Objetos

Los objetos del alfabeto de un sistema P se escriben usando identificadores válidos, estando permitido incluir índices entre paréntesis. Por ejemplo, $x_{i,2n+1}$ y Yes se escriben como `x{i,2*n+1}` y `Yes` respectivamente.

La multiplicidad de un objeto se representa usando el operador `*`. Por ejemplo, x_i^{2n+1} se escribe como `x{i}*(2*n+1)`.

Cadenas

Las cadenas se escriben entre los caracteres `<` y `>` y se construyen mediante la concatenación de identificadores válidos y el símbolo `.`; es decir, `<identifier1. ... .identifierN>`. Por ejemplo, `<cap.RNAP.op>`.

Subcadenas

Las subcadenas son usadas en reglas de reescritura de cadenas y la sintaxis es similar a la de las cadenas, pero es posible utilizar el carácter `?` para representar una secuencia arbitraria (que puede ser vacía) de identificadores válidos concatenados por el símbolo `.` (punto). Por ejemplo, `<cap.?.NAP.op>` es una subcadena de la cadena `<cap.op.op.op.NAP.op>` y también de la cadena `<cap.NAP.op>`.

Especificación del modelo utilizado

Como el lenguaje P-Lingua soporta más de un modelo de sistemas P, es necesario especificar al principio del fichero qué modelo se está utilizando. Cada modelo incluye una serie de restricciones; por ejemplo, las reglas de creación de membranas no se permiten en sistemas P symport/antiport. En estos casos, el compilador de P-Lingua posee un analizador que detecta e identifica tales errores. El modelo utilizado se especifica usando la sentencia `@model<model_name>` al principio del fichero. Los modelos permitidos son:

```
@model<membrane_division>

@model<membrane_creation>

@model<transition_psystem>
```

```
@model<probabilistic_psystem>

@model<stochastic_psystem>

@model<symport_antiport_psystem>

@model<tissue_psystems>
```

Definición de módulos

La sintaxis para definir un módulo es la siguiente.

```
def module_name(param1,..., paramN)
{
  sentence0;
  sentence1;
  ...
  sentenceM;
}
```

El nombre de un módulo, `module_name`, debe ser un identificador válido y no pueden existir dos módulos con el mismo nombre. Los parámetros del módulo deben ser identificadores válidos y no pueden aparecer repetidos. Es posible definir módulos sin parámetros. Cada parámetro recibe un valor numérico que es asignado en la llamada al módulo (ver más abajo).

Todos los programas escritos en P-Lingua deben contener un módulo `main` sin parámetros. Para comenzar a procesar el fichero de entrada, el compilador buscará ese módulo principal.

Un módulo está estructurado en sentencias. En P-Lingua existen sentencias para definir la estructura de membranas de un sistema P, para especificar

multiconjuntos, para definir reglas, para definir variables y para llamar a otros módulos. A continuación, se muestra como se escriben estas sentencias.

Llamadas a módulos

En P-Lingua, los módulos son ejecutados a través de llamadas. El formato de una sentencia que llama a un módulo con algunos valores concretos para sus parámetros se da a continuación:

```
call module_name(value1, ..., valueN);
```

donde `value`i es una expresión numérica o una variable.

Definición de la estructura inicial de membranas

Para definir la estructura inicial de membranas de un sistema P, se escribe la siguiente sentencia:

```
@mu = expr;
```

donde `expr` es una secuencia de corchetes representando la estructura de membranas, incluyendo algunos identificadores para especificar la etiqueta y la carga eléctrica de cada membrana.

Ejemplos:

1. $[[\,]^{0}_{2}]^{0}_{1} \equiv$ `@mu = [[]'2]'1`

2. $[[\,]^{0}_{b}[\,]^{-}_{c}]^{+}_{a} \equiv$ `@mu = +[[]'b, -[]'c]'a`

En sistemas P probabilísticos (`@model<probabilistic>`), es posible definir varios entornos. Para ello se incluyen identificadores de entorno en la estructura inicial, tal como se muestra en el siguiente ejemplo:

```
@mu = [ [ []'2 ]'1 ]'101,101 [ [ []'2 ]'1]'102,102]'global;
```

Los entornos de un sistema P multientorno son considerados en P-Lingua como membranas especiales de un sistema P que trabaja a modo de célula (*cell-like*). Por este motivo, es necesario incluir una membrana piel; en el ejemplo ha sido incluida con la etiqueta `global`.

Las membranas que representan entornos incluyen etiqueta de membrana e identificador de entorno. En el ejemplo existen dos membranas que corresponden, respectivamente, a la membrana etiquetada 101 dentro del entorno 101 y a la membrana etiquetada 102 dentro del entorno 102.

Todas las membranas que se encuentran dentro de una membrana que representa un entorno heredan el mismo identificador de entorno (es decir, el compilador entiende que están en el mismo entorno que el representado por la membrana que las contiene). En el ejemplo, ambos entornos contienen el mismo esqueleto `[ []'2 ]'1`.

Cuando se utilizan números para representar etiquetas y entornos, se establece por convenio que las etiquetas de membranas comienzan la numeración en 1 y los identificadores de entorno comienzan en 101 o en 1001, dependiendo del número de membranas en el sistema P.

Definición de las células iniciales de un sistema P de tejido

Cuando se define un sistema P de tejido (es decir, el fichero comienza por `@model<tissue_psystems>`), en lugar de definir la estructura inicial de membranas, se deben definir las células iniciales del sistema. La siguiente sentencia cumple este propósito:

```
@mu = [[ ]'1 ...[ ]'q]'0;
```

donde se definen q células dentro de un entorno etiquetado con 0. Por ejemplo, para un sistema P de tejido con 2 células: `@mu = [[ ]'1 [ ]'2]'0;`

Definición de multiconjuntos

La siguiente sentencia define el multiconjunto inicial asociado a la membrana etiquetada `label`.

```
@ms(label) = multiset_of_objects;
```

donde `label` es una etiqueta de membrana y `multiset_of_objects` es un multiconjunto de objetos separados por comas, pudiendo indicar la multiplicidad de los objetos con el operador `*`. El carácter `#` se usa para representar el multiconjunto vacío. Con esta sentencia, todas las membranas etiquetadas con `label` reciben el multiconjunto inicial. Por ejemplo:`@ms(2) = a1*10,b;`

En la definición de un sistema P estocástico (es decir, el fichero empieza con la línea `@model<stochastic>`), las cadenas están permitidas en el contenido inicial de las membranas:

```
@ms(label) = multiset_of_objects_and_strings;
```

En sistemas P de tejido, la etiqueta del entorno puede ser utilizada para definir el alfabeto de Ω, cuyos elementos son procesados como si tuvieran multiplicidad infinita (sin necesidad de añadir ningún símbolo adicional). Por ejemplo: `@ms(0) = a,b,c;`

En la definición de un sistema P probabilístico, se permite indicar de manera opcional el entorno en el cual se encuentra la membrana cuyo multiconjunto se está referenciando:

```
@ms(label,environment) = multiset_of_objects;
```

Por ejemplo, `@ms(2,101) = a,b*12;` asigna el multiconjunto `a,b*12` a la membrana etiquetada con 2 del entorno 101. No obstante, también es válida la expresión `@ms(2) = c,d*16;` en sistemas P probabilísticos que permite asignar el multiconjunto `c,d*16` a todas las membranas etiquetadas con 2.

Unión de multiconjuntos

P-Lingua permite definir la unión de dos multiconjuntos usando la siguiente sentencia:

```
@ms(label) += multiset_of_objects;
```

En este caso, el multiconjunto de objetos definido se añade al multiconjunto existente en la membrana etiquetada con `label`.

Para sistemas P estocásticos:

```
@ms(label) += multiset_of_objects_and_strings;
```

Para sistemas P probabilísticos, está permitida también la siguiente sentencia:

```
@ms(label,environment) += multiset_of_objects;
```

Definición de reglas

Como se ha mencionado previamente, cada modelo de sistemas P permite sólo unos tipos de reglas. A continuación se enumeran los diferentes tipos de reglas que están permitidos en los diferentes modelos que se consideran.

`@model<mebrane_division>`

1. Se pueden definir reglas de evolución del tipo $[a \rightarrow v]_h^\alpha$ de las siguientes maneras:
   - α`[a --> v]'h;`
   - α`[a]'h -->` α`[v]'h;`
   - α`[a]'h --> [v];`
2. Se pueden definir reglas de comunicación (*send-in*) del tipo $a[\,]_h^\alpha \rightarrow [b]_h^\beta$ de las siguientes formas:

- aα[]'h --> β[b]'h;
- aα[]'h --> β[b];

3. Se pueden definir reglas de comunicación (*send-out*) del tipo $[a]_h^\alpha \rightarrow b[\,]_h^\beta$ de las siguientes maneras:
   - α[a]'h --> bβ[]'h;
   - α[a]'h --> bβ[];
   - α[a]'h --> β[]b;
4. Se pueden definir reglas de división del tipo $[a\,]_h^\alpha \rightarrow [b]_h^\beta[c]_h^\gamma$ de las siguientes formas:
   - α[a]'h --> β[b]'hγ[c]'h;
   - α[a]'h --> β[b]γ[c];
5. El formato para definir reglas de disolución del tipo $[a]_h^\alpha \rightarrow b$ es el siguiente:

   α[a]'h --> b;

donde a, b y c son objetos; v es un multiconjunto de objetos; h es una etiqueta; y α y β son identificadores de cargas eléctricas.

Algunos ejemplos:

- $[x \rightarrow y^{10}z]_1^+ \equiv$ +[x --> y*10,z]'1;
- $[s_1 \rightarrow r_{3,2}s]_h^- \equiv$ -[s{1}]'h --> -[r{3,2},s]'h;
- $[x \rightarrow \lambda]_2^0 \equiv$ [x]'2 --> [#];
- $[x_{i,1} \rightarrow r_{i,1}^4]_2^+ \equiv$ +[x{i,1} --> r{i,1}*4]'2;
- $d_k[\,]_2^0 \rightarrow [d_{k+1}]_2^0 \equiv$ d{k}[]'2 --> [d{k+1}];
- $[d_k]_2^+ \rightarrow []_2^0 d_k \equiv$ +[d{k}]'2 --> []d{k};

- $[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- \equiv$ `[d{k}]'2 --> +[d{k}]-[d{k}];`
- $[a]_2^- \rightarrow b \equiv$ `-[a]'2 --> b;`

`@model<membrane_creation>`

1. Las reglas del tipo 1, 2, 3 y 5 de `@model<membrane_division>` deben ser escritas con el mismo formato.
2. Se pueden definir reglas de creación de membranas del tipo $[a]_h^\alpha \rightarrow [[b]_{h_1}^\beta]_h^\alpha$ de las siguientes maneras:
   - α`[a]'h -->` α`[`β`[b]'`h_1`]'h;`
   - α`[a]'h -->` α`[`β`[b]'`h_1`];`

donde a y b son objetos; h y h_1 son etiquetas de membranas; y α y β son identificadores de cargas eléctricas. Por ejemplo:

$[d_k]_2^0 \rightarrow [[d_{k+1}]_3^-]_2^0 \equiv$ `[d{k}]'2 --> [-[d{k+1}]'3]'2;`

`@model<transition_psystem>`

1. Las reglas del tipo $u[v]_h \rightarrow u_1[v_1]_h$ pueden ser escritas de las siguientes formas:
   - `u [v]'h -->` u_1 `[`v_1`]'h;`
   - `u [v]'h -->` u_1 `[`v_1`];`
   - `[v -->` v_1`]'h;` (sólo si $u = u_1 = \emptyset$)
2. Las reglas de disolución del tipo $u[v]_h \rightarrow w$ pueden ser escritas de las siguientes maneras:
   - `u [v]'h --> w;`
   - `u [v]'h --> [w,@d]'h;`

- `u [v]'h --> [w,@d];`

3. Las reglas del tipo $[u \rightarrow v, w_{out}, w_{1(in_{h_1})} \ldots w_{n(in_{h_n})}]_h$ pueden ser escritas de las siguiente formas:

   - `[u []'h`$_1$ `...[]'h`$_n$`]'h --> w[v [w`$_1$`]'h`$_1$ `...[w`$_n$`]'h`$_n$`]'h;`
   - `[u []'h`$_1$ `...[]'h`$_n$`]'h --> w[v [w`$_1$`]'h`$_1$ `...[w`$_n$`]'h`$_n$`];`
   - `[u []'h`$_1$ `...[]'h`$_n$ `--> v [w`$_1$`]'h`$_1$ `...[w`$_n$`]'h`$_n$`]'h;` (sólo si $w = \emptyset$)

donde $u, v, w, u_1, v_1, w_1, \ldots, w_n$ son multiconjuntos de objetos; y $h, h_1, \ldots, h_n$ son etiquetas de membranas.

De manera opcional, se permite incluir el objeto especial `@d` en cualquier membrana de la parte derecha de cualquier regla con el fin de representar que la membrana que lo contiene será disuelta tras ejecutar la regla.

Los sistemas P de transición permiten prioridad entre sus reglas, el orden de prioridad se especifica en P-Lingua como una expresión numérica entre paréntesis a la izquierda de la regla, teniendo mayor prioridad aquellas que tienen menor número.

Algunos ejemplos:

- $[a^5\, c \rightarrow d_{out}\, e\, f_{in_1}\, g_{in_2}\, \lambda]_0 \equiv$
  `[a*5,c []'1 []'2]'0 --> d[@d,e,[f]'1 [g]'2]'0;`
- $[b^2 \rightarrow c\, d]_1 > [b \rightarrow x, y]_1 \equiv$
  `(1) [b*2 --> c,d]'1;`
  `(2) [b --> x,y]'1;`

`@model<symport_antiport_psystem>`

1. Se pueden definir reglas de comunicación simétrica del tipo $a[b]_h^{\alpha} \rightarrow b[a]_h^{\beta}$ de las siguientes maneras:

- α`a[b]'h --> `β`b[a]'h;`
- α`a[b]'h --> `β`b[a];`

donde a y b son objetos; h es una etiqueta; y α y β son identificadores de cargas eléctricas. Por ejemplo:

$a[b]_1^+ \rightarrow b[a]_1^0 \equiv$ `a +[b]'1 --> b [a];`

`@model<probabilistic_psystem>`

1. Se pueden escribir reglas del tipo $u[v]_h^\alpha \xrightarrow{p} u_1[v_1]_h^\beta$ de las siguientes maneras:
   - `u`α`[v]'h --> u`$_1$$\beta$`[v`$_1$`]'h::p;`
   - `u`α`[v]'h --> u`$_1$$\beta$`[v`$_1$`]::p;`
   - `u`α`[v]'h,e --> u`$_1$$\beta$`[v`$_1$`]'h,e::p;`
   - `u`α`[v]'h,e --> u`$_1$$\beta$`[v`$_1$`]::p;`
2. Se pueden escribir reglas de comunicación entre entornos del tipo $(x)_j \xrightarrow{p} (y)_k$ de la siguiente manera:

   ```
   [[x]'j []'k --> []'j [y]'k]'global::p;
   ```

donde x, y son objetos; u, v, u_1, v_1 son multiconjuntos de objetos; h es una etiqueta de membrana; j, k son etiquetas de membranas que corresponden a entornos; e es un identificador de entorno; *global* es la etiqueta correspondiente a la membrana piel en la estructura inicial de membranas; α y β son identificadores de cargas eléctricas; y p es un número real comprendido entre 0 y 1 que contiene la probabilidad de la regla asociada.

Algunos ejemplos:

- $a\,b^2[c, d_1]_2^+ \xrightarrow{0{,}8} x, y[z]_2^- \equiv$ `a,b*2 +[c,d]'1 --> x,y -[z]'2 :: 0.8;`

- $Y_{i,j}[]_2 \xrightarrow{k_{i,8}} [B^{k_i,12}]_2 \equiv$ `Y{i,j}[]'2 --> [B*k{i,12}]'2::k{i,8};`
- $(x)_i \xrightarrow{p_{i,j}} (y)_j : 1 \leq i \leq E, 1 \leq j \leq E \equiv$
  `[[x]'{i} []'{j} --> []'{i} [y]'{j}]'global::p{i,j}`
  `:1<=i<=E,1<=j<=E;`

`@model<stochastic_psystem>`

1. El formato para definir reglas de reescritura de multiconjuntos de objetos del tipo $u[v]_h \xrightarrow{c} u_1[v_1]_h$ se expresa como sigue:

   `u[v]'h --> u1[v1]'h::c`

2. El formato para definir reglas de reescritura de cadenas del tipo $[u + s]_h \xrightarrow{c} [v + r]_h$ se da a continuación:

   `[u,s]'h --> [v,r]'h::c`

donde α, β y γ son identificadores de cargas eléctricas, a, b, c son objetos del alfabeto, u, u_1, v, v_1 son multiconjuntos de objetos, s, r son listas de subcadenas separadas por comas, h es una etiqueta de membranas y p, c son expresiones numéricas que representan números reales. El resultado de la evaluación de p debe ser un número comprendido entre 0 y 1, el resultado de la evaluación de c debe ser un número mayor o igual que 0.

Por ejemplo:

$$[RNAP + < cap.\omega.op >]_m \xrightarrow{c} [< cap.\omega.RNAP.op >]_m \equiv$$
`[RNAP,<cap.?.op>]'m --> [<cap.?.RNAP.op>]'m::c`

`@model<tissue_psystems>`

1. El formato para definir reglas de comunicación del tipo $(h_1, u/v, h_2)$ se expresa como sigue:

`[u]'h`$_1$ `<-->` `[v]'h`$_2$

2. Se pueden escribir reglas de división del tipo $[a]_h \rightarrow [b]_h[c]_h$ de las siguientes maneras:

   - `[a]'h --> [b]'h [c]'h`
   - `[a]'h --> [b][c]`

donde h_1, h_2 son etiquetas de células o la etiqueta del entorno, h es una etiqueta de célula; u, v son multiconjuntos de objetos; y a, b, c son objetos.

Algunos ejemplos:

- $(1, b_2c/b_3^2, 0) \equiv$ `[b{2},c]'1 <--> [b{3}*2]'0`
- $[A_1]_2 \rightarrow [B_1]_2[C_1]_2 \equiv$ `[A{1}]'2 --> [B{1}] [C{1}]`

Sentencias paramétricas

En P-Lingua es posible definir sentencias paramétricas usando el siguiente formato:

```
sentence : range1, ..., rangeN, restr1, ..., restrN;
```

donde `sentence` es una sentencia del lenguaje o una secuencia de sentencias entre llaves, y `range1, ..., rangeN` es una lista de rangos separados por comas con el siguiente formato:

```
min_value <= iterator <= max_value
```

donde `min_value` y `max_value` son expresiones numéricas, números enteros o variables, e `iterator` es un iterador local al contexto de la sentencia. Se permite utilizar el operador $<$ en lugar de $<=$.

Y `restr1, ..., restrN` son restricciones opcionales para los valores de los índices, con la siguiente sintaxis:

```
value1 <>value2
```

donde `value1` y `value2` son expresiones numéricas, números enteros o variables.

La sentencia será repetida para cada posible valor de cada índice.

Algunos ejemplos de sentencias paramétricas:

1. $[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- : 1 \leq k \leq n \equiv$
   `[d{k}]'2 --> +[d{k}]-[d{k}] : 1<= k <= n;`

2. $[x_{i,j} \rightarrow x_{i,j-1}]_2^+ : 1 \leq i \leq m, 2 \leq j \leq n, i \neq j \equiv$
   `+[x{i,j} --> x{i,j-1}]'2 : 1<=i<=m,2<=j<=n,i<>j;`

Comentarios

Los programas en P-Lingua pueden incluir comentarios escribiendo frases entre las siguientes cadenas: `/*` y `*/`.

4.2. Definición en P-Lingua de un sistema P de transición

La Figura 4.2 ilustra un sistema P de transición que genera el conjunto $\{n^2 : n \geq 1\}$, en la Tabla 4.1 se muestra una posible computación.

Definición del sistema P en P-Lingua

Téngase en cuenta que en P-Lingua, el objeto b' se escribe `bp`, debido a que el carácter ′ (comilla simple) no es un carácter permitido para representar identificadores válidos.

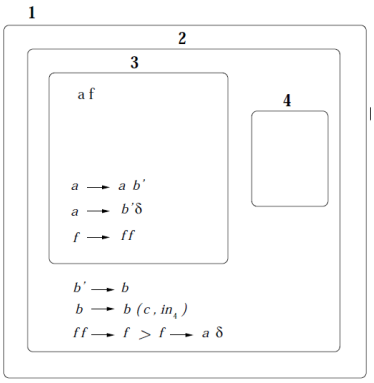


Figura 4.2: Un sistema P de transición que genera el conjunto $\{n^2 : n \geq 1\}$

```
@model<transition>
def main()
{
  call n_cuadrados();
}
def n_cuadrados()
{
  @mu = [[[]'3 []'4]'2]'1;
  @ms(3) = a,f;
  [a --> a,bp]'3;
  [a --> bp,@d]'3;
  [f --> f*2]'3;
  [bp --> b]'2;
  [b []'4 --> b [c]'4]'2;
  (1) [f*2 --> f ]'2;
  (2) [f --> a,@d]'2;
}
```

En este código se pueden observar las siguientes características sintácticas:

- Utilización de módulos y llamadas a módulos.
- Codificación de la estructura inicial de membranas, reglas y multiconjuntos iniciales de objetos.

Paso	Membrana 1	Membrana 2	Membrana 3	Membrana 4
0			af	
1			$ab'f^2$	
2			$ab'^2f^{2^2}$	
3			$ab'^3f^{2^3}$	
⋮	⋮	⋮	⋮	⋮
m			$ab'^mf^{2^m}$	
$m+1$		$b'^{(m+1)}f^{2^{m+1}}$	*disuelta*	
$m+2$		$b^{m+1}f^{2^m}$	*disuelta*	
$(m+2)+1$		$b^{m+1}f^{2^{m-1}}$	*disuelta*	e^{m+1}
$(m+2)+2$		$b^{m+1}f^{2^{m-2}}$	*disuelta*	$e^{2(m+1)}$
$(m+2)+3$		$b^{m+1}f^{2^{m-3}}$	*disuelta*	$e^{3(m+1)}$
⋮	⋮	⋮	⋮	⋮
$(m+2)+m$		$b^{m+1}f^{2^{m-m}}$	*disuelta*	$e^{m(m+1)}$
$2m+3$	ab^{m+1}	*disuelta*	*disuelta*	$e^{(m+1)(m+1)}$

Tabla 4.1: Una posible computación del sistema P

- Utilización del símbolo especial `@d` para indicar que una membrana va a ser disuelta.
- Utilización de prioridades entre las reglas.

4.3. Codificación de soluciones eficientes al problema SAT

En esta sección se muestran dos soluciones eficientes al problema **SAT** junto con sus respectivas codificaciones en P-Lingua. En ambas, se considera una fórmula proposicional genérica en forma normal conjuntiva $\varphi = C_1 \wedge \cdots \wedge C_m$ sobre n variables $x_1 \ldots x_n$ compuesta por m cláusulas $C_j = y_{j,1} \vee \cdots \vee y_{j,k_j}$, $1 \leq j \leq m$, donde $y_{j,i} \in \{x_l, \neg x_l \mid 1 \leq l \leq n\}$, $1 \leq i \leq k_j$. Se supone que no existen cláusulas con dos ocurrencias de algún x_i o $\neg x_i$ (la fórmula no es redundante a nivel de cláusulas), ni aparecen a la vez x_i y $\neg x_i$ en la misma cláusula (en este caso, la cláusula se satisface de manera trivial y puede ser eliminada).

Se codifica φ, una instancia del problema `SAT` con parámetros n y m, a

través del multiconjunto:

$$\begin{aligned} cod(\varphi) &= \{s_{i,j} \mid y_{j,r} = x_i,\ 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq r \leq k_j\} \\ &\cup \{s'_{i,j} \mid y_{j,r} = \neg x_i,\ 1 \leq i \leq n, 1 \leq j \leq m, 1 \leq r \leq k_j\}. \end{aligned}$$

En otras palabras, se reemplaza cada variable x_i de cada cláusula C_j por $s_{i,j}$ y cada variable negada $\neg x_i$ de cada cláusula C_j por $s'_{i,j}$, eliminando todos los paréntesis y conectivas. Así es posible pasar de φ a $cod(\varphi)$ en un número lineal de pasos con respecto a $n \cdot m$.

4.3.1. Una solución mediante sistemas P con membranas activas

A continuación se detalla una solución al problema **SAT** usando una familia de sistemas P reconocedores con membranas activas y reglas de división. Esta solución fue presentada originalmente por M.J. Pérez–Jiménez y otros en [94].

La instancia φ será procesada por el sistema P $\Pi(s(\varphi))$ con entrada $cod(\varphi)$, siendo $s(\varphi) = \frac{(n+m)\cdot(n+m+1)}{2} + n$ denotado por $\langle n, m\rangle$.

Para cada $(n, m) \in \mathcal{N}^2$, se considera el sistema P

$$(\Pi(\langle n, m\rangle), \Sigma(n, m), i(n, m))$$

donde

- $\Sigma(n, m) = \{s_{i,j}, s'_{i,j} : 1 \leq i \leq n,\ 1 \leq j \leq m\}$
- $i(n, m) = 2$
- $\Pi(\langle n, m\rangle) = (\Gamma(n, m), \{1, 2\}, [[\,]_2]_1, w_1, w_2, R)$, se define como sigue:
  - $\Gamma(n, m) = \Sigma(n, m) \cup \{c_k : 1 \leq k \leq n + 2\} \cup$
    $\{d_k : 1 \leq k \leq 3n + 2m + 3\} \cup$
    $\{r_{i,k} : 0 \leq i \leq n,\ 1 \leq k \leq n + 2\} \cup \{e, t\} \cup \{Yes, No\}$
  - $w_1 = \emptyset$

- $w_2 = \{d_1\}$
- El conjunto de reglas, R, es dado por:

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

  $\{[s_{i,1} \rightarrow r_{i,1}]_2^+, [s'_{i,1} \rightarrow r_{i,1}]_2^- \ : \ 1 \leq i \leq n\}$

  $\{[s_{i,1} \rightarrow \lambda]_2^-, [s'_{i,1} \rightarrow \lambda]_2^+ \ : \ 1 \leq i \leq n\}$

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

  $\{[s'_{i,j} \rightarrow s'_{i,j-1}]_2^+, [s'_{i,j} \rightarrow s'_{i,j-1}]_2^- \ : \ 1 \leq i \leq n, 2 \leq j \leq m\}$

  $\{[d_k]_2^+ \rightarrow [\,]_2^0 d_k, [d_k]_2^- \rightarrow [\,]_2^0 d_k \ : \ 1 \leq k \leq m\}$

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

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

  $\{[d_k \rightarrow d_{k+1}]_1^0 \ : \ m \leq k \leq 3n-3\}; [d_{3n-2} \rightarrow d_{3n-1}e]_1^0$

  $e[\,]_2^0 \rightarrow [c_1]_2^+; [d_{3n-1} \rightarrow d_{3n}]_1^0$

  $\{[d_k \rightarrow d_{k+1}]_1^0 \ : \ 3n \leq k \leq 3n+2m+2\}$

  $[r_{1,2n}]_2^+ \rightarrow [\,]_2^- r_{1,2n} \ ; \ \{[r_{i,2n} \rightarrow r_{i-1,2n}]_2^- \ : \ 1 \leq i \leq n\}$

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

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

  $[c_{n+1}]_2^+ \rightarrow [\,]_2^+ c_{n+1} \ ; \ [c_{n+1} \rightarrow c_{n+2}t]_1^0$

  $[t]_1^0 \rightarrow [\,]_1^+ t \ ; \ [c_{n+2}]_1^+ \rightarrow [\,]_1^- Yes \ ; \ [d_{3n+2m+3}]_1^0 \rightarrow [\,]_1^+ No$

Definición en P-Lingua

A continuación se va a mostrar el código del programa escrito en P-Lingua que especifica la familia de sistemas P antes descrita. En concreto, se va a considerar como ejemplo la siguiente fórmula para instanciar el problema `SAT` a resolver:

$$\varphi \equiv (x_1 + \neg x_2)(\neg x_2 + x_3 + \neg x_4)\, x_5\, \neg x_6$$

El módulo `main` puede ser modificado de manera sencilla para definir cualquier otro sistema P de la familia.

El código se estructura como sigue:

1. Módulo `main()`: define un sistema P reconocedor con membranas activas y reglas de división resolviendo el problema `SAT` para la fórmula descrita con 6 variables y 4 cláusulas. En primer lugar, este módulo llama al modulo `Sat(n,m)` para $(n, m) \equiv (6, 4)$. A continuación, el módulo incluye el multiconjunto inicial de la membrana de entrada con $cod(\varphi)$ donde los objetos $s_{i,j}$ son escritos `s{i,j}`, representando que la variable x_i se encuentra en la cláusula C_j, y los objetos $s'_{i,j}$ se codifican por `sp{i,j}`, representando que la variable $\neg x_i$ está en la cláusula C_j.

2. Módulo `Sat(n,m)`: define una familia de sistemas P reconocedores con membranas activas y reglas de división resolviendo el problema `SAT` para cualquier instancia con n variables y m cláusulas.

El código es el siguiente:

```
def Sat(n,m)
{
 @mu = [[]'2]'1;
 @ms(2) = d{1};
 [d{k}]'2 --> +[d{k}]-[d{k}] : 1 <= k <= m;
 {
  +[s{i,1} --> r{i,1}]'2;
  -[sp{i,1} --> r{i,1}]'2;
  -[s{i,1} --> #]'2;
  +[sp{i,1} --> #]'2;
 } : 1 <= i <= n;
 {
  +[s{i,j} --> s{i,j-1}]'2;
  -[s{i,j} --> s{i,j-1}]'2;
  +[sp{i,j} --> sp{i,j-1}]'2;
  -[sp{i,j} --> sp{i,j-1}]'2;
 } : 1<=i<=n, 2<=j<=m;
```

```
 {
  +[d{k}]'2 --> []d{k};
  -[d{k}]'2 --> []d{k};
 } : 1<=k<=m;
 d{k}[]'2 --> [d{k+1}] : 1<=k<=m-1;
 [r{i,k} --> r{i,k+1}]'2 : 1<=i<=n, 1<=k<=2*m-1;
 [d{k} --> d{k+1}]'1 : m <= k<= 3*m-3;
 [d{3*m-2} --> d{3*m-1},e]'1;
 e[]'2 --> +[c{1}];
 [d{3*m-1} --> d{3*m}]'1;
 [d{k} --> d{k+1}]'1 : 3*m <= k <= 3*m+2*n+2;
 +[r{1,2*m}]'2 --> -[]r{1,2*m};
 -[r{i,2*m} --> r{i-1,2*m}]'2 : 1<= i <= n;
 r{1,2*m}-[]'2 --> +[r{0,2*m}];
 -[c{k} --> c{k+1}]'2 : 1<=k<=n;
 +[c{n+1}]'2 --> +[]c{n+1};
 [c{n+1} --> c{n+2},t]'1;
 [t]'1 --> +[]t;
 +[c{n+2}]'1 --> -[]Yes;
 [d{3*m+2*n+3}]'1 --> +[]No;
} /* End of Sat module */
def main()
{
 call Sat(6,4);
 @ms(2) += s{1,1}, sp{2,1}, sp{2,2}, s{3,2},
           sp{4,2}, s{5,3}, sp{6,4};
} /* End of main module */
```

4.3.2. Una solución mediante sistemas P de tejido

A continuación se describe una solución al problema SAT presentada por M.A. Martínez-del-Amor y otros en [77] que utiliza una familia de sistemas P de tejido de grado $q \geq 1$ con reglas de comunicación symport/antiport y reglas de división de células.

La instancia φ será procesada por el sistema P de tejido $\Pi(s(\varphi))$ con entrada $cod(\varphi)$, siendo $s(\varphi) = \frac{(n+m)\cdot(n+m+1)}{2}+n$ denotado por $\langle n, m\rangle$. Entonces,

se construye el sistema P de tejido reconocedor de grado 2:

$$\Pi(\langle n, m\rangle) = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \mathcal{M}_2, R, 2),$$

con los siguientes componentes:

$$\begin{aligned}
\Gamma &= \Sigma \cup \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{r_i \mid 1 \leq i \leq m\} \\
&\quad \cup \{T_i, F_i \mid 1 \leq i \leq n\} \cup \{T_{i,j}, F_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m+1\} \\
&\quad \cup \{b_i \mid 1 \leq i \leq 3n+m+1\} \cup \{c_i \mid 1 \leq i \leq n+1\} \\
&\quad \cup \{d_i \mid 1 \leq i \leq 3n+nm+2m+1\} \\
&\quad \cup \{e_i \mid 1 \leq i \leq 3n+nm+2m+3\} \cup \ \{f, q, \texttt{yes}, \texttt{no}\}, \\
\Sigma &= \{s_{i,j}, s'_{ij} : \ 1 \leq i \leq n, \ 1 \leq j \leq m\}, \\
\Omega &= \Gamma - \{\texttt{yes}, \texttt{no}\}, \\
\mathcal{M}_1 &= \texttt{yes no}\ b_1\ c_1\ d_1\ e_1, \\
\mathcal{M}_2 &= f\ a_1\ a_2\ \ldots\ a_n,
\end{aligned}$$

El conjunto R está formado por las siguientes reglas:

- **Reglas de división:**
  $r_{1,i} \equiv [a_i]_2 \rightarrow [T_i]_2[F_i]_2$, para $1 \leq i \leq n$

- **Reglas de comunicación:**
  $r_{2,i} \equiv (1, b_i/b_{i+1}^2, 0)$, para $1 \leq i \leq n$
  $r_{3,i} \equiv (1, c_i/c_{i+1}^2, 0)$, para $1 \leq i \leq n$
  $r_{4,i} \equiv (1, d_i/d_{i+1}^2, 0)$, para $1 \leq i \leq n$
  $r_{5,i} \equiv (1, e_i/e_{i+1}, 0)$, para $1 \leq i \leq 3n+nm+2m$
  $r_6 \equiv (1, b_{n+1}c_{n+1}d_{n+1}/f, 2)$
  $r_{7,i} \equiv (2, c_{n+1}T_i/c_{n+1}T_{i,1}, 0)$, para $1 \leq i \leq n$
  $r_{8,i} \equiv (2, c_{n+1}F_i/c_{n+1}F_{i,1}, 0)$, para $1 \leq i \leq n$
  $r_{9,ij} \equiv (2, T_{i,j}/t_iT_{i,j+1}, 0)$, para $1 \leq i \leq n, 1 \leq j \leq m$
  $r_{10,ij} \equiv (2, F_{i,j}/f_iF_{i,j+1}, 0)$, para $1 \leq i \leq n, 1 \leq j \leq m$
  $r_{11,i} \equiv (2, b_i/b_{i+1}, 0)$, para $n+1 \leq i \leq 3n+m$
  $r_{12,i} \equiv (2, d_i/d_{i+1}, 0)$, para $n+1 \leq i \leq 3n+m$
  $r_{13,ij} \equiv (2, b_{3n+m+1}t_is_{i,j}/b_{3n+m+1}r_j, 0)$, para $1 \leq i \leq n, 1 \leq j \leq m$

$r_{14,ij} \equiv (2, b_{3n+m+1} f_i s'_{i,j} / b_{3n+m+1} r_j, 0)$, para $1 \leq i \leq n, 1 \leq j \leq m$
$r_{15,i} \equiv (2, d_i / d_{i+1}, 0)$, para $3n + m + 1 \leq i \leq 3n + nm + m$
$r_{16,i} \equiv (2, d_{3n+nm+m+i} r_i / d_{3n+nm+m+i+1}, 0)$, para $1 \leq i \leq m$
$r_{17} \equiv (2, d_{3n+nm+2m+1} / q\ \texttt{yes}, 1)$
$r_{18} \equiv (1, e_{3n+nm+2m+1} / e_{3n+nm+2m+2} q, 0)$
$r_{19} \equiv (1, e_{3n+nm+2m+2} / e_{3n+nm+2m+3}, 0)$
$r_{20} \equiv (2, \texttt{yes} / \lambda, 0)$
$r_{21} \equiv (1, e_{3n+nm+2m+3} q\ \texttt{no} / \lambda, 2)$
$r_{22} \equiv (2, \texttt{no} / \lambda, 0)$

Definición en P-Lingua

A continuación se describe un código en P-Lingua que define la familia de sistemas P de tejido presentada anteriormente. En concreto, se va a considerar como ejemplo la siguiente fórmula para instanciar el problema `SAT` a resolver:

$$\varphi = (x_1 + x_2 + \neg x_3 + x_4)(x_1 + x_2)(x_1 + \neg x_2 + \neg x_3)(x_2 + \neg x_3 + x_4)$$
$$(\neg x_1 + x_2 + x_4)(x_2 + \neg x_3 + x_4)(x_1 + x_4)(\neg x_1 + x_2 + x_3 + x_4)$$

El módulo `main` puede ser modificado de manera sencilla para definir cualquier otro sistema P de tejido perteneciente a la familia. El código fuente se estructura como sigue:

1. Módulo `main()`: define un sistema P reconocedor de tejido resolviendo el problema `SAT` para la fórmula antes descrita con 4 variables y 8 cláusulas. En primer lugar, se realiza una llamada al módulo `sat_tissue(n,m)` para $(n, m) \equiv (4, 8)$. Seguidamente. se introduce el multiconjunto inicial de la célula de entrada con $cod(\varphi)$ donde los objetos $s_{i,j}$ se escriben en P-Lingua como `s{i,j}` representando que la variable x_i está presente en la cláusula C_j, y los objetos $s'_{i,j}$ se escriben como `sp{i,j}` representando que la variable $\neg x_i$ se encuentra en la cláusula C_j.

2. Módulo `sat_tissue(n,m)`: define una familia de sistemas P de tejido

resolviendo el problema `SAT` para cualquier instancia con n variables y m cláusulas.

3. Módulo `init_cells()`: define las células iniciales del sistema P de tejido, asignando al mismo tiempo la etiqueta 0 al entorno

4. Módulo `init_rules(n,m)`: define las reglas de la familia.

5. Módulo `init_multisets(n)`: define los multiconjuntos iniciales de las células.

6. Módulo `init_environment(n,m)`: inicializa el multiconjunto del entorno.

El código es el siguiente:

```
@model<tissue_psystems>
def main()
{
 call sat_tissue(4,8);
 @ms(2) += s{1,1},s{2,1},sp{3,1},s{4,1},
           sp{1,2},s{2,2},
           s{1,3},sp{2,3},sp{3,3},
           s{2,4},sp{3,4},s{4,4},
           sp{1,5},s{2,5},s{4,5},
           s{2,6},sp{3,6},s{4,6},
           s{1,7},s{4,7},
           sp{1,8},s{2,8},s{3,8},s{4,8};
}
def sat_tissue(n,m)
{
  call init_cells();
  call init_multisets(n);
  call init_environment(n,m);
  call init_rules(n,m);
}
def init_cells()
{
```

```
  @mu = [[]'1 []'2]'0;
}
def init_rules(n,m)
{
  /* r1 */ [a{i}]'2 --> [T{i}] [F{i}] : 1<=i<=n;
  {
    /* r2 */ [b{i}]'1 <--> [b{i+1}*2]'0;
    /* r3 */ [c{i}]'1 <--> [c{i+1}*2]'0;
    /* r4 */ [d{i}]'1 <--> [d{i+1}*2]'0;
  } : 1<=i<=n;
  /* r5 */ [e{i}]'1 <--> [e{i+1}]'0   : 1<=i<=3*n+n*m+2*m;
  /* r6 */ [b{n+1},c{n+1},d{n+1}]'1 <--> [f]'2;
  {
    /* r7 */ [c{n+1},T{i}]'2 <--> [c{n+1},T{i,1}]'0;
    /* r8 */ [c{n+1},F{i}]'2 <--> [c{n+1},F{i,1}]'0;
  } : 1<=i<=n;
  {
    /* r9 */  [T{i,j}]'2 <--> [t{i},T{i,j+1}]'0;
    /* r10 */ [F{i,j}]'2 <--> [f{i},F{i,j+1}]'0;
  } : 1<=i<=n,1<=j<=m;
  {
    /* r11 */ [b{i}]'2 <--> [b{i+1}]'0;
    /* r12 */ [d{i}]'2 <--> [d{i+1}]'0;
  } : n+1<=i<=(n+1)+(2*n+m)-1;
  {
    /* r13 */ [b{3*n+m+1},t{i},s{i,j}]'2 <--> [b{3*n+m+1},r{j}]'0;
    /* r14 */ [b{3*n+m+1},f{i},sp{i,j}]'2 <--> [b{3*n+m+1},r{j}]'0;
  } : 1<=i<=n,1<=j<=m;
  /* r15 */ [d{i}]'2 <--> [d{i+1}]'0 : 3*n+m+1<=i<=3*n+n*m+m;
  /* r16 */ [d{3*n+n*m+m+i},r{i}]'2 <--> [d{3*n+n*m+m+i+1}]'0 : 1<=i<=m;
  /* r17 */ [d{3*n+n*m+2*m+1}]'2 <--> [yes,q]'1;
  /* r18 */ [e{3*n+n*m+2*m+1}]'1 <--> [e{3*n+n*m+2*m+2},q]'0;
  /* r19 */ [e{3*n+n*m+2*m+2}]'1 <--> [e{3*n+n*m+2*m+3}]'0;
  /* r20 */ [yes]'2 <--> [#]'0;
  /* r21 */ [e{3*n+n*m+2*m+3},no,q]'1 <--> [#]'2;
  /* r22 */ [no]'2 <--> [#]'0;
}
def init_multisets(n)
{
```

```
  @ms(1) = yes,no,b{1},c{1},d{1},e{1};
  @ms(2) = f;
  @ms(2) += a{i} : 1<=i<=n;
}
def init_environment(n,m)
{
  @ms(0) = f,q;
  @ms(0) += s{i,j},sp{i,j} : 1<=i<=n,1<=j<=m;
  @ms(0) += a{i},t{i},f{i} : 1<=i<=n;
  @ms(0) += r{i} : 1<=i<=m;
  @ms(0) += T{i},F{i} : 1<=i<=n;
  @ms(0) += T{i,j},F{i,j} : 1<=i<=n,1<=j<=m+1;
  @ms(0) += b{i} : 1<=i<=3*n+m+1;
  @ms(0) += c{i} : 1<=i<=n+1;
  @ms(0) += d{i} : 1<=i<=3*n+n*m+2*m+1;
  @ms(0) += e{i} : 1<=i<=3*n+n*m+2*m+3;
}
```

4.4. Una biblioteca de software para *Membrane Computing*

En esta sección se presenta el desarrollo de *pLinguaCore*, una biblioteca programada en Java [128] que implementa los elementos comunes de los simuladores de sistemas P (analizados en el capítulo anterior) facilitando la producción de aplicaciones informáticas para *Membrane Computing* y, de manera particular, facilitando la integración de P-Lingua en diversos entornos de software. Esta biblioteca se encuentra bajo licencia de software libre GNU GPL [125] y todas las aplicaciones desarrolladas en esta memoria se basan en ella.

La funcionalidad implementada se divide en tres módulos:

- Lectura y análisis de ficheros de texto que definen sistemas P: la biblioteca puede leer diversos tipos de formatos de fichero de texto que definen sistemas P, siendo P-Lingua uno de ellos. En cualquier caso, la

biblioteca detecta y localiza los errores léxico/sintácticos y semánticos en los ficheros.

- Simulación de computaciones de sistemas P: se han implementado diversos algoritmos de simulación para cada uno de los modelos soportados, siendo posible realizar simulaciones de computaciones paso a paso o hasta un estado de parada.
- Exportación de la definición de sistemas P: con el fin de posibilitar la compatibilidad con otras aplicaciones, la biblioteca puede exportar la definición de un sistema P (previamente leída de un fichero de entrada) a ficheros de salida que pueden ser interpretados por otras aplicaciones.

Esta biblioteca no es un producto cerrado, sino que puede ser ampliada mediante la inclusión de:

- Nuevos formatos de fichero para definir sistemas P.
- Nuevos modelos de sistemas P.
- Nuevos algoritmos de simulación para los modelos reconocidos.
- Nuevos formatos para la exportación de sistemas P.

La biblioteca se puede descargar de la página web `http://www.p-lingua.org`, en donde se puede encontrar su información técnica y detalles de programación, así como el formato de los ficheros utilizados, junto con documentación general sobre el proyecto P-Lingua.

4.4.1. Formatos de fichero para definir sistemas P

Los formatos de fichero utilizados en pLinguaCore para almacenar la definición de un sistema P se pueden clasificar en dos categorías: *Formatos de entrada* (cuyos ficheros pueden ser leídos y analizados por pLinguaCore)

y *Formatos de salida* (cuyos ficheros pueden ser generados por pLinguaCore). Algunos formatos pueden pertenecer a ambas categorías.

La definición de cada uno de estos formatos se puede encontrar detallada en la página web `http://www.p-lingua.org`.

Formatos de entrada

Los formatos de fichero para definir sistemas P que puede leer y procesar pLinguaCore son:

- El formato P-Lingua
- El formato P-XML

La biblioteca implementa un analizador léxico/sintáctico para cada uno de estos formatos. En el caso del formato P-Lingua, también detecta errores semánticos (desde el punto de vista de la programación) y los localiza en el fichero. De esta manera se garantiza que si la biblioteca tiene éxito en el proceso de lectura de un fichero P-Lingua, entonces este fichero define un sistema P sin errores ni incoherencias con el modelo utilizado.

El formato P-XML es un lenguaje basado en XML para definir sistemas P. La diferencia fundamental con P-Lingua es que en P-XML no se pueden definir módulos ni parámetros y, por tanto, no se pueden escribir familias de sistemas P.

Formatos de salida

Los formatos de fichero para definir sistemas P que pueden ser generados por la biblioteca son:

- El formato P-XML
- El formato P-Bin

El formato P-Bin es un formato binario que codifica sistemas P de la misma manera que P-XML, pero con la ventaja de ocupar menos espacio en disco al utilizar codificación binaria.

Ambos formatos están diseñados para servir de entrada a simuladores que no utilizan directamente P-Lingua como formato para definir sistemas P. De esta manera, es posible aprovechar la definición de un sistema P escrita en P-Lingua en cualquier simulador tras un proceso de traducción por parte de pLinguaCore, garantizando que los ficheros generados no contienen errores y son coherentes con el modelo de sistemas P seleccionado. Por ejemplo, los simuladores basados en GPUs presentados en [25, 26, 27] utilizan directamente el formato P-Bin, pero el usuario define el sistema P a simular mediante P-Lingua.

4.5. Simulación por software de sistemas P

En la biblioteca pLinguaCore implementa diferentes algoritmos de simulación secuenciales para los modelos soportados de sistemas P. Cada uno de estos algoritmos permite reproducir paso a paso una de las posibles computaciones del sistema P definido. En cada paso se almacena en memoria un objeto Java que codifica la configuración actual del sistema P. Según las necesidades del usuario, es posible volver a configuraciones previas para permitir una simulación interactiva.

Debido a las similitudes entre algunos modelos, es posible usar algunos algoritmos de simulación para diferentes modelos. Es importante hacer destacar que se supone que la definición del sistema P está libre de errores que puedan conducir a una computación incorrecta, debido a que el analizador de P-Lingua ha comprobado los posibles errores de programación.

4.5.1. Simulación de sistemas P de transición y sistemas P symport/antiport

El algoritmo de simulación aquí presentado sirve para reproducir computaciones de sistemas P de transición o de symport/antiport, indistintamente. El algoritmo genera una posible computación cada vez que es ejecutado, aplicando las reglas de manera maximal y no determinista. La posibilidad de utilizar cargas está contemplada, aunque los sistemas P de transición no las tenga, propiamente. A efectos del simulador, para un sistema P sin polarización se considera que todas las membranas de la configuración inicial tienen carga neutra 0, así mismo, el analizador de P-Lingua no permite ninguna regla que altere las cargas de las membranas.

La selección de reglas se realiza en dos iteraciones, en la primera se eligen reglas mediante el criterio de selección especificado en el simulador y, en la segunda, se aplican las reglas seleccionadas de manera maximal.

I. Inicialización

1. C_0 es la configuración inicial.
2. $R_{sel} = \{\}$ es un conjunto de tuplas que servirá para almacenar las reglas seleccionadas para ser ejecutadas en cada paso, junto con las membranas implicadas en cada regla y el número de veces que se ejecutará cada regla.
3. $C_t = C_0$ es la configuración actual.

II. Selección de reglas

Para cada `i`, $1 \leq i \leq 2$, hacer

1. Para cada membrana $m = [\,]_h^\alpha \in C_t$ hacer
   a) Para cada regla $r \in R$ aplicable a membranas con etiqueta h y carga α hacer
      1) Sea $u[v]_h^\alpha$ la parte izquierda de r

2) Sea N el mínimo número tal que u^N aparece en el padre de m y v^N aparece en m
3) Si `i`$= 1$, entonces
   - Generar N', un número entero aleatorio según una distribución uniforme en el intervalo $[0, N]$
   - Hacer $N = N'$
4) Eliminar N instancias de u en el padre de m y N instancias de v en m
5) Añadir $\langle r, m, N \rangle$ a R_{sel}

III. Ejecución de reglas

1. Para cada tupla $\langle r, m, N \rangle \in R_{sel}$ hacer
   a) Añadir N veces los multiconjuntos de la parte derecha de r
   b) Actualizar cargas
   c) Disolver membranas

IV. Finalización

1. Si $R_{sel} \neq \emptyset$ entonces
   - Hacer $C_{t+1} = C_t$
   - Hacer $R_{sel} = \{\}$
   - Ir a **II**
2. **Fin**

4.5.2. Simulación de sistemas P con membranas activas y reglas de división o creación

En este apartado se presenta un algoritmo de simulación que puede ser utilizado para sistemas P con membranas activas y reglas de división

o creación indistintamente. El algoritmo genera un posible computación, téngase en cuenta que cuando se trabaja con sistemas P reconocedores, todas las computaciones con la misma entrada producen la misma respuesta (confluencia).

Este algoritmo sólo considera sistemas P con división, disolución y creación de membranas elementales.

I. Inicialización

1. C_0 es la configuración inicial.
2. $R_{sel} = \{\}$ es un conjunto de tuplas que servirá para almacenar las reglas seleccionadas para ser ejecutadas en cada paso, junto con las membranas implicadas en cada regla y el número de veces que se ejecutará cada regla.
3. $C_t = C_0$ es la configuración actual.

II. Selección de reglas

1. Para cada membrana $m = [\,]_h^\alpha \in C_t$ hacer
   - *a*) Hacer `flag-solo-evolución`$=False$
   - *b*) Para cada regla $r \in R$ aplicable a membranas con etiqueta h y carga α hacer
     - Si r es de tipo $[a \rightarrow v]_h^\alpha$ y a aparece en m entonces
       1) Sea N la multiplicidad de a en m.
       2) Eliminar todas las instancias de a en m
       3) Añadir $\langle r, m, N\rangle$ a R_{sel}
     - Si $\neg$`flag-solo-evolución` entonces
       - Si r es de tipo $a[\,]_h^\alpha \rightarrow [b]_h^\beta$, m no es piel y a aparece en la membrana m_1 padre de m, entonces
         1) Eliminar una instancia de a en m_1
         2) Añadir $\langle r, m, 1\rangle$ a R_{sel}

3) Hacer `flag-solo-evolución`=$True$

- Si r es de tipo $[a]_h^\alpha \rightarrow b[\,]_h^\beta$ y a aparece en m, entonces
  1) Eliminar una instancia de a en m
  2) Añadir $\langle r, m, 1\rangle$ a R_{sel}
  3) Hacer `flag-solo-evolución`=$True$
- Si r es de tipo $[a]_h^\alpha \rightarrow [b]_h^\beta[c]_h^\gamma$ y m no es piel, entonces
  1) Eliminar una instancia de a en m
  2) Añadir $\langle r, m, 1\rangle$ a R_{sel}
  3) Hacer `flag-solo-evolución`=$True$
- Si r es de tipo $[a]_h^\alpha \rightarrow b$, m no es piel y a aparece en m, entonces
  1) Eliminar una instancia de a en m
  2) Añadir $\langle r, m, 1\rangle$ a R_{sel}
  3) Hacer `flag-solo-evolución`=$True$
- Si r es de tipo $[a]_h^\alpha \rightarrow [[b]_{h_1}^\beta]_h^\alpha$ y a aparece en m, entonces
  1) Eliminar una instancia de a en m
  2) Añadir $\langle r, m, 1\rangle$ a R_{sel}
  3) Hacer `flag-solo-evolución`=$True$

III. Ejecución de reglas

1. Para cada tupla $\langle [a \rightarrow v]_h^\alpha, m, N\rangle \in R_{sel}$ hacer
   a) Añadir N veces el multiconjunto v a m
2. Para cada tupla $\langle a[\,]_h^\alpha \rightarrow [b]_h^\beta, m, 1\rangle \in R_{sel}$ hacer
   a) Añadir el objeto b a m
   b) Cambiar la carga de m a β
3. Para cada tupla $\langle [a]_h^\alpha \rightarrow b[\,]_h^\beta, m, 1\rangle \in R_{sel}$ hacer
   a) Añadir el objeto b a m

b) Cambiar la carga de m a β

4. Para cada tupla $\langle [a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma, m, 1\rangle \in R_{sel}$ hacer

   a) Crear una nueva membrana m' con etiqueta h y carga β

   b) Copiar el multiconjunto de m en m'

   c) Cambiar la carga de m a γ

   d) Añadir el objeto b al multiconjunto de m'

   e) Añadir el objeto c al multiconjunto de m

   f) Asignar m' como hija del padre de m

5. Para cada tupla $\langle [a]_h^\alpha \rightarrow [[b]_{h_1}^\beta]_h^\alpha, m, 1\rangle \in R_{sel}$ hacer

   a) Crear una nueva membrana m' con etiqueta h_1 y carga β

   b) Añadir el objeto b al multiconjunto de m'

   c) Asignar m' como hija de m

6. Para cada tupla $\langle [a]_h^\alpha \rightarrow b, m, 1\rangle \in R_{sel}$ hacer

   a) Añadir el multiconjunto de m al multiconjunto del padre de m

   b) Añadir b al multiconjunto del padre de m

   c) Eliminar m

IV. Finalización

1. Si $R_{sel} \neq \emptyset$ entonces

   - Hacer $C_{t+1} = C_t$
   - Hacer $R_{sel} = \{\}$
   - Ir a **II**

2. **Fin**

4.5.3. Simulación de sistemas P probabilísticos

Dos diferentes algoritmos de simulación han sido desarrollados e integrados en la biblioteca pLinguaCore para el modelo de sistemas P probabilísticos. El primero se llama *Uniform Random Distribution Algorithm*. El segundo proporciona una mayor eficiencia gracias a la utilización de la distribución binomial, llamándose *Binomial Random Distribution Algorithm*

Cada uno de estos algoritmos sigue un esquema similar a los anteriores:

1. Inicialización
2. Selección de reglas
3. Ejecución de reglas
4. Finalización

Ambos algoritmos se diferencian únicamente en la estrategia utilizada para seleccionar las reglas aplicables a una configuración dada.

Téngase en cuenta que en este modelo no existen reglas de división y, por tanto, se establece una correspondencia biunívoca entre membranas y etiquetas.

Uniform Random Distribution Algorithm

El algoritmo siguiente determina el conjunto de reglas aplicables a una configuración C_t del sistema, en el instante t.

(a) Las reglas se clasifican en conjuntos de tal manera que todas las reglas que se encuentren en el mismo conjunto tienen la misma parte izquierda.

(b) Sea $\{r_1, \ldots, r_z\}$ uno de los conjuntos de reglas mencionados. Supongamos que la parte izquierda común es $u\ [v]_i^\alpha$ y sus respectivas constantes probabilísticas son $c_{r_1}, \ldots, c_{r_z}$ en el instante t. Para determinar cómo

son aplicadas estas reglas a una configuración dada, se procede como sigue:

- Se calcula el mayor número N tal que u^N aparece en la membrana padre de i y v^N aparece en la membrana i.
- Se generan N números aleatorios x, tal que $0 \leq x < 1$.
- Para cada k $(1 \leq k \leq z)$, sea n_k la cantidad de números aleatorios generados pertenecientes al intervalo $[\ \sum_{j=0}^{k-1} c_{r_j}\ ,\ \sum_{j=0}^{k} c_{r_j})$ (asumiendo que $c_{r_0} = 0$).
- Para cada k $(1 \leq k \leq z)$, la regla r_k se aplica n_k veces.

Binomial Random Distribution Algorithm

El algoritmo siguiente determina el conjunto de reglas aplicables a una configuración C_t del sistema en el instante t.

(a) Las reglas se clasifican en conjuntos de tal manera que todas las reglas que se encuentren en el mismo conjunto tienen la misma parte izquierda.

(b) Sea $\{r_1, \ldots, r_z\}$ uno de los conjuntos de reglas mencionados. Supongamos que la parte izquierda común es $u\ [v]_i^\alpha$ y sus respectivas constantes probabilísticas son $c_{r_1}, \ldots, c_{r_z}$ en el instante t. Para determinar cómo son aplicadas estas reglas a una configuración dada, se procede como sigue:

(c) Sea $F(N, p)$ una función que devuelve un número aleatorio discreto, según la distribución binomial $B(N, p)$

- Se calcula el mayor número N tal que u^N aparece en la membrana padre de i y v^N aparece en la membrana i.
- Sea $d = 1$

- Para cada k $(1 \leq k \leq z-1)$ hacer
  - sea $c_{r_k} = \frac{c_{r_k}}{d}$
  - sea $n_k = F(N, c_{r_k})$
  - sea $N = N - n_k$
  - sea $q = 1 - c_{r_k}$
  - sea $d = d * q$
- Sea $n_t = N$
- Para cada k $(1 \leq k \leq z)$, la regla r_k se aplica n_k veces.

4.5.4. Simulación de sistemas P de tejido con reglas de comunicación y división

El algoritmo de simulación aquí descrito genera una posible computación para sistemas P de tejido con reglas de comunicación y división. Recuérdese que cuando se trabaja con sistemas P reconocedores, todas las computaciones generan la misma respuesta (confluencia).

I. Inicialización

1. C_0 es la configuración inicial con n células $c_1, \ldots, c_n$.
2. c_0 es una célula virtual (con etiqueta 0) que representa el entorno, donde todos los objetos iniciales tienen multiplicidad infinita.
3. $R_{sel} = \{\}$ es un conjunto de tuplas que servirá para almacenar las reglas seleccionadas para ser ejecutadas en cada paso, junto con las membranas implicadas en cada regla y el número de veces que se ejecutará cada regla.
4. $C_t = C_0$ es la configuración actual.

II. Selección de reglas de comunicación

1. Para cada *regla de comunicación symport/antiport* $(i, u/v, j)$ hacer

   a) Para cada célula $c_{k_1} \in C_t$ con etiqueta i hacer

      1) Sea N el número más grande tal que el multiconjunto de c_{k_1} contiene N copias del multiconjunto u

      2) Para cada célula $c_{k_2} \in C_t$ con etiqueta j, mientras $N > 0$ hacer

         - Sea M el número más grande $M \leq N$ tal que el multiconjunto de c_{k_2} contiene M copias del multiconjunto v
         - Eliminar M copias de u del multiconjunto de c_{k_1}
         - Eliminar M copias de v del multiconjunto de c_{k_2}
         - Añadir $\langle c_{k_1}, c_{k_2}, (i, u/v, j), M\rangle$ a R_{sel}
         - Hacer $N = N - M$

III. Selección de reglas de división

1. Para cada *regla de división* $[a]_i \rightarrow [b]_i[c]_i$ hacer

   a) Para cada célula $c_k \in C_t$ con etiqueta i tal que c_k no aparece en R_{sel} hacer

      1) Si a se encuentra en el multiconjunto de c_k, entonces

         - Borrar 1 instancia de a del multiconjunto c_k
         - Añadir $\langle c_k, [a]_i \rightarrow [b]_i[c]_i\rangle$ a R_{sel}

IV. Ejecución de reglas

1. Para cada tupla $\langle c_{k_1}, c_{k_2}, (i, u/v, j), M\rangle$ de R_{sel} hacer

   a) Añadir M copias de v al multiconjunto de c_{k_1}

   b) Añadir M copias de u al multiconjunto de c_{k_2}

2. Para cada tupla $\langle c_k, [a]_i \rightarrow [b]_i[c]_i\rangle$ de R_{sel} hacer

   a) Crear una nueva célula c'_k con etiqueta i y el mismo multiconjunto que c_k

b) Añadir 1 instancia de b al multiconjunto de c_k

c) Añadir 1 instancia de c al multiconjunto de c'_k

V. Finalización

1. Si $R_{sel} \neq \emptyset$, entonces
   - Hacer $C_{t+1} = C_t$
   - Hacer $R_{sel} = \{\}$
   - Ir a **II**
2. **Fin**.

4.6. Herramientas para la línea de comandos

En el marco de P-Lingua, se han desarrollado dos herramientas para la línea de comandos: la primera es un compilador que permite traducir entre cualquiera de los formatos de entrada y cualquiera de los formatos de salida, detectando y localizando los posibles errores; la segunda es un simulador que recibe la definición de un sistema P, reconoce el modelo al cual pertenece y ejecuta la simulación paso a paso de una de las posibles computaciones, salvando la información de las configuraciones generadas en un fichero de texto.

Estas herramientas pueden ser descargadas bajo licencia GNU GPL de la página web `http://www.p-lingua.org/`, así como la explicación de su utilización.

4.6.1. Compilador para la línea de comandos

La herramienta de compilación para la línea de comandos es multiplataforma (es decir, que puede ser ejecutada sobre diferentes sistemas operativos) y permite la traducción entre cualquiera de los formatos de fichero de entrada

soportados por pLinguaCore a cualquiera de los formatos de salida. La herramienta puede serejecutada desde una consola del sistema operativo con la siguiente sintaxis:

```
plingua [-input_format] in_file [-output_format] out_file
[-v verbosity_level] [-h]
```

La cabecera `plingua` informa al programa que debe compilar un sistema P de un fichero origen a un fichero destino, donde `in_file` contiene la definición del sistema P que debe ser compilado, y `out_file` es el nombre del fichero que será generado (por defecto recibe el mismo nombre que el fichero de entrada, pero con distinta extensión). Los argumentos opcionales están representados entre corchetes:

- La opción `-input_format` indica el formato en el cual se encuentra el fichero de entrada, que debe ser uno de los formatos de entrada reconocidos por pLinguaCore:
  - `P-Lingua`
  - `P-XML`
- Si no se introduce ningún formato de entrada, se supone, por defecto, el formato P-Lingua por defecto.
- La opción `-output_format` indica el formato en el cual será generado el fichero de salida, que debe ser uno de los formatos de salida soportados por pLinguaCore:
  - `P-XML`
  - `P-Bin`
- Si no se indica ningún formato, se supone, por defecto, el formato P-XML por defecto.
- La opción `-v` indica el nivel de detalle de los mensajes mostrados durante el proceso de compilación (entre 0 y 5). El nivel por defecto es 3.

- La opción `-h` muestra información de ayuda.

4.6.2. Simulador para la línea de comandos

La herramienta de simulación para la línea de comandos es multiplataforma y permite reproducir computaciones de los sistemas P definidos, mediante la aplicación de los algoritmos de simulación presentados en esta memoria. El simulador puede ser ejecutado desde una consola del sistema operativo con la siguiente sintaxis:

```
plingua_sim [-input_format] input_file -o output_file
[-v verbosity level] [-h] [-to timeout] [-st steps] [-mode
simulatorID]
```

La cabecera `plingua_sim` informa que se debe simular una computación de un sistema P, donde `input_file` es un fichero de texto que codifica un sistema P según algún formato de entrada reconocido por pLinguaCore y `output_file` es el nombre de un fichero de texto donde se almacenará el informe sobre la simulación producida. Los argumentos opcionales se expresan entre corchetes:

- La opción `-input_format` define el formato seguido por `input_file`, que debe ser un formato de entrada reconocible por pLinguaCore. El formato por defecto es P-Lingua.
- La opción `-v verbosity level` indica el nivel de detalle de los mensajes mostrados durante el proceso de compilación (entre 0 y 5). El nivel por defecto es 3.
- La opción `-h` sirve para mostrar información de ayuda.
- La opción `-to timeout` establece un tiempo máximo (*timeout*) en milisegundos para la simulación. Cuando se alcanza el *timeout*, la simulación se detiene. Si la computación alcanza antes un estado de parada, el *timeout* no tiene efecto.

- La opción `-st steps` establece un número máximo de pasos de computación a simular. Cuando se alcanza ese número de pasos, la simulación se detiene. Si antes se alcanza un estado de parada (o se alcanza el *timeout*), entonces esta opción no tiene efecto.
- La opción `-mode simulatorID` especifica el algoritmo de simulación a utilizar.
  - `transition`
  - `active_membranes`
  - `binomial_probabilistic`
  - `uniform_probabilistic`
  - `tissue`

  Esta opción genera un error en el caso de que el algoritmo de simulación seleccionado no sea consistente con el modelo de sistema P definido en el fichero de entrada. Si no se indica ningún algoritmo, se establece uno por defecto para cada modelo de sistema P soportado.

La información generada sobre la computación simulada es la siguiente:

1. Estructura inicial del sistema P.
2. Multiconjuntos iniciales.
3. Conjunto de reglas.
4. Para cada configuración simulada:
   a) Multiconjunto de objetos en el entorno.
   b) Multiconjunto de objetos en cada compartimento del sistema.
   c) Reglas ejecutadas.

Parte III

Aplicación al estudio de ecosistemas

Capítulo 5

Modelos de ecosistemas basados en sistemas P

En la evolución de un ecosistema se encuentran involucrados un gran número de factores, a menudo interconectados entre ellos de manera dinámica. Esto hace que el proceso de modelizar un ecosistema sea una tarea demasiado laboriosa y, consecuentemente, sea necesario acotar el problema estableciendo valores fijos para un subconjunto de las variables implicadas en el diseño.

V. Volterra y A. Lotka propusieron el primer modelo de dinámica de poblaciones basado en sistemas de ecuaciones diferenciales: el modelo presa-depredador con limitaciones de recursos para la presa (Volterra, 1925) y el modelo de reacciones químicas en donde las concentraciones químicas oscilan (Lotka, 1926). A partir de ellos se obtiene un sistema de ecuaciones diferenciales que generaliza ambos modelos, y que es conocido con el nombre de *modelo de Lotka-Volterra*.

En contraste con las ecuaciones diferenciales, los sistemas P corresponden explícitamente al carácter discreto de las componentes de un ecosistema y usan reglas de reescritura de multiconjuntos de objetos que representan las variables del mismo. La estocasticidad inherente, el ruido externo y la incertidumbre de los ecosistemas se puede capturar mediante el uso de estrategias probabilísticas.

El uso de sistemas P para modelizar ecosistemas permite el estudio de la evolución simultánea de un alto número de especies. Además, la tarea de añadir nuevos ingredientes al modelo es relativamente simple, debido a la flexibilidad y modularidad de los sistemas P.

En la primera sección se describe un marco general basado en sistemas P de tipo probabilístico para la modelización de ecosistemas, así como un algoritmo de simulación que trata de capturar la semántica del modelo.

Teniendo presente que los sistemas P no han sido implementados en medios electrónicos ni biológicos y que están siendo usados para modelizar ecosistemas reales, se hace necesario el desarrollo de aplicaciones informáticas que permitan realizar una primera validación del modelo a través de la comparación de datos obtenidos experimentalmente con los que se obtienen en el modelo vía un simulador que permita ejecuciones en ordenadores electrónicos. En la sección 5.2 se plantea esta problemática.

El marco de modelización está acompañado por un entorno general de simulación por software basado en P-Lingua. En la sección 5.3 se presenta una herramienta diseñada como software para un simulador basado en P-Lingua, EcoSim 2.0. Dicha aplicación consiste en una serie de interfaces gráficas de usuario sobre la biblioteca *pLinguaCore*, así como un conjunto de ficheros de texto en formato P-Lingua que definen los modelos utilizados. Esta aplicación presenta dos modos de funcionamiento, por una parte se comporta como una *caja negra* para el usuario final ecólogo, proporcionando la funcionalidad de editar los parámetros iniciales del ecosistema y lanzar experimentos virtuales; por otra parte, la herramienta también sirve como asistente al diseño y validación experimental del modelo, permitiendo interactuar con el proceso de simulación computacional.

En la última sección de este capítulo se ilustra el contenido de las secciones anteriores mediante la descripción de un modelo de un ecosistema simple, utilizando el marco de modelización presentado así como las herramientas informáticas desarrolladas.

5.1. Sistemas P probabilísticos

En esta sección vamos a adaptar el marco de modelización basado en sistemas P presentado en el capítulo 2 para su uso en el estudio y análisis de la dinámica de poblaciones.

Recordemos que, según la Definición 2.1, un esqueleto de un sistema P extendido con membranas activas de grado $q \geq 1$, $\Pi = (\Gamma, \mu, R)$, consta de un conjunto de membranas etiquetadas por $0, \ldots, q-1$ jerarquizadas según una estructura μ. Todas las membranas tienen carga neutra y, además, cada una de ellas tiene asociado un conjunto finito de reglas de R.

Definición 5.1. *Un sistema P funcional–probabilístico con membranas activas de grado $q \geq 1$, usando T unidades de tiempo, $T \geq 1$, es una tupla*

$$\Pi = (\Gamma, \mu, R, T, \{f_r : \ r \in R\}, \mathcal{M}_0, \ldots, \mathcal{M}_{q-1})$$

en donde:

- (Γ, μ, R) *es el esqueleto de un sistema P extendido con membranas activas de grado q.*
- T *es un número natural, $T \geq 1$;*
- *Para cada $r \in R$, f_r es una función computable cuyo dominio es $\{1, \ldots, T\}$ y su rango está contenido en $[0, 1]$, de tal manera que verifica lo siguiente:*
  - ⋆ *Si $r_1, \ldots, r_z$ son las reglas de R que tienen la misma parte izquierda que r (por ejemplo, $u[v]_i^\alpha$), entonces $\sum_{j=1}^{z} f_{r_j}(t) = 1$, para $t = 1, \ldots, T$.*
- $\mathcal{M}_0, \ldots, \mathcal{M}_{q-1}$ *son multiconjuntos de objetos sobre Γ inicialmente colocados en las regiones de μ etiquetadas por $0, \ldots, q-1$, respectivamente.*

Un sistema P funcional–probabilístico con membranas activas de grado $q \geq 1$, usando T unidades de tiempo, $\Pi = (\Gamma, \mu, R, T, \{f_r : r \in R\}, \mathcal{M}_0, \ldots, \mathcal{M}_{q-1})$, se puede considerar como un conjunto de q membranas polarizadas, jerarquizadas por la estructura μ y etiquetadas inyectivamente por $0, \ldots, q-1$. El número natural $T \geq 1$ representa el tiempo de simulación del sistema. Para cada regla $r \in R$ y cada instante $t = 1, \ldots, T$, el número $f_r(t) \in [0, 1]$ representa una constante probabilística asociada a la regla r en el instante t, es la probabilidad que tiene esa regla de ser ejecutada, bajo el supuesto de que sea aplicable en ese instante. Notaremos de manera genérica $r : u[v]_i^\alpha \xrightarrow{f_r(t)} u'[v']_i^{\alpha'}$. Si $f_r(t) = 1$, entonces omitiremos la expresión $f_r(t)$ y escribiremos más brevemente $r : u[v]_i^\alpha \longrightarrow u'[v']_i^{\alpha'}$.

La tupla de multiconjuntos que están presentes en un instante dado en las q membranas del sistema, junto con las polarizaciones de éstas, constituye la *configuración* del sistema en ese instante. La tupla $(\mathcal{M}_0, \ldots, \mathcal{M}_{q-1})$, y todas las membranas con carga neutra, nos describen la configuración inicial del sistema Π.

El sistema P puede pasar de una configuración a otra mediante la aplicación de las reglas del conjunto R de acuerdo con el siguiente criterio:

- Una regla $u[v]_i^\alpha \to u'[v']_i^{\alpha'}$ es aplicable a una membrana etiquetada por i cuya carga eléctrica es α si el multiconjunto v está contenido en esa membrana y el multiconjunto u está contenido en su padre. Si una tal regla es aplicada, entonces los multiconjuntos u y v son eliminados de las citadas membranas y son reemplazados por los multiconjuntos u' y v', respectivamente. Además, la polarización pasa a ser α'.

- Las reglas del sistema son aplicadas en una forma que denominamos *paralela, maximal y consistente*; es decir, para cada $i \in \{0, \ldots, q-1\}$, $\alpha, \alpha' \in \{0, +, -\}$, todas las reglas del tipo $u[v]_i^\alpha \to u'[v']_i^{\alpha'}$, para u, v, u', v' multiconjuntos sobre Γ, que hayan sido seleccionados para su ejecución en un cierto instante, han de ser aplicadas simultáneamente en dicho paso.

- Para cada par de multiconjuntos u, v sobre Γ, cada $i \in \{0, \ldots, q-1\}$

y cada $\alpha \in \{0, +, -\}$, si $r_1, \ldots, r_z$ son las reglas aplicables cuya parte izquierda es $u[v]_i^\alpha$ en un cierto instante t, entonces esas reglas serán aplicadas de acuerdo con las correspondientes probabilidades $f_{r_1}(t), \ldots, f_{r_z}(t)$; es decir, si esas reglas compiten por k bloques de objetos (en la membrana i aparece v^k y en su padre aparece u^k), entonces esos bloques serán distribuidos de acuerdo con las distintas probabilidades asociadas a esas reglas en el instante considerado.

Definición 5.2. *Un sistema P multientorno funcional–probabilístico con membranas activas de grado (q, m), con $q \geq 1$, $m \geq 1$, usando T unidades de tiempo, $T \geq 1$, es un sistema P multientorno funcional con membranas activas de grado (q, m, m), usando T unidades de tiempo*

$$(G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,j} : r \in R_\Pi, 1 \leq j \leq m\}, \{\mathcal{M}_{ij} : 0 \leq i \leq q-1,\ 1 \leq j \leq m\})$$

en donde:

- *R_E es un conjunto finito de reglas de comunicación entre entornos, de la forma*

$$(x)_{e_j} \xrightarrow{p_{(x,j,j')}} (y)_{e_{j'}}$$

  en donde $x, y \in \Sigma$, $(e_j, e_{j'}) \in S$, y $p_{(x,j,j')}$ es una función computable cuyo dominio es $\{1, 2, \ldots, T\}$ y su rango está contenido en $[0, 1]$. Estas reglas satisfacen la siguiente propiedad: para cada entorno e_j, si $\{e_{j_1}, \ldots e_{j_z}\}$ es el conjunto de nodos en G alcanzables desde e_j, entonces $\sum_{i=1}^{z} p_{(x,j,j_i)}(t) = 1$, para cada $x \in \Sigma$ y $t = 1, \ldots, T$;

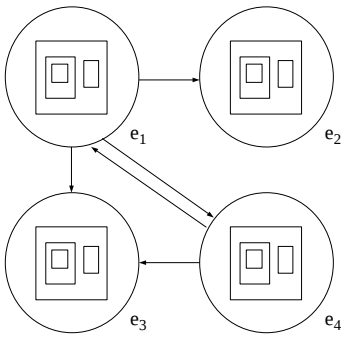
- *Para cada regla $r \in R_\Pi$ y para cada j, $1 \leq j \leq m$, $f_{r,j}$ es una función computable cuyo dominio es $\{1, \ldots, T\}$ y su rango está contenido en $[0, 1]$;*

- *Para cada $1 \leq j \leq m$, $(\Gamma, \mu, R_\Pi, T, \{f_{r,j} : r \in R_\Pi\}, \mathcal{M}_{0,j}, \ldots, \mathcal{M}_{q-1,j})$, que notaremos Π_j, es un sistema P funcional–probabilístico con membranas activas de grado $q \geq 1$ usando T unidades de tiempo. Notaremos por R_{Π_j} el conjunto R_Π en donde cada regla $r \in R_\Pi$ tiene asociada la función probabilística $f_{r,j}$.*

Un sistema P multientorno funcional–probabilístico con membranas activas de grado (q, m), con $q \geq 1,\ m \geq 1$, usando T unidades de tiempo, $T \geq 1$,

$$(\Gamma, \Sigma, G, R_E, \Pi, \{f_{r,j} : r \in R_\Pi, 1 \leq j \leq m\}, \{\mathcal{M}_{ij} : 0 \leq i \leq q-1,\ 1 \leq j \leq m\})$$

se puede considerar como un conjunto de m entornos $e_1, \ldots, e_m$ conectados a través de los arcos de un grafo dirigido G, como indica la siguiente figura.



Cada entorno e_j contiene un sistema P funcional–probabilístico con membranas activas de grado $q \geq 1$, usando T unidades de tiempo, Π_j. Todos los sistemas Π_j tienen el *mismo* esqueleto, Π, y $\mathcal{M}_{0,j}, \ldots, \mathcal{M}_{q-1,j}$ describen los correspondientes multiconjuntos iniciales de Π_j.

Al aplicar una regla de comunicación entre entornos $(x)_{e_j} \xrightarrow{p_{(x,j,j')}} (y)_{e_{j'}}$, el objeto x pasa del entorno e_j al entorno $e_{j'}$, posiblemente transformado en otro objeto y. En cualquier instante t, $1 \leq t \leq T$, en el que un objeto x está en el entorno e_j, la regla será aplicada de acuerdo a su probabilidad que viene dada por $p_{(x,j,j')}(t)$; es decir, si $\{e_{j1}, \ldots, e_{jz}\}$ es el conjunto de nodos alcanzables desde el nodo e_j y k es el número de copias del objeto x en el entorno e_j, entonces esos objetos son distribuidos entre los diferentes entornos $\{e_{j1}, \ldots, e_{jz}\}$ de acuerdo con las probabilidades $p_{(x,j,j_1)}(t), \ldots, p_{(x,j,j_z)}(t)$.

La tupla $(E_1, M_{0,1}, \ldots, M_{q-1,1}, \ldots, E_m, M_{0,m}, \ldots, M_{q-1,m})$ de multiconjuntos presentes en cada instante en los m entornos y en cada una de las regiones de los sistemas P colocados en esos entornos, junto con las polarizaciones de cada membrana en cada entorno, constituyen la *configura-*

ción del sistema en ese momento. La configuración inicial del sistema es $(\emptyset, \mathcal{M}_{0,1}, \ldots, \mathcal{M}_{q-1,1}, \ldots \emptyset, \mathcal{M}_{0,m}, \ldots, \mathcal{M}_{q-1,m})$, y todas las membranas con carga neutra. Es decir, admitiremos que, inicialmente, todos los entornos están vacíos.

El sistema puede pasar de una configuración a otra mediante la aplicación de las reglas del conjunto $R = R_E \cup \bigcup_{j=1}^{m} R_{\Pi_j}$ como sigue: en cada paso de computación, el número de veces que las reglas serán aplicadas se elige de acuerdo con las probabilidades asociadas a esas reglas en ese instante, y todas las reglas aplicables serán simultáneamente aplicadas de acuerdo con el criterio de maximal consistencia antes indicado.

Un algoritmo de simulación para sistemas P probabilísticos

A continuación, se presenta un algoritmo de simulación que permite describir la semántica de un modelo computacional basado en sistemas P probabilísticos.

(a) Generar una partición del conjunto de reglas de tal manera que en cada conjunto de la partición aparezcan todas aquellas reglas que tienen la misma parte izquierda.

(b) Sea $F(N,p)$ una función que devuelve un número aleatorio discreto a partir de la función de distribución binomial $B(N,p)$.

(c) Para cada paso de simulación hacer:

- Considerar un orden aleatorio sobre los conjuntos de la partición.
- Para cada conjunto de reglas $\{r_1, \ldots, r_t\}$ cuya parte izquierda es $u\,[v]_i^{\alpha}$ y sus respectivas constantes probabilísticas son $c_{r_1}, \ldots, c_{r_t}$, y de acuerdo con el orden considerado, hacer
  - Elegir aleatoriamente una regla del conjunto $\{r_1, \ldots, r_t\}$
  - Calcular el mayor número N tal que u^N aparece en la membrana padre de i y v^N aparece en la membrana i.
  - $d \longleftarrow 1$
  - Para cada k $(1 \leq k \leq t-1)$, de acuerdo con el orden seleccionado, hacer

$c_{r_k} \longleftarrow \frac{c_{r_k}}{d}$;

$n_{r_k} \longleftarrow F(N, c_{r_k})$;

$N \longleftarrow N - n_{r_k}$;

$q \longleftarrow 1 - c_{r_k}$;

$d \longleftarrow d * q$;

- $n_{r_t} \longleftarrow N$

- Cada regla r es aplicada n_r veces.

5.2. Validación experimental y experimentación virtual

La aleatoriedad inherente a la dinámica de los ecosistemas hace inviable la validación formal de modelos que tratan de reproducir su comportamiento. Por ello, es necesario realizar la validación de manera experimental, mediante la comparación de los resultados generados por herramientas de simulación con los datos experimentales obtenidos directamente del ecosistema real.

El protocolo de validación experimental es extensible a cualquier modelo computacional de procesos de la vida real e incluye la depuración incremental del modelo, tal como se muestra en la Figura 5.1. El protocolo es el siguiente:

1. Extracción de datos cuantitativos y cualitativos del proceso objeto de estudio; en el caso de los ecosistemas, será necesaria la implicación de expertos en Ecología que proporcionen esos datos mediante la observación directa, el trabajo de campo y la extrapolación de análisis estadísticos.

2. Diseño de un modelo basado en sistemas P probabilísticos a través de los datos proporcionados por los expertos y tras un proceso de interacción con ellos.

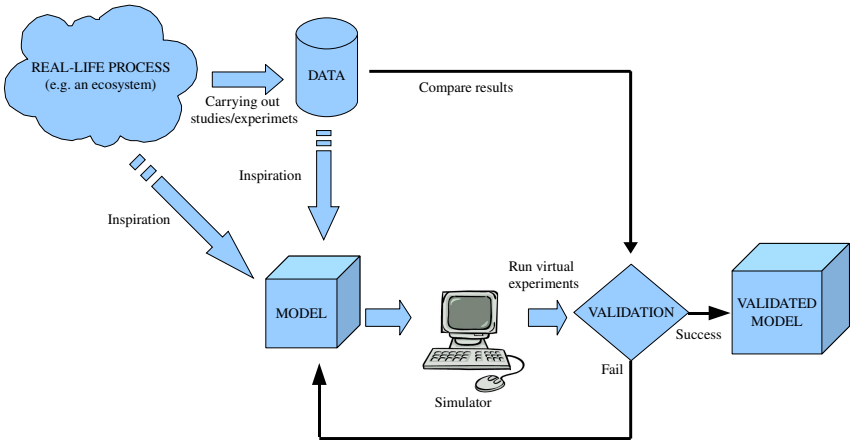


Figura 5.1: Protocolo de validación experimental

3. Desarrollo de una herramienta de simulación que permita reproducir el comportamiento del proceso durante un periodo de tiempo que ha sido estudiado previamente y del que se tiene datos experimentales. En el caso de los ecosistemas, estos datos pueden variar desde la evolución del número de animales de cada especie a lo largo de los años hasta las tasas de fertilidad o mortalidad en función de determinados parámetros.

4. Comparación de los resultados obtenidos por el simulador con los datos obtenidos experimentalmente. Si el margen de error es aceptable, entonces se puede considerar que el modelo ha sido validado experimentalmente. En caso contrario, se volverá al paso 2 en un proceso iterativo de depuración del modelo hasta conseguir un modelo validado.

Tal como se puede observar, el desarrollo de una herramienta que permita simular el comportamiento del modelo con la ayuda del ordenador es fundamental para el proceso de validación y depuración del modelo.

Por otra parte, una vez que se considera validado un modelo, es posible analizar la dinámica del mismo ante distintos escenarios que pudieran ser interesantes para los expertos, lo cual puede asistir a los expertos en el planteamiento de hipótesis plausibles.

El protocolo de experimentación virtual sería el siguiente:

1. Los expertos en el proceso modelizado sugieren el estudio de la evolución del sistema a partir de determinadas condiciones iniciales.

2. Se introducen en el simulador los parámetros que determinan el escenario hipotético de cada uno de los experimentos virtuales y se realizan las correspondientes simulaciones mediante la ejecución de la aplicación informática. Los resultados obtenidos pueden proporcionar hipótesis plausibles sobre la evolución esperada del modelo bajo las condiciones inicialmente fijadas.

3. Los expertos proceden a filtrar las hipótesis obtenidas, descartando aquellas que consideren menos plausibles.

4. En aquellos casos en los cuales sea posible, las hipótesis seleccionadas serán constatadas mediante un proceso de experimentación real, arrojando nuevo conocimiento.

En este caso, se puede comprobar que la herramienta de simulación es un valioso asistente para la formulación de hipótesis plausibles que pudieran ser de utilidad para el experto.

5.3. Software para la simulación

En esta memoria se han considerado tres casos de estudio, uno procedente de la literatura y dos ecosistemas reales. Para la validación y la experimentación virtual sobre los ecosistemas estudiados, se han desarrollado simuladores

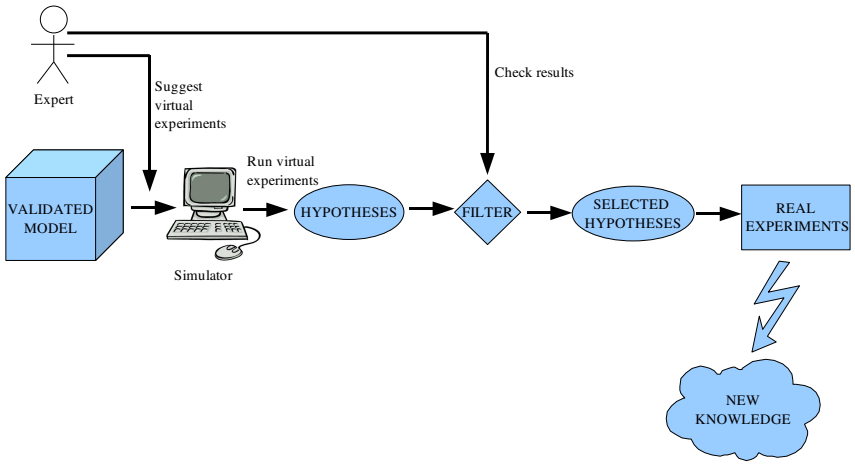


Figura 5.2: Protocolo de experimentación virtual

ad hoc, cada uno de los cuales corresponde a un ecosistema concreto. Todos ellos comparten el mismo motor de simulación (pLinguaCore) y se diferencian unos de otros en el modelo que define el ecosistema (escrito en un fichero P-Lingua) así como en la manera de introducir los datos de entrada e interpretar los datos de salida (la interfaz gráfica de usuario). Por ello, a partir de ahora se utilizará el término *familia de software EcoSim 2.0* para referirnos al conjunto de tres aplicaciones desarrolladas que están relacionadas entre sí y que comparten gran parte del código:

- EcoSim 2.0 Tritrophic
- EcoSim 2.0 Bearded Vulture
- EcoSim 2.0 Zebra Mussel

Se ha utilizado el lenguaje de programación P-Lingua para definir los sistemas P que modelizan cada uno de los ecosistemas; y la simulación de

los mismos se realiza a través del algoritmo de simulación de sistemas P probabilísticos descrito en el capítulo anterior.

Todas las aplicaciones de la familia EcoSim 2.0 proporcionan dos modos de funcionamiento, cada uno de ellos está dirigido a una categoría o tipo de usuario: *el ecólogo* y *el diseñador*. La herramienta permite diferentes tipos de acciones (casos de uso) para cada tipo de usuario.

Por una parte, el usuario ecólogo es el usuario final de la aplicación y, por tanto, no necesita tener ningún conocimiento acerca del paradigma *Membrane Computing*, ni siquiera acerca del modelo. Por ello, el programa se comporta como una especie de *caja negra*. El objetivo del usuario ecólogo es el desarrollo de experimentos virtuales sobre el ecosistema simulado y para este propósito el programa permite las siguientes acciones:

- Editar los parámetros iniciales del ecosistema.
- Seleccionar el número de años a simular.
- Seleccionar el número total de simulaciones a realizar.
- Guardar y cargar todos los parámetros introducidos en ficheros binarios con extensión *ec2*.
- Ejecutar simulaciones del ecosistema.
- Consultar los resultados de las simulaciones, los cuales se presentan mediante tablas y gráficas que se pueden configurar interactivamente.
- Guardar las gráficas y tablas como imágenes en formato *png*.

Con estas acciones, *el usuario ecólogo* puede analizar distintos escenarios del ecosistema y realizar experimentos virtuales, pudiendo grabar las distintas gráficas de resultados en ficheros. De este modo, es posible estudiar la evolución de la población de cada una de las especies durante los años simulados bajo unas condiciones iniciales determinadas; así como la variación de otros importantes valores, tales como la biomasa producida por especie y por año.

De manera transparente para el usuario ecólogo, el programa instancia en cada simulación un sistema P que codifica el comportamiento del ecosistema con los parámetros iniciales introducidos.

Por cada año simulado, se realizarán un determinado número de simulaciones fijado a priori, lo cual permite amortiguar el ruido inherente a los procesos estocásticos. Los resultados se expresarán a través de los valores medios y las desviaciones típicas de los valores estudiados.

El *usuario diseñador* es el responsable de especificar, depurar y validar la familia de sistemas P que usa el programa. La validación experimental se realiza mediante la comparación de los resultados obtenidos en las simulaciones con los valores reales observados y obtenidos experimentalmente en el ecosistema. Cada año de simulación corresponde a un determinado número de pasos de computación en el sistema P que se fija a priori por el usuario diseñador.

La especificación de la familia de sistemas P que modeliza el ecosistema objeto de estudio se describe por el usuario diseñador en un fichero en formato P-Lingua con extensión *pli*. Posteriormente, este usuario indica a la aplicación informática la ruta del fichero P-Lingua.

El programa ofrece al usuario diseñador las mismas acciones que al ecólogo, pero se comporta como una *caja blanca*, permitiendo interactuar con el proceso de simulación y facilitando así el proceso de depuración del modelo. Con este fin, se han incluido algunas acciones adicionales:

- Selección del fichero P-Lingua asociado a la aplicación.
- Compilación de ficheros P-Lingua.
- Simulación paso a paso.
- Selección del número de pasos de computación que corresponden a un año en el ecosistema modelizado.

El simulador puede trabajar en dos modos de funcionamiento (usuario ecólogo y usuario diseñador), y se establece el modo deseado editando un fichero de texto llamado *config.cnf*.

La aplicación es multiplataforma, en el sentido de que puede ejecutarse en diversos entornos y sistemas operativos, y ha sido desarrollada en el lenguaje de programación Java [128] junto con el entorno gráfico Swing [132], usando además las siguientes bibliotecas:

- pLinguaCore [134], que es responsable de procesar los ficheros P-Lingua y generar simulaciones computacionales.
- Colt [129], que es responsable de la generación de números aleatorios.
- JDom [130] para el procesamiento de ficheros XML.
- JFreeChart [131], que es responsable de la generación de gráficas obtenidas tras la realización de un proceso de simulación.

Todas las aplicaciones pertenecientes a la familia EcoSim 2.0 se encuentran bajo licencia de software libre GNU GPL [125] que permite el libre uso y modificación bajo unas condiciones determinadas. En el sitio web `http://www.p-lingua.org` se pueden descargar, junto con su código fuente, documentación técnica y manuales de usuario.

5.4. Un ejemplo: Interacciones tritróficas

En esta sección se analiza un ejemplo sencillo que nos va a permitir ilustrar el marco de modelización y las aplicaciones informáticas descritas en las secciones anteriores. Más concretamente, se estudia un ecosistema extraido de la literatura y se presenta un modelo basado en sistemas P probabilísticos, así como la especificación del modelo en P-Lingua y su simulación a través del programa *EcoSim 2.0 Tritrophic*.

5.4.1. Formulación del problema

Se trata de modelizar un sistema dinámico complejo, denominado *tritrófico*, que es, propiamente, una familia de ecosistemas ampliamente estudiados en

Ecología bajo el nombre de *interacciones tritróficas*.

Este tipo de sistema contiene tres categorías de especies que constituyen lo que se denomina como *cadena trófica*. En este ejemplo se contemplan de manera genérica tres tipos comunes de especies: plantas, animales herbívoros y animales carnívoros, siguiendo las ideas de Y. Suzuki y otros [113].

La dinámica del sistema se basa en el hecho de que los animales herbívoros se alimentan de plantas y, a su vez, los animales carnívoros se alimentan de herbívoros. Con el fin de mantener la sencillez del modelo, se supondrá que cuando un herbívoro o carnívoro consigue alimentarse, entonces se reproduce. Además, conviene tener presente que cuando un herbívoro se alimenta, la planta libera una sustancia química que atrae a los carnívoros. De igual manera, admitiremos que cada año se produce en el sistema la muerte de una determinada cantidad de individuos por causas naturales.

Por motivos de simplicidad, no se han contemplado las posibles diferencias de edad entre individuos de ninguna especie.

5.4.2. Diseño de un modelo basado en sistemas P

El modelo que se considera para estudiar el ecosistema antes descrito, es un sistema P multientorno funcional-probabilístico con membranas activas de grado $(2, 1)$; es decir, se trata de un modelo que consta de un único entorno el cual contiene un sistema P con dos membranas (en realidad, el comportamiento del modelo se puede reducir a un único sistema P que trabaja a modo de células).

$$(G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,1} : \ r \in R_\Pi\}, \{\mathcal{M}_{01}, \mathcal{M}_{11}\})$$

en donde:

- $G = (\{1\}, \{(1, 1)\})$
- El grafo del sistema es $\Gamma = \{X_{i,1} \ : \ 1 \leq i \leq 3\} \cup \{a_i \ : \ 1 \leq i \leq 4\} \cup \{b, c\}$ y $\Sigma = \emptyset$.

Es decir, Γ es el alfabeto de trabajo del sistema, en donde los símbolos $X_{1,1}, X_{2,1}$ y $X_{3,1}$ representan las plantas, los animales herbívoros y los animales carnívoros, respectivamente. El símbolo c se utiliza para codificar la cantidad de sustancias químicas que producen las plantas cuando los herbívoros se alimentan de ellas. Los símbolos a_i $(1 \leq i \leq 4)$ y b son elementos auxiliares que se introducen por razones técnicas.

- El conjunto R_E de reglas del entorno será vacío, con lo cual el entorno no juega ningún papel relevante en este modelo.

- El sistema $\Pi = (\Gamma, \mu, R)$ tiene como estructura de membranas $\mu = [[\,]_1]_0$; es decir, la estructura consta de la membrana piel y una membrana interna. Recordemos que la carga eléctrica neutra es omitida.

  - Los objetos que representan las plantas evolucionan de acuerdo a sus reglas de reproducción en la membrana piel.
  - Los objetos asociados a las distintas especies evolucionan de acuerdo a las reglas de alimentación y mortalidad en la membrana interna etiquetada por 1.

- $\mathcal{M}_{01}$ y $\mathcal{M}_{11}$ especifican los multiconjuntos iniciales de objetos presentes en cada membrana y representan las poblaciones iniciales que habitan en el sistema;

  - $\mathcal{M}_{01} = \{X_{1,1}^{q_{1,1}}\}$, en donde la multiplicidad $q_{1,1}$ indica la cantidad inicial de plantas.
  - $\mathcal{M}_{11} = \{X_{i,1}^{q_{i,1}} : 2 \leq i \leq 3\} \cup \{a_1\}$, en donde la multiplicidad $q_{i,1}$ indica el número de animales de la especie i (animales herbívoros y animales carnívoros).

- El conjunto R de reglas del sistema es el siguiente:

  - Reproducción de plantas.
    - $r_1 \equiv X_{1,1}[\,]_1 \xrightarrow{k_{1,1}} [X_{1,1}^2]_1$
    - $r_2 \equiv X_{1,1}[\,]_1 \xrightarrow{1-k_{1,1}} [X_{1,1}]_1$

- $r_3 \equiv [a_1]_1 \rightarrow [a_2]_1$

- Alimentación y reproducción de animales herbívoros.
  - $r_4 \equiv [X_{2,1}, X_{1,1}^{10}]_1 \xrightarrow{k_{2,1}} b[X_{2,1}^2, c]_1^+$
  - $r_5 \equiv [X_{2,1}, X_{1,1}^{10}]_1 \xrightarrow{1-k2,1} b[X_{2,1}, X_{1,1}^{10}]_1^+$
  - $r_6 \equiv [a_2]_1 \rightarrow b[a_3]_1^+$
- Alimentación y reproducción de animales carnívoros.
  - $r_7 \equiv [X_{3,1}, X_{2,1}, c]_1^+ \xrightarrow{k_{3,1}} b[X_{3,1}^2]_1^-$
  - $r_8 \equiv [X_{3,1}, X_{2,1}, c]_1^+ \xrightarrow{1-k_{3,1}} b[X_{3,1}, X_{2,1}, c]_1^-$
  - $r_9 \equiv [a_3]_1^+ \rightarrow b[a_4]_1^-$
- Mortalidad, reinicio del ciclo y eliminación de objetos.
  - $r_{10} \equiv [X_{1,1}]_1^- \xrightarrow{k_{1,2}} b[\,]_1$
  - $r_{11} \equiv [X_{1,1}]_1^- \xrightarrow{1-k_{1,2}} X_{1,1}[\,]_1$
  - $r_{12} \equiv [X_{2,1}]_1^- \xrightarrow{k_{2,2}} b[\,]_1$
  - $r_{13} \equiv [X_{2,1}]_1^- \xrightarrow{1-k_{2,2}} b[X_{2,1}]_1$
  - $r_{14} \equiv [X_{3,1}]_1^- \xrightarrow{k_{3,2}} b[\,]_1$
  - $r_{15} \equiv [X_{3,1}]_1^- \xrightarrow{1-k_{3,2}} b[X_{3,1}]_1$
  - $r_{16} \equiv [a_4]_1^- \rightarrow b[a_1]_1$
  - $r_{17} \equiv [b]_0 \rightarrow [\,]_0$

En donde los valores $k_{i,1}$ y $k_{i,2}$ $(1 \leq i \leq 3)$ son valores constantes que representan:

- $k_{i,1}$: la probabilidad de alimentación y reproducción de la especie i.
- $k_{i,2}$: la probabilidad de mortalidad de la especie i por causas naturales.

Obsérvese que en este ejemplo las funciones probabilísticas asociadas a las reglas son funciones constantes.

En la tabla 5.1 aparecen los valores asignados a las constantes probabilísticas asociadas a las distintas reglas y que han sido obtenidos por *ensayo y*

Especie	i	$k_{i,1}$	$k_{i,2}$	$q_{i,1}$
Plantas	1	0,6	0,05	10000
Herbívoros	2	0,4	0,15	1000
Carnívoros	3	0,1	0,08	100

Tabla 5.1: Valores utilizados

error, así como los valores de las multiplicidades $q_{i,1}$ $(1 \leq i \leq 3)$ que se han considerado en la simulación.

5.4.3. Un simulador basado en P-Lingua

El simulador desarrollado para este ejemplo concreto recibe el nombre de *EcoSim 2.0 Tritrophic* y está compuesto por la biblioteca *pLinguaCore* más una interfaz gráfica de usuario que permite al usuario ecólogo introducir los valores de los parámetros iniciales del ecosistema. Dichos parámetros son los siguientes:

- El número inicial de individuos de cada tipo.
- Las probabilidades de alimentación y reproducción.
- Las probabilidades de mortalidad.

La interfaz de usuario ha sido desarrollada con Java Swing y en la figura 5.3 se puede ver una imagen del proceso de introducción de parámetros.

Una vez introducidos los parámetros, el usuario puede seleccionar el número de años a simular y las simulaciones a realizar por año. Al lanzar un proceso de simulación, el programa instancia un sistema P concreto que modeliza el ecosistema particular objeto de estudio, para los parámetros introducidos inicialmente.

La salida de la aplicación presenta, a través de la interfaz gráfica, las tablas de población y las gráficas de valores medios y desviación típica para

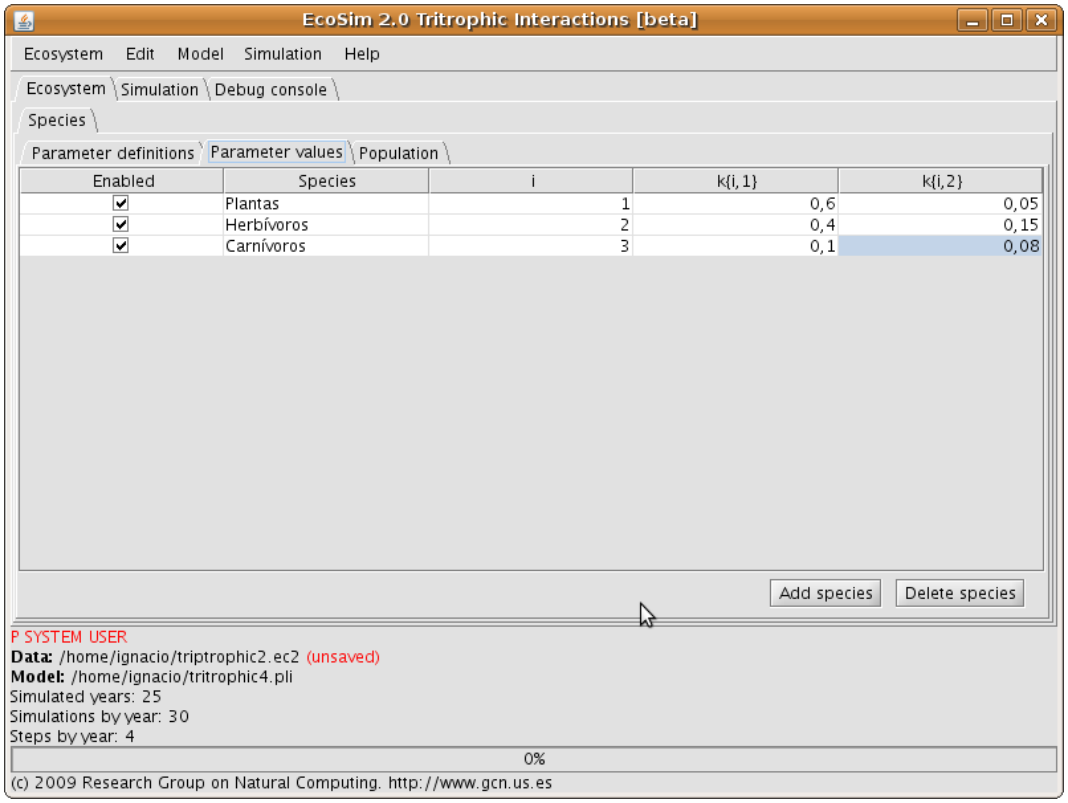


Figura 5.3: La interfaz gráfica de usuario de *EcoSim 2.0 Tritrophic*

cada especie, por cada año que se ha simulado. La figura 5.4 muestra los resultados obtenidos para un proceso de simulación del ecosistema durante 25 años, realizando 50 simulaciones por año.

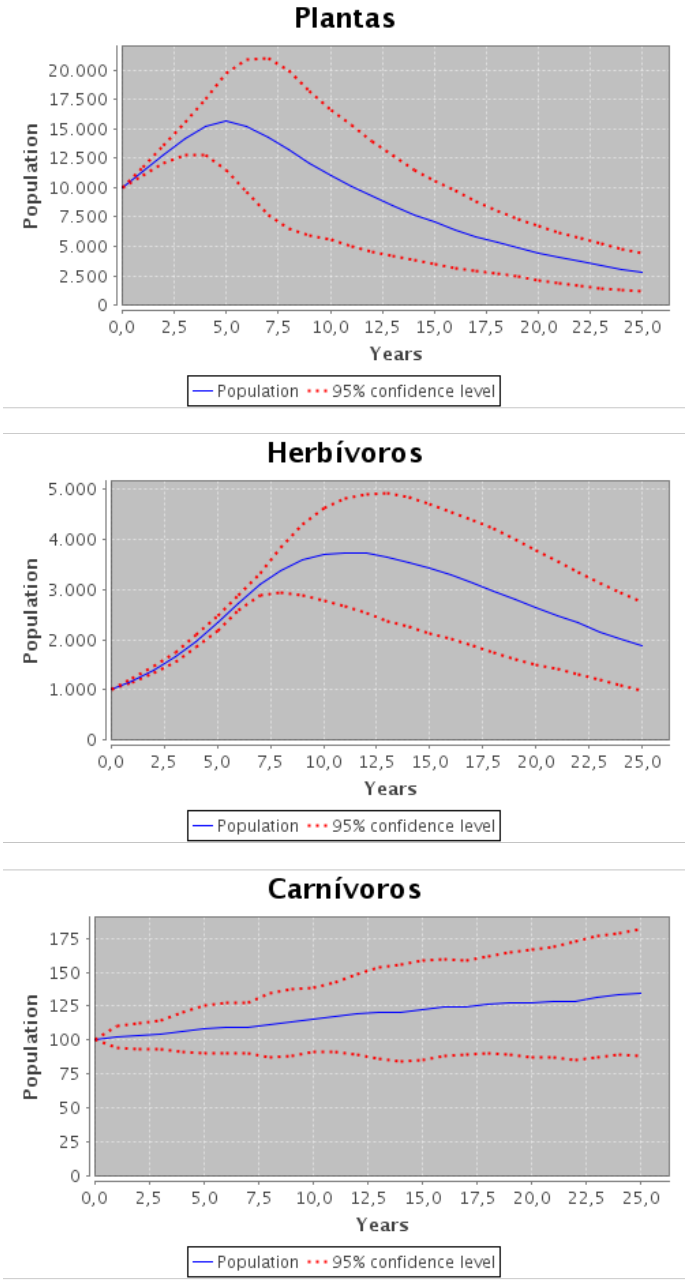


Figura 5.4: Resultados de la simulación con *EcoSim 2.0 Tritrophic*

El usuario diseñador debe escribir el sistema P en un fichero de texto en formato P-Lingua e indicar a la aplicación, mediante la interfaz de usuario, la ruta al fichero. También debe seleccionar el número de pasos de computación que corresponden a un ciclo. En nuestro ejemplo, de acuerdo con el modelo presentado, un ciclo se completa en cuatro pasos de computación.

La aplicación ofrece al usuario diseñador la posibilidad de depurar el modelo especificado en el fichero P-Lingua, mediante la simulación paso a paso, tal como se muestra en la figura 5.5, a través de una serie de botones de la interfaz gráfica de usuario que permiten:

(a) Establecer la ruta al fichero P-Lingua que especifica el modelo.

(b) Inicializar el modelo con los parámetros especificados en la interfaz y comprobar los posibles errores.

(c) Ejecutar simulaciones paso a paso, obteniendo la siguiente información para cada paso de computación simulado: el conjunto de reglas seleccionadas, el estado de las polarizaciones de las membranas y el contenido de los multiconjuntos en cada uno de los compartimentos.

Al igual que todo el software de la familia EcoSim 2.0, esta aplicación se encuentra bajo licencia GNU GPL [125] y se puede descargar de la página web `http://www.p-lingua.org` junto con su código fuente, la documentación técnica y el manual de usuario.

Definición del modelo en P-Lingua

A continuación se presenta el contenido del fichero en formato P-Lingua que codifica la familia de sistemas P que modeliza el ecosistema concreto objeto de estudio. Téngase en cuenta que existe una serie de variables de P-Lingua para las cuales no se especifica ningún valor de manera explícita en el fichero. Esto es debido a que la aplicación EcoSim 2.0 se encarga de asignar los valores adecuados a estas variables (obtenidos de la interfaz gráfica) antes de instanciar un sistema P de la familia. Las variables son las siguientes:

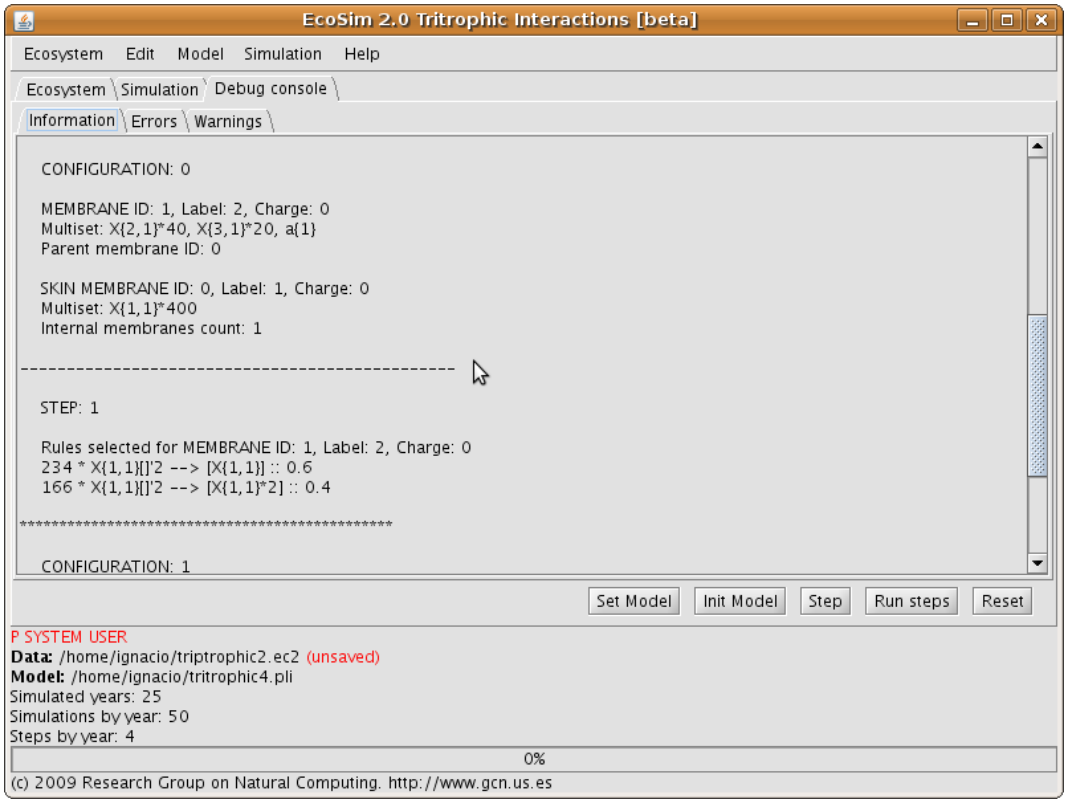


Figura 5.5: Depuración del modelo con *EcoSim 2.0 Tritrophic*

- `q{1,1}`,`q{2,1}` y `q{3,1}` que corresponden a las multiplicidades iniciales de las distintas especies que integran el sistema.
- `k{1,1}`,`k{2,1}`,`k{3,1}`,`k{1,2}`,`k{2,2}` y `k{3,2}` que corresponden a las constantes probabilísticas asociadas a las reglas.

El código es el siguiente:

```
@model<probabilistic>

def main()
{
 @mu = [ []'1 ]'0;
```

```
 @ms(0) = X{1,1}*q{1,1};
 @ms(1) = X{2,1}*q{2,1},X{3,1}*q{3,1},a{1};

 /* r1 */  X{1,1} []'1 --> [X{1,1}*2]'1  :: k{1,1};
 /* r2 */  X{1,1} []'1 --> [X{1,1}]'1 :: 1- k{1,1};
 /* r3 */  [a{1}]'1 --> [a{2}]'1 :: 1;

 /* r4 */ [X{2,1},X{1,1}*10]'1 --> b +[X{2,1}*2,c]'1 :: k{2,1};
 /* r5 */ [X{2,1},X{1,1}*10]'1 --> b +[X{2,1},X{1,1}*10]'1 :: 1-k{2,1};
 /* r6 */ [a{2}]'1 --> b +[a{3}]'1 :: 1;

 /* r7 */ +[X{3,1},X{2,1},c]'1 --> b -[X{3,1}*2]'1 :: k{3,1};
 /* r8 */ +[X{3,1},X{2,1},c]'1 --> b -[X{3,1},X{2,1},c]'1 :: 1-k{3,1};
 /* r9 */ +[a{3}]'1 --> b -[a{4}]'1 :: 1;

 /* r10 */ -[X{1,1}]'1 --> b []'1 :: k{1,2};
 /* r11 */ -[X{1,1}]'1 --> X{1,1} []'1 :: 1-k{1,2};
 /* r12 */ -[X{2,1}]'1 --> b []'1 :: k{2,2};
 /* r13 */ -[X{2,1}]'1 --> b [X{2,1}]'1 :: 1-k{2,2};
 /* r14 */ -[X{3,1}]'1 --> b []'1 :: k{3,2};
 /* r15 */ -[X{3,1}]'1 --> b [X{3,1}]'1 :: 1-k{3,2};
 /* r16 */ -[a{4}]'1 --> b [a{1}]'1 :: 1;
 /* r17 */  [b]'0 --> []'0 :: 1;
}
```

Capítulo 6

Casos de estudio

Este capítulo está dedicado a la presentación de unas aplicaciones prácticas del marco de modelización computacional de ecosistemas basado en sistemas P multientorno funcional–probabilísticos, así como de las herramientas de software desarrolladas para ese tipo de modelos.

En la sección 6.1 se estudia la problemática existente en un ecosistema real de la zona pirenaico–catalana en donde habita un número reducido de parejas de una ave carroñera en peligro de extinción: el *quebrantahuesos*. Se trata de analizar la dinámica de dicho ecosistema ante diferentes escenarios que sean de interés para los expertos, a fin de poder predecir su evolución y adoptar medidas encaminadas a un desarrollo sostenible de la especie citada en el ecosistema objeto de estudio. Para ello, se presenta un modelo basado en sistemas P del tipo antes mencionado y se detallan las características específicas de un software de simulación desarrollado al efecto.

En la sección 6.2 se estudia un ecosistema real enclavado en el pantano de Ribarroja gestionado por Endesa. Debido a la aparición en 2001 de una especie exótica invasora (el *mejillón cebra*), se están produciendo graves alteraciones en el ecosistema debido a la destrucción de especies autóctonas del pantano, así como importantes daños económicos en distintas canalizaciones y maquinarias que tiene la empresa citada en dicha zona. Para ello, se ha diseñado un modelo basado en sistemas P que trata de replicar la dinámica del ecosistema

y se detallan las características específicas de una aplicación informática desarrollada *ad hoc*, para la realización de simulaciones del modelo diseñado.

6.1. Un ecosistema real relacionado con el quebrantahuesos

El quebrantahuesos (*Gypaetus barbatus*) es un ave carroñera en peligro de extinción que se encuentra distribuida por las zonas montañosas de Eurasia y África, siendo una de las rapaces menos habituales en Europa (unas 150 parejas en 2007). Su mayor peculiaridad es que se alimenta casi exclusivamente de huesos de ungulados salvajes y domésticos [19].

En el Pirineo Catalán, al noreste de la Península Ibérica, el conjunto de los ciervos, corzos, rebecos, gamos y ovejas constituyen el 67 % de los recursos alimentarios de los carroñeros; el 33 % restante incluye pequeños mamíferos (por ejemplo, perros y gatos), grandes mamíferos (por ejemplo, vacas y caballos), mamíferos medianos (por ejemplo, jabalíes, cabras y muflones) y aves [73]. Una pareja de quebrantahuesos necesita para sobrevivir una media de 341 Kg de huesos por año [74, 75].

Durante el periodo de dispersión (desde que comienza la capacidad de vuelo hasta que las aves se vuelven territoriales a los 6 o 7 años de edad), los jóvenes quebrantahuesos cubren largas distancias viviendo en diferentes zonas pero sin asentarse en ningún territorio. Así por ejemplo, la superficie media cubierta por cuatro jóvenes quebrantahuesos monitorizados desde que comenzaron a volar fue de 4932 km^2 (en un rango que va desde 950 km^2 hasta 10294 km^2) [110]. Estas aves se vuelven territoriales cuando entran en la etapa de reproducción, el área aproximada de un territorio puede variar entre unos 250 km^2 y 650 km^2. El crecimiento anual de la población de quebrantahuesos en el Pirineo Catalán está estimado en un 4-5 %, la mayoría de la población permanece principalmente cerca de las estaciones de alimentación situadas en el Pirineo central (Aragón).

El quebrantahuesos no es la única especie de ave carroñera de los Pirineos, pues en este ecosistema existen también otras especies, como el buitre egipcio (*Neophron percnopterus*) y el buitre leonado euroasiático (*Gyps fulvus*), que compiten por el territorio y la alimentación, y que se alimentan mayoritariamente no de los huesos sino de la carne de animales muertos.

Las especies de ungulados presentan similitudes en su comportamiento natural, pues todos ellos son herbívoros que alcanzan su edad adulta al año de edad y, en general, llegan a la madurez sexual a los dos años. Dichas especies son también similares en cuanto a la tasa de mortalidad, que según se estima, es de un 50 % el primer año de vida y de un 6 % el resto de los años. Por otra parte, existen importantes diferencias entre dichas especies, algunas debidas a causas naturales y otras provocadas por la actividad humana. Por ejemplo, en la población de ciervos la mortalidad de los machos es mayor que la de las hembras debido a que los cazadores matan solamente animales machos para conseguir el trofeo de la cornamenta; por ello, se llevan sólo la cabeza dejando en el lugar el cuerpo, que puede ser aprovechado por los carroñeros. Otro ejemplo es el de la población de ovejas que, por ser animales domésticos, está estrechamente controlada por los propietarios de los rebaños, de modo que son éstos los que limitan el tamaño y el crecimiento de la población de estos bóvidos. Debido a un descenso de la fertilidad a la edad de ocho años, las ovejas suelen ser retiradas de los rebaños a esa edad. Por otra parte, la mayoría de los corderos son vendidos al mercado y retirados en el primer año de vida, sólo entre el el 20 % y el 30 % de los corderos, mayoritariamente hembras, se dejan en el campo para reemplazar las ovejas que han muerto de manera natural o han sido retiradas del rebaño [115].

Finalmente, hay que considerar dos factores importantes: la densidad máxima de población y las limitaciones de las fuentes de alimentación. En el ecosistema hay unos valores máximos de población para cada una de las especies, de tal manera que cuando la población supera esos umbrales, sobrevienen procesos naturales de autorregulación; por ejemplo, la aparición de epizootias u otras enfermedades diversas. Por otra parte, hay que tener en cuenta la cantidad de recursos naturales (hierba) disponible en el ecosistema

para la alimentación de las especies herbívoras.

El estudio de la evolución de la población y del comportamiento de las distintas especies en sus interacciones mutuas y con su entorno es un aspecto de sumo interés para la conservación y gestión del hábitat natural de los carroñeros en el Pirineo Catalán, especialmente del quebrantahuesos. En este sentido, tiene una gran importancia el diseño de un modelo formal de computación que reproduzca el comportamiento del ecosistema para su posible simulación por ordenador. Esto hace necesario el desarrollo de una herramienta informática que pueda asistir a los ecólogos, gestores y conservacionistas en la validación del modelo, la realización de experimentos virtuales y la selección de hipótesis plausibles que se deducen de dichos experimentos.

6.1.1. Diseño de un modelo basado en sistemas P

En [19] fue presentado el primer modelo de un ecosistema relacionado con el quebrantahuesos en el Pirineo Catalán, usando sistemas P. El modelo fue validado experimentalmente usando una aplicación informática desarrollada por los autores en el lenguaje de programación C++, utilizando datos experimentales obtenidos por los expertos durante los últimos 14 años. Para ese periodo de tiempo el ajuste del modelo respecto del ecosistema real, fue bastante aceptable, si bien el modelo generaba un crecimiento indefinido en los tamaños de población, debido a que no se había tenido presente algunos ingredientes relevantes del ecosistema en esa primera aproximación. Por ese motivo, el modelo inicial no era adecuado para realizar predicciones a largo plazo, si bien constituyó un punto de partida importante que constataba la idoneidad de los sistemas P como marco novedoso para la modelización de ecosistemas reales. No obstante, existían también otras limitaciones, concretamente de espacio y de alimento, que no habían sido contempladas.

En una segunda versión del modelo [20], se incorporaron algunos ingredientes nuevos con la finalidad de eliminar algunas de esas limitaciones. Cabe mencionar que en esta versión se utilizó por primera vez la herramienta de simulación EcoSim 2.0 y el lenguaje de programación P-Lingua para

la validación experimental del modelo. Conviene resaltar que el modelo presentado en [20] consta de un único entorno y, por tanto, puede ser considerado como un caso particular de un sistema P funcional–probabilístico con membranas activas que trabaja a modo de cálulas, en donde solamente se utilizaron dos polarizaciones: neutra y positiva. El ecosistema está compuesto por 13 especies de animales que proporcionan 18 tipos de animales en el modelo debido a la gestión de las distintas especies domésticas.

Mientras que en [19] el estudio del ecosistema está centrado en la modelización de una sóla especie de carroñero, el quebrantahuesos, en [20] se añadieron al modelo otras dos especies de carroñeros: el buitre egipcio y el buitre leonado que compiten entre sí y con el quebrantahuesos, tanto por el territorio como por el alimento. El quebrantahuesos se alimenta de huesos y las otras dos especies de carne de animales muertos. Una importante fuente de alimentación de estas especies consiste en los restos de animales domésticos y, por esa razón, se incluyeron cabras, vacas y caballos en el modelo presentado en [20].

El resultado final fue el diseño de un modelo computacional del ecosistema real objeto de estudio. Ese modelo ha sido validado experimentalmente usando la aplicación informática *EcoSim 2.0 Bearded Vulture*, contrastando los resultados que se deducen del simulador con los suministrados por los expertos, tal como se puede apreciar en la figura 6.1. En todos los casos, los datos reales se encuentran en un intervalo de confianza del 95 %, representado en la figura por líneas punteadas. Actualmente, el modelo y la aplicación informática antes citada constituyen herramientas útiles que están siendo utilizadas por los expertos en la toma de decisiones para la gestión del ecosistema real objeto de estudio.

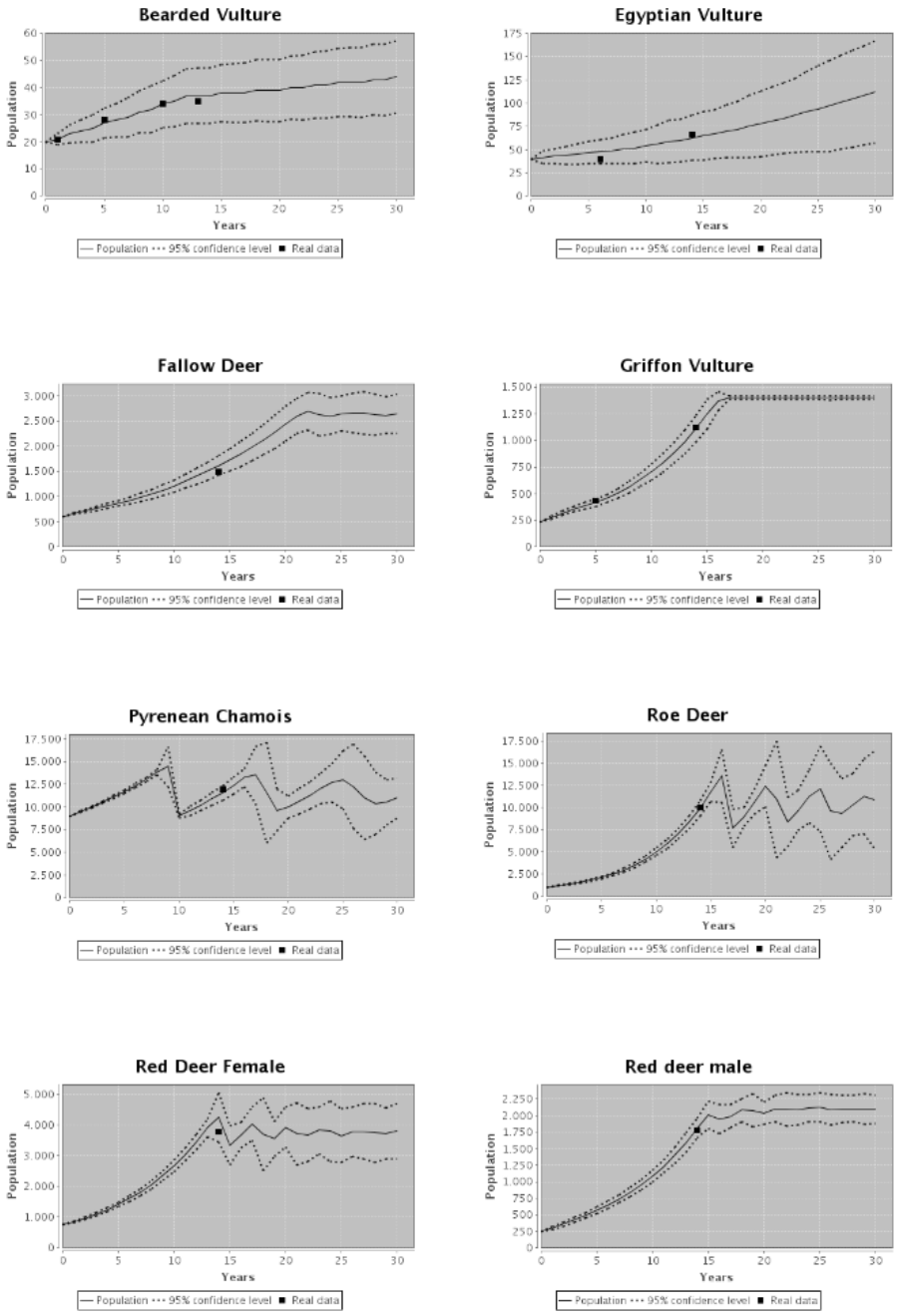


Figura 6.1: Resultados de la validación experimental

Descripción del modelo

Sea D un número natural mayor que 0, que representa el número de años que serán simulados en la evolución del ecosistema real antes descrito. En la definición de un sistema P probabilístico que modeliza el ecosistema objeto de estudio, $n = 18$ es el número de diferentes tipos de animales de las 13 especies que componen el ecosistema (3 carroñeros, 6 ungulados salvajes y 4 ungulados domésticos). Se consideran dos tipos de animales para el ciervo, debido al hecho de que los machos son más apreciados por los cazadores y eso implica que la tasa de mortalidad de los machos ($i = 6$) sea mayor que la de las hembras ($i = 5$). También se consideran dos tipos de animales, denominados por A (anuales) y P (periódicos) para los animales domésticos (excepto los caballos) porque algunos de ellos solamente pasan seis meses en la montaña.

A continuación, se presenta una lista de las constantes que se asociarán a las reglas del modelo o a los parámetros que intervienen en ella, y se detallan los correspondientes significados (el índice i, $1 \leq i \leq n$, representa el tipo de animal).

- $g_{i,1}$: es una constante que vale 1 para animales salvajes y 0 para animales domésticos.

- $g_{i,2}$: indica el intervalo de tiempo que los animales pasan en la montaña durante el año.

- $g_{i,3}$: representa la edad a la que se alcanza el tamaño adulto; es decir, la edad a partir de la cual el animal se alimenta como un adulto y, en el caso de morir, la edad a partir de la cual el animal dejaría una cantidad de biomasa similar a la que corresponde a un adulto. Por otra parte, a partir de esta edad los animales han superado la primera fase crítica en la cual la probabilidad de morir es alta.

- $g_{i,4}$: indica la edad a la que comienzan a ser fértiles.

- $g_{i,5}$: representa la edad a la que terminan de ser fértiles.

- $g_{i,6}$: representa la esperanza media de vida en el ecosistema.
- $g_{i,7}$: indica la densidad máxima en el ecosistema.
- $g_{i,8}$: número de animales que sobreviven después de alcanzar la densidad máxima en el ecosistema.
- $k_{i,1}$: indica la proporción de hembras en la población.
- $k_{i,2}$: representa la tasa de fertilidad (proporción de hembras fértiles que se pueden reproducir).
- $k_{i,3}$: indica el número de descendientes por cada hembra fértil que se reproduce.
- $k_{i,4}$: es un parámetro cuyo valor es 0 en el caso en que la especie pasa por un crecimiento natural, y su valor es 1 cuando el animal es nómada (el quebrantahuesos se desplaza por diferentes lugares hasta que cumple 6 o 7 años de edad y, entonces, se establece en un territorio concreto).
- $m_{i,1}$: indica la proporción de mortalidad por causas naturales en los primeros años de vida; es decir, cuando $edad < g_{i,3}$.
- $m_{i,2}$: representa la proporción de mortalidad en los animales adultos; es decir, cuando $edad \geq g_{i,3}$.
- $m_{i,3}$: indica el porcentaje de animales domésticos pertenecientes a poblaciones no estables que son retiradas en los primeros años de edad.
- $m_{i,4}$: es un parámetro cuyo valor es 1 si el animal muere a la edad de $g_{i,6}$ y no es retirado del ecosistema, o es igual a 0 si el animal no muere a la edad de $g_{i,6}$ pero es retirado.
- $f_{i,1}$: indica la cantidad de huesos de animales jóvenes cuando mueren; es decir, cuando $edad < g_{i,3}$.

- $f_{i,2}$: indica la cantidad de carne de animales jóvenes cuando mueren; es decir, cuando $edad < g_{i,3}$.

- $f_{i,3}$: indica la cantidad de huesos de animales adultos cuando mueren; es decir, cuando $edad \geq g_{i,3}$.

- $f_{i,4}$: indica la cantidad de carne de animales adultos cuando mueren; es decir, cuando $edad \geq g_{i,3}$.

- $f_{i,5}$: indica la cantidad de huesos necesarios por año y animal (1 unidad es igual a 0.5 kg de huesos).

- $f_{i,6}$: indica la cantidad de hierba necesaria por año y animal.

- $f_{i,7}$: indica la cantidad de carne necesaria por año y animal.

Los valores de estas constantes han sido obtenidos experimentalmente (ver [18], [38], [76], [73] para detalles) excepto $k_{i,4}$ (que es un factor característico de la especie) y $m_{i,4}$ (que es un factor antrópico). Las constantes k, m y f están asociadas a las reglas de reproducción, mortalidad y alimentación, respectivamente, como se verá a continuación. Las constantes g están asociadas a las restantes.

El ecosistema queda modelizado por el siguiente sistema P multientorno funcional–probabilístico con membranas activas de grado (2,1) y dos cargas eléctricas (neutra y positiva).

$$\Pi_D = (G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,1} : \ r \in R_\Pi\}, \{\mathcal{M}_{11}, \mathcal{M}_{21}\})$$

En donde:

- El grafo del sistema es $G = (\{1\}, \{(1,1)\})$

- $\Gamma = \{X_{ij},\ Y_{ij},\ V_{ij},\ Z_{ij} : \ 1 \leq i \leq n,\ 0 \leq j \leq g_{i,6}\} \cup$
  $\{B,\ G,\ M,\ B',\ G',\ M',\ C,\ C'\} \cup \{h_s : \ 1 \leq s\} \cup$
  $\{H_i,\ H_i',\ F_i,\ F_i',\ T_i,\ a_i,\ b_{0i},\ b_i,\ d_i,\ e_i : \ 1 \leq i \leq n\}$

$\Sigma = \emptyset$

Es decir, Γ es el alfabeto de trabajo del sistema, en donde los símbolos X, Y y Z representan el mismo animal en diferentes estados. El índice i se encuentra asociado con el tipo de animal y el índice j con su edad, siendo $g_{i,6}$ la esperanza de vida. Los símbolos auxiliares B, B' representan huesos, M, M' representan carne y G, G' representan la cantidad de hierba disponible para la alimentación de las especies en el ecosistema. Los objetos H_i, H'_i representan la biomasa de huesos y los objetos F_i, F'_i representan la biomasa de carne dejada por las especies i en diferentes estados.

El objeto C habilita la creación de objetos B', M' y G' que codifican huesos y carne (introducidos en el ecosistema de manera artificial) así como la cantidad de hierba generada por el propio ecosistema. Por otra parte, el objeto C produce objetos C' que a su vez generan objetos C propiciando el reinicio del ciclo. En el diseño del sistema P, se han considerado diferentes objetos (p.ej. G, G') que representan la misma entidad (en este caso, hierba) con el propósito de sincronizar el modelo. T_i es un objeto que se usa para contar los animales existentes de la especie i. Si una especie supera la densidad máxima, los valores se autorregulan.

Los objetos $b_{0,i}$, b_i y e_i permiten controlar el número máximo de animales por especie en el ecosistema. Cuando se provoca la regulación de la densidad de población, el objeto a_i produce la eliminación del número de animales de la especie i que excede de la densidad máxima. El objeto d_i se utiliza para controlar los animales domésticos que son elimunados del ecosistema para su comercialización.

- El sistema $\Pi = (\Gamma, \mu, R)$ tiene como estructura de membranas $\mu = [\,[\,]_2\,]_1$; es decir, la estructura de membranas consta de la membrana la piel y una membrana interna. La primera región es importante para controlar que las densidades de cada especie no superen el umbral del ecosistema. Los animales se reproducen, se alimentan y mueren en la membrana interna. Por simplicidad, la polarización neutra será omitida.

- $\mathcal{M}_{11}$ y $\mathcal{M}_{21}$ especifican los multiconjuntos de objetos sobre Γ inicialmente localizados en las regiones de μ y que codifican las poblaciones iniciales, así como el alimento que existe inicialmente en el ecosistema;

  - $\mathcal{M}_{11} = \{b_{0i},\ X_{ij}^{q_{ij}},\ h_t^{q_{1j}} : 1 \leq i \leq n,\ 0 \leq j \leq g_{i,6}\}$, en donde q_{ij} indica el número de animales de la especie i y edad j, inicialmente presentes en el ecosistema y

    $$t = \text{máx}\{1,\ \lceil \frac{\sum_{j=8}^{21} q_{1j} - 6}{1{,}352} \rceil\}$$

    El valor de t que aparece en la expresión matemática anterior se obtiene realizando una regresión lineal (que nos proporciona el crecimiento de la población).
  - $\mathcal{M}_{21} = \{C\}$

- El conjunto R está formado por las siguientes reglas:

  - La primera regla representa la contribución de recursos energéticos al ecosistema al principio de cada ciclo y es esencial para la evolución del sistema.

    $r_0 \equiv [C \rightarrow B'^{\alpha} M'^{\beta} G'^{\gamma} C']_2^0$, en donde α y β son los kilos de huesos y carne introducidos externamente en el ecosistema, y γ es la cantidad de hierba producida por el ecosistema.

    La segunda regla es utilizada para sincronizar el proceso.

    $r_1 \equiv [b_{0,i} \rightarrow b_i]_1^0$.

  - *Reglas de variación de la población.*

    Se consideran dos casos debido a que en las especies nómadas, la variación de la población está influenciada por los animales de ecosistemas externos.

    - Caso 1. Especies que no son nómadas ($k_{i,4} = 0$).

◇ Machos adultos:

$$r_2 \equiv [X_{ij} \xrightarrow{(1-k_{i,1})\cdot(1-k_{i,4})} Y_{ij}]_1^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,4} \leq j < g_{i,5} \end{cases}$$

◇ Hembras adultas que se pueden reproducir:

$$r_3 \equiv [X_{ij} \xrightarrow{k_{i,2}\cdot k_{i,1}\cdot(1-k_{i,4})} Y_{ij}Y_{i0}^{k_{i,3}}]_1^0, \begin{cases} 1 \leq i \leq 4, \\ g_{i,4} \leq j < g_{i,5} \end{cases}$$

$$r_4 \equiv [X_{ij} \xrightarrow{k_{i,2}\cdot k_{i,1}\cdot(1-k_{i,4})} Y_{ij}Y_{i0}^{k_{i,3}}]_1^0, \begin{cases} 7 \leq i \leq n, \\ g_{i,4} \leq j < g_{i,5} \end{cases}$$

$$r_5 \equiv [X_{5j} \xrightarrow{0,5\cdot k_{5,2}} Y_{5j}Y_{50}^{k_{i,3}}]_1^0, \ g_{5,4} \leq j < g_{5,5}.$$

$$r_6 \equiv [X_{5j} \xrightarrow{0,5\cdot k_{5,2}} Y_{5j}Y_{60}^{k_{i,3}}]_1^0, \ g_{5,4} \leq j < g_{5,5}.$$

◇ Hembras adultas que no se reproducen:

$$r_7 \equiv [X_{ij} \xrightarrow{(1-k_{i,2})\cdot k_{i,1}\cdot(1-k_{i,4})} Y_{ij}]_1^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,4} \leq j < g_{i,5} \end{cases}$$

◇ Hembras y machos viejos que no se reproducen:

$$r_8 \equiv [X_{ij} \xrightarrow{1-k_{i,4}} Y_{ij}]_1^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,5} \leq j \leq g_{i,6} \end{cases}$$

◇ Animales jóvenes que no se reproducen:

$$r_9 \equiv [X_{ij} \xrightarrow{1-k_{i,4}} Y_{ij}]_1^0, \begin{cases} 1 \leq i \leq n, \\ 1 \leq j < g_{i,4} \end{cases}$$

○ Caso 2. Especies nómadas ($k_{i,4} = 1$).

$$r_{10} \equiv [X_{1j}h_s \xrightarrow{v_s} Y_{1(g_{i,4}-1)}Y_{1j}h_{s+1}^2]_1^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,4} \leq j \leq g_{i,6}, \\ t \leq s \leq D_1 \end{cases}$$

siendo $v_s = 1{,}352/(1{,}352s + 6)$ y D_1=min$\{21, \ D + t - 1\}$.

$$r_{11} \equiv [X_{1j}h_s \xrightarrow{0,01} Y_{1(g_{i,4}-1)}Y_{1j}h_{s+1}^2]_1^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,4} \leq j \leq g_{i,6}, \\ D_3 \leq s \leq D_2 \end{cases}$$

donde D_2=max$\{21,\ D+t-1\}$ y D_3=max$\{21,\ t\}$.

$$r_{12} \equiv [X_{1j}h_s \xrightarrow{1-v_s} Y_{1j}h_{s+1}]_1^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,4} \leq j \leq g_{i,6}, \\ t \leq s \leq D_1 \end{cases}$$

$$r_{13} \equiv [X_{1j}h_s \xrightarrow{0,99} Y_{1j}h_{s+1}]_1^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,4} \leq j \leq g_{i,6}, \\ D_3 \leq s \leq D_2 \end{cases}$$

- *Reglas de mortalidad.*
  - Animales jóvenes que sobreviven:

    $$r_{14} \equiv Y_{ij}[\ ]_2^0 \xrightarrow{1-m_{i,1}-m_{i,3}} [V_{ij}T_i]_2^+, \begin{cases} 1 \leq i \leq n, \\ 0 \leq j < g_{i,3} \end{cases}$$

  - Animales jóvenes que mueren:

    $$r_{15} \equiv Y_{ij}[\ ]_2^0 \xrightarrow{m_{i,1}} [H_i'^{f_{i,1}\cdot g_{i,2}} F_i'^{f_{i,2}\cdot g_{i,2}} B'^{f_{i,1}\cdot g_{i,2}} M'^{f_{i,2}\cdot g_{i,2}}]_2^+, \begin{cases} 1 \leq i \leq n, \\ 0 \leq j < g_{i,3} \end{cases}$$

  - Animales jóvenes que son retirados del ecosistema:

    $$r_{16} \equiv [Y_{ij} \xrightarrow{m_{i,3}} \lambda]_1^0, \begin{cases} 1 \leq i \leq n, \\ 0 \leq j < g_{i,3} \end{cases}$$

  - Animales adultos que no alcanzan la esperanza media de vida y sobreviven:

    $$r_{17} \equiv Y_{ij}h_s^{k_{i,4}}[\ ]_2^0 \xrightarrow{1-m_{i,2}} [V_{ij}T_i h_s^{k_{i,4}}]_2^+, \begin{cases} 1 \leq i \leq n, \\ g_{i,3} \leq j < g_{i,6}, \\ t+1 \leq s \leq D+t \end{cases}$$

  - Animales adultos que no alcanzan la esperanza media de vida y mueren:

    $r_{18} \equiv Y_{ij}h_s^{k_{i,4}}[\,]_2^0 \xrightarrow{m_{i,2}} [H_i'^{f_{i,3}\cdot g_{i,2}} F_i'^{f_{i,4}\cdot g_{i,2}} B'^{f_{i,3}\cdot g_{i,2}} M'^{f_{i,4}\cdot g_{i,2}} V_{i,g_{i,4}-1}^{k_{i,4}} h_s^{k_{i,4}} T_i^{k_{i,4}}]_2^+,$
    en donde $1 \leq i \leq n,\ g_{i,3} \leq j < g_{i,6},\ t+1 \leq s \leq D+t.$

  - Animales que alcanzan la esperanza media de vida y mueren:

$r_{19} \equiv Y_{ig_{i,6}} h_s^{k_{i,4}} [\,]_2^0 \xrightarrow{c_{19}} [H_i'^{f_{i,3}\cdot g_{i,2}} F_i'^{f_{i,4}\cdot g_{i,2}} B'^{f_{i,3}\cdot g_{i,2}} M'^{f_{i,4}\cdot g_{i,2}} V_{i,g_{i,4}-1}^{k_{i,4}} h_s^{k_{i,4}} T_i^{k_{i,4}}]_2^+,$, en donde $1 \leq i \leq n$, $t+1 \leq s \leq D+t$ y siendo $c_{19} = k_{i,4} + (1-k_{i,4}) \cdot (m_{i,4} + (1-m_{i,4}) \cdot m_{i,2})$.

- Animales que alcanzan la esperanza media de vida y son retirados del ecosistema:

$$r_{20} \equiv [Y_{ig_{i,6}} h_s^{k_{i,4}} \xrightarrow{(1-k_{i,4})\cdot(1-m_{i,4})\cdot(1-m_{i,2})} \lambda]_1, \begin{cases} 1 \leq i \leq n, \\ t+1 \leq s \leq D+t \end{cases}$$

- *Reglas de regulación de la densidad.*

  - Creación de objetos que posibilitan el control del número máximo de animales en el ecosistema:

$$r_{21} \equiv b_i[\,]_2^0 \rightarrow [b_i a_i^{\lceil 0{,}9 * g_{i,7} \rceil} e_i^{\lceil 0{,}2 * g_{i,7} \rceil}]_2^+, \ 1 \leq i \leq n.$$

  - Evaluación de la densidad de las distintas especies en el ecosistema:

$$r_{22} \equiv [T_i^{g_{i,7}} a_i^{(g_{i,7}-g_{i,8})} \rightarrow \lambda]_2^+, \ 1 \leq i \leq n.$$

  - Generación de aleatoriedad en el número de animales:

$$r_{23} \equiv [e_i \xrightarrow{0,5} a_i]_2^+, \ 1 \leq i \leq n.$$

$$r_{24} \equiv [e_i \xrightarrow{0,5} \lambda]_2^+, \ 1 \leq i \leq n.$$

  - Cambio de los nombres de objetos que representan animales:

$$r_{25} \equiv [V_{ij} \rightarrow Z_{ij}]_2^+, \begin{cases} 1 \leq i \leq n, \\ 0 \leq j < g_{i,6} \end{cases}$$

  - Cambio de los nombres de objetos que representan recursos alimentarios:

$$r_{26} \equiv [G' \rightarrow G]_2^+.$$

$$r_{27} \equiv [B' \rightarrow B]_2^+.$$

$$r_{28} \equiv [M' \rightarrow M]_2^+.$$

$$r_{29} \equiv [C' \to C]_2^+.$$

$$r_{30} \equiv [H_i' \to H_i]_2^+,\ 1 \leq i \leq n.$$

$$r_{31} \equiv [F_i' \to F_i]_2^+,\ 1 \leq i \leq n.$$

- *Reglas de alimentación.*

$$r_{32} \equiv [Z_{ij} h_s^{k_{i,4}} a_i B^{f_{i,5} \cdot g_{i,2}} G^{f_{i,6} \cdot g_{i,2}} M^{f_{i,7} \cdot g_{i,2}}]_2^+ \to X_{i(j+1)} h_s^{k_{1,4}} [\,]_2^0, \begin{cases} 1 \leq i \leq n, \\ 0 \leq j \leq g_{i,6}, \\ t+1 \leq s \leq D+t \end{cases}$$

- *Reglas de actualización.*

  El objetivo de las siguientes reglas consiste en realizar una actualización al final del año. En particular, el alimento sobrante no es útil para el siguiente año y, por tanto, hay que eliminarlo. Si la cantidad de comida no es suficiente, entonces algunos animales tienen que morir.

  - Eliminación de los huesos, carne y hierba sobrantes:

$$r_{33} \equiv [G \to \lambda]_2^0.$$

$$r_{34} \equiv [M \to \lambda]_2^0.$$

$$r_{35} \equiv [B \to \lambda]_2^0.$$

$$r_{36} \equiv [T_i \to \lambda]_2^0,\ 1 \leq i \leq n.$$

$$r_{37} \equiv [a_i \to \lambda]_2^0,\ 1 \leq i \leq n.$$

$$r_{38} \equiv [e_i \to \lambda]_2^0,\ 1 \leq i \leq n.$$

$$r_{39} \equiv [b_i]_2^0 \to b_i[\,]_2^0,\ 1 \leq i \leq n.$$

$$r_{40} \equiv [H_i]_2^0 \to H_i[\,]_2^0,\ 1 \leq i \leq n.$$

$$r_{41} \equiv [F_i]_2^0 \to F_i[\,]_2^0,\ 1 \leq i \leq n.$$

  - Animales jóvenes que mueren por falta de comida:

$$r_{42} \equiv [Z_{ij} \xrightarrow{g_{i,1}} H_i'^{f_{i,1}} F_i'^{f_{i,2}} B'^{f_{i,1}} M'^{f_{i,2}}]_2^0, \begin{cases} 1 \leq i \leq n, \\ 0 \leq j < g_{i,3} \end{cases}$$

$$r_{43} \equiv [Z_{ij}]_2^0 \xrightarrow{1-g_{i,1}} d_i[\ ]_2^0, \begin{cases} 1 \leq i \leq n, \\ 0 \leq j < g_{i,3} \end{cases}$$

- Animales adultos que mueren por falta de comida:

$$r_{44} \equiv [Z_{ij} h_s^{k_{1,4}} \xrightarrow{g_{i,1}} H_i'^{f_{i,3}} F_i'^{f_{i,4}} B'^{f_{i,3}} M'^{f_{i,4}}]_2^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,3} \leq j \leq g_{i,6}, \\ t+1 \leq s \leq D+t \end{cases}$$

$$r_{45} \equiv [Z_{ij} h_s^{k_{1,4}} \xrightarrow{1-g_{i,1}} \lambda]_2^0, \begin{cases} 1 \leq i \leq n, \\ g_{i,3} \leq j \leq g_{i,6}, \\ t+1 \leq s \leq D+t \end{cases}$$

El propósito de las siguientes reglas es eliminar los objetos H y F asociados con la cantidad de biomasa dejada por cada especie.

$$r_{46} \equiv [H_i \rightarrow \lambda]_1^0,\ 1 \leq i \leq n.$$

$$r_{47} \equiv [F_i \rightarrow \lambda]_1^0,\ 1 \leq i \leq n.$$

En la figura 6.2 se puede observar un diagrama de módulos que corresponde a la dinámica del modelo diseñado.

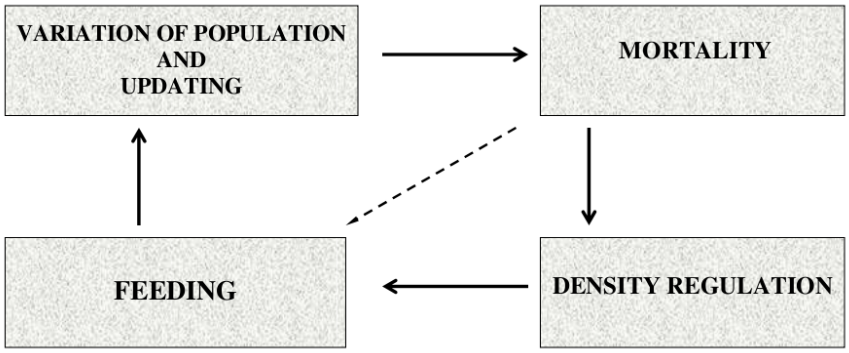


Figura 6.2: Módulos del modelo diseñado

6.1.2. Un simulador basado en P-Lingua

El simulador desarrollado recibe el nombre de *EcoSim 2.0 Bearded Vulture* y se puede descargar de la página web `http://www.p-lingua.org`, junto con su código fuente, la documentación técnica y el manual de usuario. Dicho simulador está compuesto por la biblioteca *pLinguaCore* para realizar simulaciones de sistemas P y por una interfaz gráfica de usuario desarrollada *ad hoc* para el ecosistema del quebrantahuesos. La aplicación permite los modos de funcionamiento y los casos de uso generales descritos en el capítulo anterior para la familia de software *EcoSim 2.0*:

- Depuración del modelo mediante simulaciones paso a paso, obteniendo información detallada sobre cada paso de computación simulado.
- Definición del número de pasos de computación que corresponden a un año en el ecosistema.
- Definición del número de años a simular y el número total de simulaciones a realizar.
- Edición de los parámetros iniciales del ecosistema, tales como constantes probabilísticas y poblaciones iniciales, así como la posibilidad de salvar y cargar estos datos.
- Selección del fichero de P-Lingua que especifica la familia de sistemas P que modeliza el ecosistema.
- Generación de tablas y gráficas de resultados representando la evolución del ecosistema a lo largo de los años simulados.

A continuación se detallan las peculiaridades de la interfaz gráfica desarrollada *ad hoc* para el ecosistema del quebrantahuesos. Cabe recordar que existen dos tipos de usuario: el *diseñador* que se encarga de definir, depurar y validar el modelo; y el *ecólogo/experto* que utiliza la aplicación para ejecutar experimentos virtuales a partir de diferentes escenarios iniciales que les puede resultar interesantes.

La interfaz gráfica de usuario permite al usuario ecólogo/experto introducir todos los parámetros iniciales del ecosistema del quebrantahuesos, mediante una serie de tablas de entrada implementadas con Java Swing. Todos los valores introducidos pueden salvarse en ficheros con formato binario y extensión *.ec2*. Los parámetros que el usuario ecólogo puede configurar son:

- Poblaciones iniciales.
- Cantidad de recursos de biomasa iniciales.
- Densidad máxima de población para cada especie.
- Constantes probabilísticas asociadas a cada especie.

Como datos de salida, la interfaz presenta las tablas y las gráficas de población media para cada una de las especies del ecosistema durante los años simulados, incluyendo intervalos de confianza basados en la desviación típica. Las gráficas presentadas en la Figura 6.1 han sido generadas por *EcoSim 2.0 Bearded Vulture*.

La aplicación también genera tablas y gráficas para representar la biomasa producida por especie, año y tipo (carne o huesos). A este respecto, se presentan dos gráficas de barras que muestran la evolución de la biomasa total durante los años simulados, uno corresponde a la biomasa de huesos y otro a la de carne. La aplicación también permite la generación interactiva de gráficas, donde el usuario selecciona un conjunto de especies productoras de biomasa, determina un tipo de biomasa y elige una especie carroñera. A partir de esos datos, la aplicación genera una gráfica de la biomasa producida por las especies seleccionadas a lo largo de la simulación, indicando con distintos colores la biomasa que genera cada una de las especies. Además, sobre esta gráfica de barras se superpone una gráfica lineal indicando las necesidades alimentarias de la especie carroñera elegida. En la Figura 6.3 se puede ver una pantalla del programa durante el proceso de generación de gráficas interactivas. Finalmente, cabe decir que este simulador ha sido utilizado para realizar la validación experimental del modelo y actualmente está siendo utilizado por los expertos

como una herramienta de trabajo para la toma de decisiones sobre el ecosistema objeto de estudio.

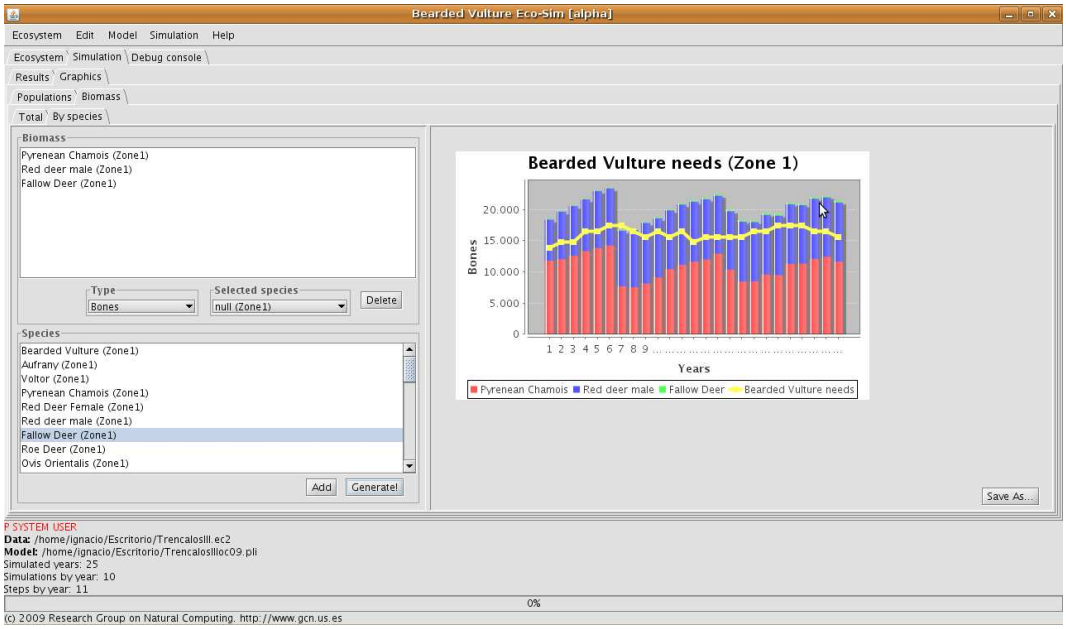


Figura 6.3: La interfaz gráfica de usuario de *EcoSim 2.0 Bearded Vulture*

6.2. Un ecosistema real relacionado con el mejillón cebra

El mejillón cebra (*Dresseina polymorpha*) es un tipo de bivalvo no comestible, que es una especie exótica invasora en muchos lugares de la Península Ibérica y Europa. Este tipo de mejillón invade los pantanos y ríos, desplazando a las especies autóctonas y provocando graves daños ecológicos y económicos, especialmente en presas hidráulicas y tuberías, al adherirse a las estructuras.

El objetivo de este capítulo es la presentación de un modelo computacional de un ecosistema relacionado con el mejillón cebra en el pantano de Ribarroja y

gestionado por Endesa. Se trata de conocer la evolución de esta especie en dicho ecosistema con el fin de analizar posibles medidas de gestión del embalse que permita controlar su invasión y amortiguar los efectos negativos que provoca esa presencia.

El mejillón cebra tiene una esperanza de vida de 3 años [118], con una proporción sexual de 1:1. La reproducción suele tener lugar cuando la temperatura del agua alcanza los 15 ^{o}C aproximadamente.

En una primera fase, se producen larvas velígeras que permanecen creciendo en la columna de agua por un periodo de 15 a 28 días [67], tras ese periodo, los mejillones se adosan al substrato. A lo largo del año, el mejillón cebra presenta dos ciclos reproductivos, siendo el primero de ellos el más importante. En el primer ciclo, las larvas nacen a lo largo de la primavera y pueden alcanzar la madurez sexual antes de que comience el segundo ciclo reproductivo, que tiene lugar al final del verano. En cualquier caso, el número de descendientes depende de la edad del mejillón adulto.

Los mejillones cebra se pueden unir unos a otros formando grandes aglomeraciones o columnas compactas que pueden alterar la vida en el ecosistema o producir importantes daños materiales. Cuando la densidad de estas aglomeraciones supera los 250000 individuos por m^2, los mejillones que se encuentran en la parte inferior mueren debido a que no pueden obtener nutrientes u oxígeno del entorno.

El ecosistema objeto de estudio se encuentra en el pantano de Ribarroja, localizado en el noreste de la Península Ibérica. Se trata de un entorno natural en donde se encuentra una gran variedad de especies autóctonas y en donde la pesca es una de las actividades principales. Su longitud es aproximadamente de 28 Km, su profundidad máxima es de 28 m, la mínima es de 2 m y su capacidad total es de 210 Hm^3.

La presencia de Dresseina polymorpha en el pantano fue detectada a mediados del año 2001 [108]. La rápida expansión de este tipo de mejillón ha acarreado graves alteraciones en el desarrollo sostenible de muchos ecosistemas, lo que ha convertido a dicha especie en objetivo de numerosas investigaciones.

En la actualidad los expertos han obtenido una gran cantidad de datos acerca de dicho ecosistema real, lo cual posibilitan el desarrollo de un modelo formal susceptible de ser validado experimentalmente.

El pantano recibe agua de otros pantanos y de tres ríos. El agua más caliente y superficial proviene de los ríos, mientras que el agua más fría y profunda procede de los pantanos. Tras la realización de exhaustivos estudios se ha llegado a la conclusión de la existencia de 17 zonas bien diferenciadas en el pantano, que pueden ser analizadas en base a las fluctuaciones de temperatura. La primera consecuencia de esta estructuración es que los ciclos de reproducción del mejillón comienzan en diferentes momentos en función de las distintas zonas consideradas. En segundo lugar, hay que destacar el hecho de que el substrato del pantano es diferente en cada zona, de tal manera que la densidad de mejillones varía según las zonas; por ejemplo, si un mejillón cae en una superficie arenosa entonces no puede sobrevivir, en cambio no sucede lo mismo con otro tipo de sustrato existente en el pantano.

6.2.1. Diseño de un modelo basado en sistemas P

En [21] se presentó un primer modelo que actualmente se encuentra en proceso de validación experimental. La biología y el ciclo vital de los mejillones en el pantano es muy compleja y, por ese motivo, sólo se han modelizado los procesos de reproducción, mortalidad, fijación al substrato, movimiento de larvas en la columna de agua y control de densidades. A pesar de ello, el modelo diseñado parece ser bastante prometedor, a juicio de los expertos de Endesa S.A., una vez analizados los resultados obtenidos a partir de las primeras simulaciones.

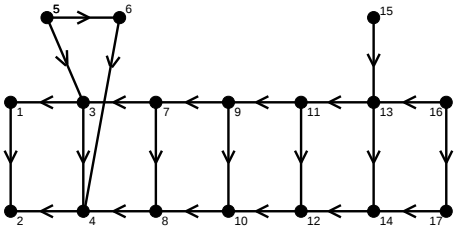
La temperatura del agua es un aspecto esencial para el desarrollo de los mejillones y ha sido contemplada en el modelo. Por otra parte, también se ha tenido en cuenta el efecto de la intervención humana en la gestión anual del pantano debido a las alteraciones que provoca el movimiento de larvas entre las distintas zonas.

El modelo propuesto consiste en el siguiente sistema P multientorno funcional–probabilístico con membranas activas de grado (5,17) usando T unidades de tiempo:

$$(G, \Gamma, \Sigma, R_E, \Pi, \{f_{r,j} : \ r \in R_\Pi, \ 1 \leq j \leq 17\}, \{\mathcal{M}_{ij} : \ 0 \leq i \leq 4, \ 1 \leq j \leq 17\})$$

En donde:

- El grafo del sistema es $G = (V, S)$, donde $V = \{e_1, \ldots, e_{17}\}$ y los arcos son (v, v), para cada $v \in V$, añadiendo los representados en la siguiente figura:



- El alfabeto de trabajo es:

$$\begin{aligned}
\Gamma \ = \ & \{X_{s,c,a}, \ Y_{s,c,a} : \ 1 \leq s \leq 6, \ 1 \leq c \leq 2, \ 1 \leq a \leq T\} \cup \\
& \{W_{d,c,a}, \ Q_{d,c,a} : \ 0 \leq d \leq 189, \ 1 \leq c \leq 2, \ 1 \leq a \leq T\} \cup \\
& \{WE_{c,a,s} : \ 1 \leq c \leq 2, \ 1 \leq a \leq T, 1 \leq s \leq 17\} \cup \\
& \{L_{dc} : \ 0 \leq d \leq 28, \ 1 \leq c \leq 2\} \cup \ \{t_i, \ t'_i : \ 1 \leq i \leq 3\} \cup \\
& \{D_{cdf} : \ 1 \leq \ c \leq 2, 1 \leq d \leq 14, 1 \leq f \leq 7\} \cup \{D_c : \ 1 \leq c \leq 2\} \cup \\
& \{a_s : \ 0 \leq s \leq 5\} \cup \ \{c_i : \ 1 \leq i \leq 2\} \cup \{e_i : \ 1 \leq i \leq 6\} \cup \\
& \{f_i : \ 1 \leq i \leq 5\} \cup \ \{t, \ b, \ M, \ e, \ f\} \cup \ \{\eta_i : \ 0 \leq i \leq 153\}
\end{aligned}$$

  En donce los símbolos X e Y están asociados a diferentes fases de la vida del mejillón. L está asociado a larvas y W, WE, Q representan mejillones nacidos en el ciclo actual. El objeto M representa las muerte de un mejillón. El objeto D permite conocer el momento en el que la temperatura alcanza el límite de reproducción. El objeto t_i es una señal que indica el comienzo de la reproducción en la membrana i. El objeto a_s permite controlar la densidad de mejillones. El objeto c_i establece el estado actual del ciclo reproductivo.

- Los objetos que pueden estar presentes en el entorno son los pertenecientes al siguiente alfabeto:

$$\Sigma = \{WE_{c,a,s},\ W_{1,c,a} :\ 1 \leq c \leq 2,\ 1 \leq a \leq T,\ 1 \leq s \leq 17\}$$

- El conjunto R_E de reglas del entorno consiste en:

$$r_{22,c,a,s} \equiv \left(WE_{c,a,s} \right)_{e_j} \ \rightarrow\ \left(W_{1,c,a} \right)_{e_s},\ \begin{cases} 1 \leq c \leq 2, \\ 1 \leq a \leq T, \\ (e_j, e_s) \in S, \\ j \neq s \end{cases}$$

$$r_{23,c,a,j} \equiv \left(WE_{c,a,j} \right)_{e_j} \ \rightarrow\ \left(W_{1,c,a} \right)_{e_j},\ \begin{cases} 1 \leq c \leq 2, \\ 1 \leq a \leq T, \\ 1 \leq j \leq 17 \end{cases}$$

- $\Pi = (\Gamma, \mu, R_\Pi)$ es el esqueleto de un sistema P funcional–probabilístico con polarización, de grado 5, cuya estructura de membranas es: $\mu = [\ [\ ]_1[\ ]_2[\ ]_3[\ ]_4\ ]_0$.

- $\Pi_j = (\Gamma, \mu, R_{\Pi_j}, T, \{f_{r,j} :\ r \in R_\Pi\}, \mathcal{M}_{0j}, \ldots, \mathcal{M}_{q-1,j})$, $1 \leq j \leq 17$, es un sistema P funcional–probabilístico con membranas activas de grado 5 usando T unidades de tiempo.

  - Para cada r, $r \neq 20, 21$, y cada j, $1 \leq j \leq 17$, $f_{r,j}$ es una función computable definida por $f_{r,j}(a) = k_{r,j}$, para cada $a = 1, 2, \ldots, T$, y $k_{r,j}$ es un número real entre 0 y 1, asociado con r y j.
    - $f_{r_{20},j}(a) = F(\varphi(a), j, s)$, donde $\varphi(a)$ es el tipo de gestión del pantano para el año a $(a = 1, 2, \ldots, T)$, y $F(x, y, z)$ es un número real entre 0 y 1 proporcionando la probabilidad de intercambiar larvas entre entornos.
    - $f_{r_{21},j}(a) = F(\varphi(a), j, 0)$ proporciona la probabilidad de que las larvas desaparezcan del sistema P.

- El conjunto R_{Π_j} consta de las siguientes reglas:

  Sincronización. En los pasos 103 y 136 se produce un proceso de sincronización.

  $$r_1 \equiv [\eta_i]_0^0 \rightarrow [\eta_{i+1}]_0^0,\ 0 \leq i \leq 151, i \neq 103, i \neq 136$$

  Simulación de temperaturas. La señal de activación del ciclo de reproducción se obtiene cuando se alcanza la temperatura adecuada; por ello, es necesario simular la temperatura del agua. En el modelo se considera que existen tres intervalos de temperatura: cuando la temperatura alcanza un valor del menor intervalo, entonces un porcentaje de mejillones adultos inicia la reproducción; cuando se alcanza un valor del segundo intervalo, otro porcentaje inicia la reproducción; y cuando se alcanzan temperaturas del intervalo superior, entonces el resto de los mejillones en edad reproductiva inician la reproducción. Las reglas que regulan este proceso son las siguientes:

  $$r_{2c} \equiv [D_c \xrightarrow{f_{r_{2c},j}} \lambda]_0^0,\ \begin{cases} 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

  $$r_{3c} \equiv [D_c \xrightarrow{1-f_{r_{2c},j}} D_{c,1,1}]_0^0,\ \begin{cases} 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

  $$r_{4cdf} \equiv [D_{c,d,f} \xrightarrow{f_{r_{4cf3},j}} t_1, t_2, t_3]_0^0,\ \begin{cases} 1 \leq d \leq 14, \\ 1 \leq f \leq 7, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

  $$r_{5cdf} \equiv [D_{c,d,f} \xrightarrow{f_{r_{5cf2},j}} t_1, t_2, D_{c,d+1,f}]_0^0,\ \begin{cases} 1 \leq d \leq 14, \\ 1 \leq f \leq 7, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{6cdf} \equiv [D_{c,d,f} \xrightarrow{f_{r_{6cf1},j}} t_1, D_{c,d+1,f}]_0^0, \begin{cases} 1 \leq d \leq 14, \\ 1 \leq f \leq 7, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{7cdf} \equiv [D_{c,d,f} \xrightarrow{1-\sum_{i=4}^{i=7} f_{r_{icf(7-i)},j}} D_{c,d+1,f}]_0^0, \begin{cases} 1 \leq d \leq 14, \\ 1 \leq f \leq 7, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{8cf} \equiv [D_{c,14,f} \xrightarrow{f_{r_{8cf3},j}} t_1, t_2, t_3]_0^0, \begin{cases} 1 \leq f \leq 7, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{9cf} \equiv [D_{c,14,f} \xrightarrow{f_{r_{9cf2},j}} t_1, t_2, D_{c1f+1}]_0^0, \begin{cases} 1 \leq f \leq 7, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{10cf} \equiv [D_{c,14,f} \xrightarrow{f_{r_{10cf1},j}} t_1, D_{c,1,f+1}]_0^0, \begin{cases} 1 \leq f \leq 7, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{11cf} \equiv [D_{c,14,f} \xrightarrow{1-\sum_{i=8}^{i=10} f_{r_{icf(11-i)},j}} D_{c,1,f+1}]_0^0, \begin{cases} 1 \leq f \leq 7, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{12} \equiv t_m[\ ]_m^0 \rightarrow [t]_m^+,\ 1 \leq m \leq 3$$

$$r_{13} \equiv [t]_m^+ \rightarrow t[\ ]_m^0,\ 1 \leq m \leq 3$$

Proceso de reproducción. El proceso de reproducción tiene lugar en las membranas 1, 2 y 3, dentro de las cuales están localizados los objetos $X_{s,c,a}$ y $Q_{d,c,a}$ que representan mejillones adultos y mejillones nacidos en el ciclo previo, respectivamente. Las reglas asociadas a este proceso son las siguientes:

$$r_{14} \equiv [Q_{d,c,a}]_m^+ \xrightarrow{0,5} Y_{1,c,a} L_{0,c,a}^{g_{c,1}} [\ ]_m^0, \begin{cases} 1 \leq a \leq T, \\ n(j) \leq d \leq 180, \\ 1 \leq c \leq 2, \\ 1 \leq m \leq 3, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{15} \equiv [Q_{d,c,a}]_m^+ \xrightarrow{0,5} Y_{1,c,a} [\ ]_m^0, \begin{cases} 1 \leq a \leq T, \\ n(j) \leq d \leq 180, \\ 1 \leq c \leq 2, \\ 1 \leq m \leq 3, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{16} \equiv [Q_{d,c,a}]_m^+ \to Y_{1,c,a} [\ ]_m^0, \begin{cases} 1 \leq a \leq T, \\ 0 \leq d < n(j), \\ 1 \leq c \leq 2, \\ 1 \leq m \leq 3, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{17} \equiv [X_{s,c,a}]_m^+ \xrightarrow{0,5} Y_{s,c,a} L_{0,c,a}^{g_{c,s}} [\ ]_m^0, \begin{cases} 1 \leq a \leq T, \\ 1 \leq s \leq 6, \\ 1 \leq c \leq 2, \\ 1 \leq m \leq 3, \end{cases}$$

$$r_{18} \equiv [X_{s,c,a}]_m^+ \xrightarrow{0,5} Y_{s,c,a} [\ ]_m^0, \begin{cases} 1 \leq a \leq T, \\ 1 \leq s \leq 6, \\ 1 \leq c \leq 2, \\ 1 \leq m \leq 3, \end{cases}$$

Crecimiento de larvas. El crecimiento de las larvas se puede producir en el entorno donde nacieron, o bien se desplaza otros entornos. Las reglas que desarrollan este proceso son las siguientes:

$$r_{19} \equiv [L_{d,c,a} \to L_{d+1,c,a}]_0^0, \begin{cases} 0 \leq d < 28, \\ 1 \leq a \leq T, \\ 1 \leq c \leq 2 \end{cases}$$

$$r_{20} \equiv [L_{28,c,a}]_0^0 \xrightarrow{f_{r_{20},j}} WE_{c,a,s}[\ ]_0^0, \begin{cases} 1 \leq a \leq T, \\ 1 \leq c \leq 2, \\ 1 \leq s \leq 17, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{21} \equiv [L_{28,c,a} \xrightarrow{f_{r_{21},j}} \lambda]_0^0, \begin{cases} 1 \leq a \leq T, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

Entrada de larvas en el entorno

$$r_{24} \equiv W_{1,c,a}[\ ]_0^0 \rightarrow [W_{2,c,a}]_0^0, \begin{cases} 1 \leq a \leq T, \\ 1 \leq c \leq 2 \end{cases}$$

Mortalidad asociada con la senescencia.

$$r_{25} \equiv [Y_{6,c,a} \rightarrow M]_0^0, \begin{cases} 1 \leq a \leq T, \\ 1 \leq c \leq 2 \end{cases}$$

Todos los mejillones entran en la membrana 4.

$$r_{26} \equiv Y_{s,c,a}[\ ]_4^0 \rightarrow [Y_{s,c,a}]_4^0, \begin{cases} 1 \leq s \leq 5, \\ 1 \leq a \leq T, \\ 1 \leq c \leq 2 \end{cases}$$

$$r_{27} \equiv W_{2,c,a}[\ ]_4^0 \rightarrow [Q_{3,c,a}]_4^0, \begin{cases} 1 \leq a \leq T, \\ 1 \leq c \leq 2 \end{cases}$$

$$r_{28} \equiv [Q_{d,c,a} \rightarrow Q_{d+1,c,a}]_4^0, \begin{cases} 3 \leq d \leq 180, \\ 1 \leq a \leq T, \\ 1 \leq c \leq 2 \end{cases}$$

El ciclo de reproducción no tiene lugar. En el caso de que no se verifiquen las condiciones necesarias para el comienzo del ciclo reproductivo, se utilizan las siguientes reglas, a fin de que el proceso continúe.

$$r_{29} \equiv [\eta_{103} \rightarrow t'_1\ t'_2\ t'_3\ \eta_{104}]_0^0$$

$$r_{30} \equiv t'_m[\ ]^0_m \rightarrow [t]^-_m,\ 1 \leq m \leq 3$$

$$r_{31} \equiv [X_{s,c,a}]^-_m \rightarrow Y_{s,c,a}[\ ]^0_m,\ \begin{cases} 1 \leq s \leq 6, \\ 1 \leq a \leq T, \\ 1 \leq c \leq 2, \\ 1 \leq m \leq 3 \end{cases}$$

$$r_{32} \equiv [Q_{d,c,a}]^-_m \rightarrow Y_{1,c,a}[\ ]^0_m,\ \begin{cases} 1 \leq d \leq 180, \\ 1 \leq a \leq T, \\ 1 \leq c \leq 2, \\ 1 \leq m \leq 3 \end{cases}$$

$$r_{33} \equiv [t]^-_m \rightarrow t[\ ]^0_m,\ 1 \leq m \leq 3$$

Viabilidad de mejillones asociada a la densidad de población. En este proceso, se analiza la viabilidad de mejillones jóvenes y adultos. Para los primeros, la viabilidad depende del tipo de substrato y para los segundos, depende de su posición.

$$r_{34} \equiv \eta_{138}[\ ]^0_4 \rightarrow \eta_{139}[b]^+_4$$

$$r_{35} \equiv [h]^+_4 \rightarrow b\big[a_0^{B(k)*250000}\big]^0_4,\ 1 \leq k \leq 17$$

$$r_{36} \equiv [a_s]^+_4 \rightarrow b[a_{s+1}]^0_4,\ 0 \leq s \leq 4$$

$$r_{37} \equiv [e]^+_4 \rightarrow b[f_1]^0_4$$

$$r_{38} \equiv [e_i]^+_4 \rightarrow b[f_i]^0_4,\ 1 \leq i \leq 5$$

$$r_{39} \equiv [f_i]^0_4 \rightarrow b[e_{i+1}]^+_4,\ 1 \leq i \leq 5$$

$$r_{40} \equiv [e_6]^+_4 \rightarrow b[f]^0_4$$

$$r_{41} \equiv [Q_{d,c,a}a_0]^0_4 \rightarrow Q_{d+1,c,a}[\ ]^+_4,\ \begin{cases} 1 \leq a \leq T, \\ 3 \leq d \leq 180, \\ 1 \leq c \leq 2 \end{cases}$$

$$r_{42} \equiv [Y_{s,c,a}a_s]_4^0 \to X_{s+1,c+(-1)^{c+1},a}[\ ]_4^+, \ \begin{cases} 1 \leq a \leq T, \\ 1 \leq s \leq 5, \\ 1 \leq c \leq 2 \end{cases}$$

$$r_{43m} \equiv X_{s,c,a}[\ ]_m^0 \xrightarrow{f_{r_{43m},j}} [X_{s,c,a}]_m^0, \ \begin{cases} 1 \leq a \leq T, \\ 1 \leq m \leq 3, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{44m} \equiv Q_{d,c,a}[\ ]_m^0 \xrightarrow{f_{r_{44m},j}} [Q_{d+1+(c-1)*60,c+(-1)^{c+1},a+(c-1)}]_m^0,$$

$$\begin{cases} 1 \leq a \leq T, \\ 0 \leq d \leq 121, \\ 1 \leq m \leq 3, \\ 1 \leq c \leq 2, \\ 1 \leq j \leq 17 \end{cases}$$

$$r_{45} \equiv [Q_{d,c,a} \to Q_{d+1,c,a}]_m^0, \ \begin{cases} 1 \leq a \leq T, \\ 5 \leq d \leq 121, \\ 1 \leq m \leq 3, \\ 1 \leq c \leq 2 \end{cases}$$

Preparación para el principio de un nuevo ciclo. Con el fin de dar por finalizado un ciclo y comenzar el siguiente, hay que proceder a la eliminación de objetos creados y a la reinicialización de contadores. Esto se consigue a través de las siguientes reglas:

$$r_{46} \equiv \eta_{152}[\ ]_4^0 \to \eta_{153}[h]_4^-$$

$$r_{47} \equiv [Y_{s,c,a}]_4^- \to M[\ ]_4^0, \ \begin{cases} 1 \leq s \leq 5, \\ 1 \leq c \leq 2, \\ 1 \leq a \leq T \end{cases}$$

$$r_{48} \equiv [Q_{d,c,a}]_4^- \to M[\ ]_4^0, \ \begin{cases} 1 \leq d \leq 121, \\ 1 \leq c \leq 2, \\ 1 \leq a \leq T \end{cases}$$

$$r_{49} \equiv [a_s]_4^- \to b[\ ]_4^0, \ 0 \leq s \leq 6$$

$$r_{50} \equiv [b]_4^- \rightarrow b[\ ]_4^0$$

$$r_{51} \equiv [b \rightarrow \lambda]_0^0$$

$$r_{52} \equiv [t \rightarrow \lambda]_0^0$$

$$r_{53} \equiv [\eta_{153} \rightarrow \eta_0]_0^0$$

$$r_{54} \equiv [c_i]_4^- \rightarrow D_{i+(-1)^{i+1}}[c_{i+(-1)^{i+1}}]_4^0,\ 1 \leq i \leq 2$$

$$r_{55} \equiv [f]_4^- \rightarrow b[e]_0^4$$

- $\mathcal{M}_{0j},\ \mathcal{M}_{1j},\ \mathcal{M}_{2j},\ \mathcal{M}_{3j}$ y $\mathcal{M}_{4j}$ $(1 \leq j \leq 17)$ son multiconjuntos de objetos sobre Γ localizados inicialmente en las distintas regiones de μ:

$$\mathcal{M}_{ij} = \{X_{s,1,1}^{q_{ij}}\},\ \begin{cases} 1 \leq i \leq 3, \\ 1 \leq j \leq 17, \\ 1 \leq s \leq 6 \end{cases}$$

$$(\mathcal{M}_4)_e = \{h, c_1, e\},\ 1 \leq e \leq 17$$

$$(\mathcal{M}_0)_e = \{\eta_0,\ T_1\},\ 1 \leq e \leq 17$$

La información necesaria es la siguiente:

- Cantidad de mejillones en cada zona: q_{ij}, con $1 \leq i \leq 3$ y $1 \leq j \leq 17$.
- Temperaturas medias a lo largo del año: son necesarias para calcular las probabilidades asociadas a las reglas de la r_2 a la r_{11}.
- Porcentaje de mejillones que inician el ciclo reproductivo dependiendo de la temperatura: $f_{r43,m,j}$ y $f_{r44,m,j}$, con $1 \leq m \leq 3$ y $1 \leq j \leq 17$.

- Cantidad de larvas viables por mejillón en función de la edad: $g_{c,s}$, con $1 \leq c \leq 2$ y $1 \leq s \leq 6$.
- Probabilidad de que las larvas se muevan entre entornos: f_{r20} y f_{r21}.
- Densidad máxima de larvas vivas en cada zona: 250.000 ind/m^2.
- Edad expresada en días en la cual se alcanza la madurez sexual: $n(j)$, con $1 \leq j \leq 17$.
- Tipo de substrato por zona: $B(k)$, con $1 \leq k \leq 17$.

En la figura 6.4 se puede observar un diagrama de módulos que corresponde a la dinámica del modelo diseñado.

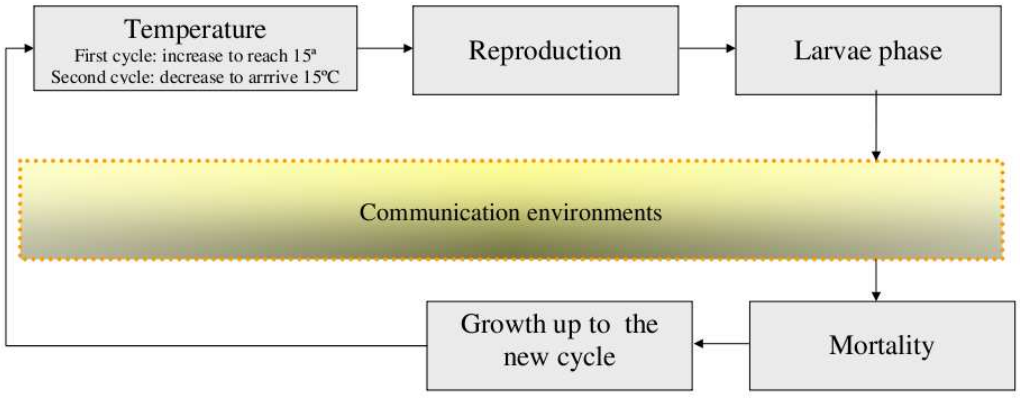


Figura 6.4: Módulos del modelo diseñado

6.2.2. Un simulador basado en P-Lingua

El simulador desarrollado recibe el nombre de *EcoSim 2.0 Zebra Mussel* y se puede descargar de la página web `http://www.p-lingua.org`, junto con su código fuente, la documentación técnica y el manual de usuario.

De manera análoga al resto de las aplicaciones de la familia de software *EcoSim 2.0*, este simulador permite los modos de funcionamiento y los casos de uso generales descritos en los capítulos previos:

- Depuración del modelo mediante simulaciones paso a paso, obteniendo información detallada sobre cada paso de computación simulado.
- Definición del número de pasos de computación que corresponden a un año en el ecosistema.
- Definición del número de años a simular y el número total de simulaciones a realizar.
- Edición de los parámetros iniciales del ecosistema, tales como constantes probabilísticas y poblaciones iniciales, así como la posibilidad de salvar y cargar estos datos.
- Selección del fichero de P-Lingua que especifica la familia de sistemas P que modeliza el ecosistema.
- Generación de tablas y gráficas de resultados representando la evolución del ecosistema a lo largo de los años simulados.

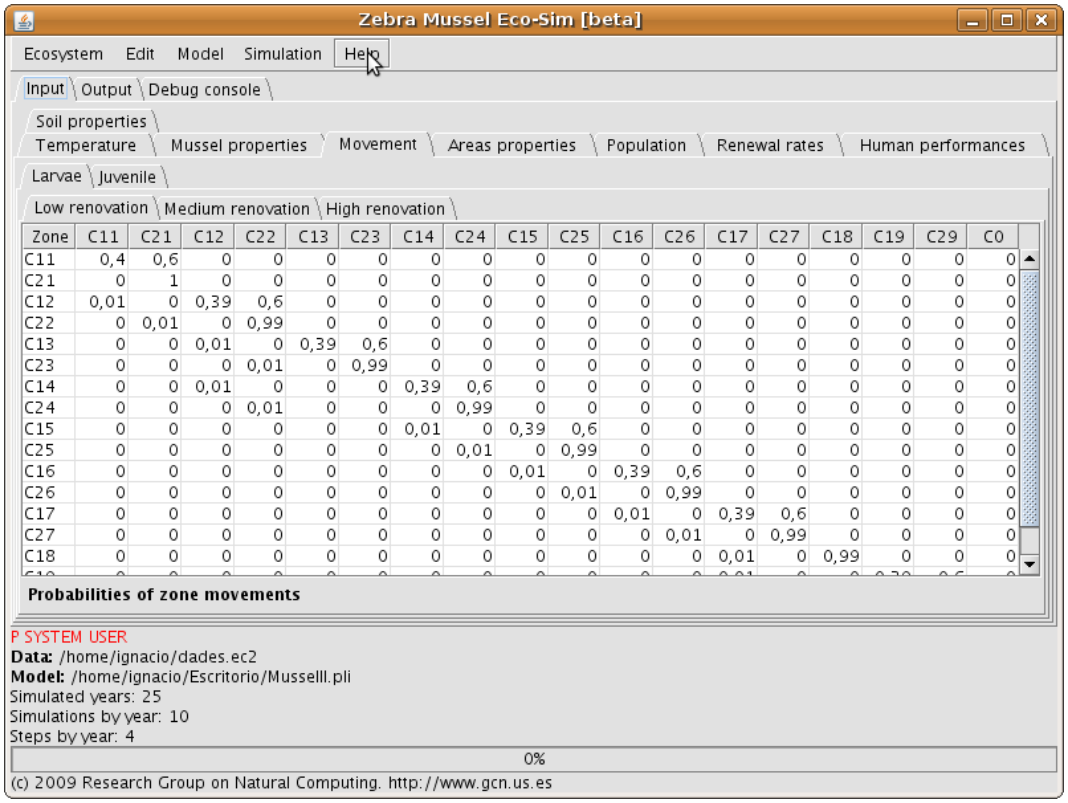
A continuación se detallan las peculiaridades de la interfaz gráfica desarrollada *ad hoc* para el ecosistema del mejillón cebra. Cabe recordar que existen dos tipos de usuario: el *diseñador* que se encarga de definir, depurar y validar el modelo, y el *ecólogo/experto* que utiliza la aplicación para realizar experimentos virtuales a partir de diferentes escenarios.

La interfaz gráfica de usuario permite al usuario ecólogo introducir los parámetros iniciales del modelo, mediante el uso de tablas:

- Poblaciones iniciales de mejillones y larvas.
- Características físicas del pantano.
- Valores medios de temperatura para cada zona.

- Densidades máximas de población para cada zona del pantano.
- Constantes probabilísticas asociadas a mejillones y larvas.

Entre las constantes probabilísticas, se incluyen las probabilidades de desplazamiento de larvas entre distintas zonas del pantano. En la figura 6.5 se puede ver una imagen de la interfaz gráfica en el momento de la edición de las probabilidades de movimiento de larvas entre zonas.



Zone	C11	C21	C12	C22	C13	C23	C14	C24	C15	C25	C16	C26	C17	C27	C18	C19	C29	C0
C11	0,4	0,6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C21	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C12	0,01	0	0,39	0,6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C22	0	0,01	0	0,99	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C13	0	0	0,01	0	0,39	0,6	0	0	0	0	0	0	0	0	0	0	0	0
C23	0	0	0	0,01	0	0,99	0	0	0	0	0	0	0	0	0	0	0	0
C14	0	0	0,01	0	0	0	0,39	0,6	0	0	0	0	0	0	0	0	0	0
C24	0	0	0	0,01	0	0	0	0,99	0	0	0	0	0	0	0	0	0	0
C15	0	0	0	0	0	0	0,01	0	0,39	0,6	0	0	0	0	0	0	0	0
C25	0	0	0	0	0	0	0	0,01	0	0,99	0	0	0	0	0	0	0	0
C16	0	0	0	0	0	0	0	0	0,01	0	0,39	0,6	0	0	0	0	0	0
C26	0	0	0	0	0	0	0	0	0	0,01	0	0,99	0	0	0	0	0	0
C17	0	0	0	0	0	0	0	0	0	0	0,01	0	0,39	0,6	0	0	0	0
C27	0	0	0	0	0	0	0	0	0	0	0	0,01	0	0,99	0	0	0	0
C18	0	0	0	0	0	0	0	0	0	0	0	0	0,01	0	0,99	0	0	0

Figura 6.5: La interfaz gráfica de usuario de *EcoSim 2.0 Zebra Mussel*

Como datos de salida, la interfaz presenta las tablas y las gráficas de población de larvas y mejillones adultos según la zona del pantano y el año simulado. En cada proceso de simulación se ejecutan varias simulaciones independientes y es posible consultar los resultados de cualquiera de las simulaciones realizadas o calcular valores medios.

Actualmente el simulador se está utilizando para la validación experimental del modelo, mediante la comparación con datos reales obtenidos experimentalmente por los expertos de Endesa.

Parte IV

Conclusiones y líneas de trabajo futuro

Capítulo 7

Conclusiones y líneas de trabajo futuro

7.1. Resumen de lo desarrollado

Un marco para la modelización basado en sistemas P

En esta memoria se presenta un marco teórico para la modelización de fenómenos complejos basado en sistemas P; en particular, el marco se puede aplicar a procesos biológicos, ya sean a nivel macroscópico o microscópico. La aleatoriedad inherente de estos procesos queda capturada por la semántica del modelo mediante estrategias estocásticas o probabilísticas.

Un entorno de programación para *Membrane Computing*

Esta memoria presenta el primer desarrollo de un entorno de programación para facilitar el desarrollo de aplicaciones informáticas en *Membrane Computing*.

Después de analizar detenidamente las diferentes aplicaciones informáticas para *Membrane Computing* producidas a lo largo de diez años, se infiere que

todas ellas contienen algunos elementos estructurales en común:

- La definición del sistema P a simular.
- El núcleo de simulación.
- La presentación de resultados al usuario.

En el presente trabajo se aportan soluciones generales para cada uno de los módulos, con la finalidad de facilitar el desarrollo de futuras aplicaciones.

P-Lingua: un lenguaje de programación para Membrane Computing

Como solución para el problema de definición de sistemas P, se ha desarrollado un nuevo lenguaje de programación. P-Lingua es un lenguaje sencillo, debido a que su sintaxis es próxima a la notación científica con la cual suelen estar familiarizados los investigadores; es modular, debido a la posibilidad de escribir programas que se componen de ciertas partes independientes; y es paramétrico, debido a la posibilidad de usar índices, parámetros e iteradores para la definición de sistemas P.

P-Lingua soporta diferentes modelos de sistemas P que trabajan a modo de célula y a modo de tejido. Este lenguaje viene acompañado por una serie de herramientas para la línea de comandos, formando un entorno completo de programación. Dicho entorno permite:

- Compilar ficheros de texto en formato P-Lingua a otros formatos, consiguiendo interoperabilidad.
- Simular computaciones de los sistemas P definidos.

Al principio de cada fichero P-Lingua, se debe incluir una línea de código que especifica el tipo de modelo de sistemas P que se está utilizando. El compilador es capaz de identificar y localizar los errores léxico/sintácticos y semánticos (desde el punto de vista de la programación); por ejemplo, puede

identificar que una regla no está permitida en el modelo que se está usando o bien que su sintaxis no es correcta.

Uno de los objetivos que nos proponemos con el desarrollo de P-Lingua es que se convierta en un estándar para la definición de sistemas P. De esta manera, se consiguen las siguientes ventajas:

- Los mismos ficheros que definen sistemas P pueden ser usados en diferentes entornos de software.
- Se reduce el coste en tiempo y esfuerzo del usuario de P-Lingua a la hora de adaptarse a nuevas aplicaciones, ya que siguen el estándar.
- Gracias al desarrollo de bibliotecas de programación que procesan el lenguaje estándar, es posible reducir el tiempo y el esfuerzo del programador con el fin de producir nuevas aplicaciones para sistemas P.

Cabe decir que, en el momento de escribir estas líneas, el lenguaje de programación P-Lingua se está usando activamente por grupos de investigación de las Universidades de Sevilla, Lleida, Universidad Autónoma de Madrid, Pitesti (Rumanía) y Sheffield (U.K.).

Algoritmos de simulación para sistemas P

Con el objetivo de añadir algoritmos de simulación eficientes al marco de P-Lingua, se han diseñado varios algoritmos para cada uno de los modelos de sistemas P soportados. Estos algoritmos reproducen una computación paso a paso del sistema P especificado, según la semántica particular de cada modelo.

En el caso de los sistemas P probabilísticos, se ha diseñado un algoritmo eficiente basado en distribuciones binomiales que ha conseguido buenos resultados (en términos de tiempo y memoria) trabajando con sistemas P que contienen cientos de membranas, más de 2^{32} objetos por membrana y más de 10^5 reglas de evolución.

pLinguaCore: una biblioteca Java para el desarrollo de simuladores

Se ha desarrollado una biblioteca de programación en lenguaje Java con la finalidad de facilitar el desarrollo de futuros simuladores y su integración con P-Lingua. Dicha biblioteca aporta las siguientes funcionalidades:

- Lectura y análisis de ficheros que especifican sistemas P.
- Simulación de computaciones de los sistemas P definidos, a través de los algoritmos de simulación considerados.
- Exportación a otros formatos de fichero.

La biblioteca no es un producto cerrado, sino que se puede ampliar para permitir nuevos formatos de fichero, modelos de sistemas P y algoritmos de simulación.

Casos de estudio

Un marco general para la modelización y simulación de ecosistemas

En esta memoria se ha diseñado un marco general para la modelización de ecosistemas basado en sistemas P probabilísticos. Este marco no solamente es teórico, sino que se encuentra complementado de manera práctica por las aplicaciones informáticas desarrolladas en esta memoria. El objetivo de dichas aplicaciones es la validación experimental y la posibilidad de realizar experimentos virtuales de los modelos formales desarrollados.

De esta manera, se presenta la familia de software *EcoSim 2.0*, que consiste en un conjunto de interfaces de usuario sobre la biblioteca *pLinguaCore*, acompañados por ficheros P-Lingua que definen los modelos. Cada uno de estas interfaces está diseñada para un modelo de ecosistema concreto y permite a un usuario final ecólogo la realización de experimentos virtuales sin necesidad de entrar en detalles del modelo; es decir, la aplicación se comporta como una *caja negra* y realiza las simulaciones computacionales de manera transparente

al usuario ecólogo. En este sentido, se puede afirmar que *EcoSim 2.0* es una de las primeras aplicaciones informáticas basadas en *Membrane Computing* para ser usadas como herramienta instrumental en otras ramas de la ciencia.

EcoSim 2.0 también permite un modo de funcionamiento para el usuario que diseña el modelo; así, la aplicación se comporta como una *caja blanca* y permite la simulación de sistemas P probabilísticos paso a paso, con el objetivo de depurar y validar experimentalmente el modelo.

El modelo que define el ecosistema a través de un sistema P es escrito en P-Lingua por el usuario que lo diseña, sin necesidad de conocer otros lenguajes de programación.

Las adaptaciones específicas de *EcoSim 2.0* se mencionan a continuación.

Simulación de un ecosistema relacionado con el quebrantahuesos

El quebrantahuesos es una especie en peligro de extinción y en esta memoria se ha desarrollado la aplicación *EcoSim 2.0 Bearded Vulture* como una herramienta de asistencia a la toma de decisiones para la gestión y conservación de un ecosistema real relacionado con el quebrantahuesos en la zona Pirenáico-Catalana.

El modelo, desarrollado en colaboración con investigadores de la Universidad de Lleida y expertos en ecología, en particular, con A. Margalida que es una de las autoridades mundiales en el estudio del quebrantahuesos, incluye 13 especies animales y ha sido validado con datos reales de 14 años, obteniendo en todo caso un intervalo de confianza del 95 %.

La herramienta informática permite alterar las condiciones iniciales del ecosistema simulado para producir experimentos virtuales que informan acerca de la evolución de la población media de cada especie a lo largo de los años y de la biomasa producida por especie y año. También es posible la generación interactiva de gráficos. El objetivo final es la formulación de hipótesis que asistan a los ecólogos en la toma de decisiones, tras un proceso previo de filtrado realizado por expertos.

Cabe decir que esta herramienta está siendo utilizada a día de hoy por expertos en ecología como asistente informático en la toma de decisiones de gestión y conservación de un ecosistema real.

Simulación de un ecosistema relacionado con el mejillón cebra

El mejillón cebra es un tipo de bivalvo no comestible, considerado como especie invasora en muchos lugares de la Península Ibérica y Europa. Este tipo de mejillón invade los pantanos y ríos, desplazando a las especies autóctonas y provocando graves daños económicos en instalaciones construidas por el hombre, especialmente en presas hidráulicas, al adherirse a las estructuras.

En colaboración con Endesa S.A., investigadores de la Universidad de Lleida y expertos en ecología, se ha desarrollado el primer modelo computacional de un ecosistema relacionado con el mejillón cebra en el pantano de Ribarroja. Este modelo reproduce el comportamiento y la propagación del mejillón cebra desde su fase larvaria hasta su fase adulta, considerando factores como la temperatura y el desplazamiento de mejillones por el pantano mediante barcos de navegación.

La herramienta *EcoSim 2.0 Zebra Mussel* ha sido desarrollada con la finalidad de validar experimentalmente el modelo y asistir a la toma de decisiones sobre la gestión del pantano mediante la realización de experimentos virtuales. El objetivo final es controlar la población de mejillones en el pantano a través de decisiones de gestión apoyadas por el modelo computacional y la aplicación informática.

Actualmente el modelo se encuentra en su fase de validación experimental con datos reales, para lo cual se está usando el software desarrollado.

Página web y licencia de software

En la página web `http://www.p-lingua.org/` se puede encontrar información técnica sobre el lenguaje desarrollado, y es posible descargar todas las aplicaciones y bibliotecas presentadas en esta memoria junto con

la documentación técnica y los manuales de usuario. En la página se encuentra también un foro que sirve de punto de encuentro para la comunidad de usuarios y desarrolladores de P-Lingua, a fin de mejorar la aplicación informática.

Todas las aplicaciones y bibliotecas informáticas presentadas en esta memoria se encuentran al servicio de la comunidad científica bajo licencia de software libre GNU GPL [125].

7.2. Líneas futuras de investigación

En esta memoria se presentan los trabajos relacionados con el primer desarrollo de un marco para la simulación informática de sistemas P y, por tanto, existen diversas formas de continuar el trabajo iniciado a través de posibles líneas futuras de investigación, algunas de las cuales se enumeran a continuación.

Ampliación del marco de P-Lingua

Ampliación del lenguaje de programación

Se pretende que el lenguaje de programación P-Lingua sea un estándar para la definición de sistemas P. Su sintaxis es lo suficientemente flexible como para poder admitir sistemas P pertenecientes a la mayoría de los modelos existentes. Actualmente soporta la definición de sistemas P correspondientes a varios modelos *cell-like* y *tissue-like*, para cada uno es necesario incluir una línea al principio del fichero que especifica el modelo que se está usando.

Se propone:

- **Incluir nuevos modelos**

  La manera natural de ampliar P-Lingua es seguir añadiendo modelos de sistemas P al conjunto de los modelos soportados. Para ello es necesario adecuar la sintaxis del lenguaje para las particularidades

de cada modelo (intentando mantener la sencillez del lenguaje) y ampliar la biblioteca pLinguaCore para que el analizador de ficheros P-Lingua reconozca los nuevos modelos e identifique las correspondientes restricciones semánticas.

- **Definir sistemas P por *ingredientes***

  Otra posible manera de ampliar P-Lingua consiste en considerar que cada modelo se compone sintácticamente de una serie de *ingredientes*, tales como el uso de cargas eléctricas, permitir reglas de división, establecer una estructura tipo célula o tipo tejido, etc.

  La definición de sistemas P según modelos establecidos es una buena solución porque cada modelo está bien definido desde el punto de vista teórico y se pueden desarrollar simuladores consistentes. Pero cierra las puertas a la investigación de nuevas variantes de sistemas P. Sería muy interesante permitir de manera alternativa la definición de sistemas P sin ceñirse a ningún modelo concreto, sino enumerando los *ingredientes* que se permiten.

  Para ello, se propone añadir al lenguaje nuevas sentencias que permitan la *definición de modelos* como un conjunto de posibilidades permitidas. Y luego poder definir nuevos sistemas P de acuerdo a estos modelos.

  Estos nuevos módulos de definición de modelos podrían permitir heredar (o extender) otros módulos existentes, de esta manera se puede partir de un modelo conocido y realizar variaciones, incluso de manera paramétrica.

Desarrollo de simuladores eficientes

Diseño de nuevos algoritmos y estrategias de simulación

Existen uno o más posibles algoritmos de simulación para cada uno de los modelos definidos que reproducen una posible computación paso a paso buscando la máxima eficiencia.

Se propone:

- **Incluir nuevos algoritmos de simulación**

  Para cada nuevo modelo soportado por P-Lingua, será necesario añadir al marco uno o más algoritmos de simulación.

- **Desarrollo de algoritmos generales**

  También sería deseable que los sistemas P definidos *por ingredientes* se puedan simular. En este sentido, es necesario buscar elementos semánticos en común y diseñar algoritmos de simulación más generales que funcionen para un amplio espectro de sistemas P, identificando que *ingredientes* son soportados por cada algoritmo.

- **Desarrollar heurísticas y estrategias para ganar eficiencia**

  Con la finalidad de obtener mejor eficiencia en términos de tiempo y espacio, sería conveniente estudiar estrategias basadas en heurísticas que permitan seleccionar una de las computaciones más cortas que seguirá el simulador, minimizando el tiempo total o la memoria consumida.

Diseño de simuladores usando *High Performance Computing*

La computación de alto rendimiento *High performance Computing* (o HPC) es una herramienta muy importante para la generación de simulaciones computacionales de problemas complejos. Para lograr este objetivo, la computación de alto rendimiento se apoya en tecnologías como los clusters, supercomputadores o mediante el uso de hardware paralelo (GPUs, FPGAs). Los sistemas P son dispositivos masivamente paralelos y, por tanto, se propone investigar el campo de la computación de alto rendimiento para el desarrollo de simuladores eficientes.

En concreto, resulta prometedor el paradigma GPGPU (General Purpose Graphics Proccesor Unit). Recientemente las tarjetas de vídeo de los ordenadores (*Graphics Proccesor Units* o GPUs) han desarrollado un alto nivel de paralelismo, importantes compañías como Nvidia han producido lenguajes

de programación y tarjetas especialmente diseñadas para servir de apoyo a la investigación científica. La ventaja de usar GPUs frente a otras tecnologías de HPC es que, por un bajo coste, se obtiene un alto nivel de paralelismo (por ejemplo, 240 núcleos de computación paralela en la tarjeta Nvidia Tesla C1060).

Desarrollo de un protocolo de comunicación para simuladores

El marco de P-Lingua permite la traducción de ficheros de texto en formato P-Lingua a otros ficheros que sirvan de entrada a simuladores ajenos a la biblioteca pLinguaCore.

Sería muy interesante automatizar el proceso de envío de ficheros que definen sistemas P a simuladores locales o remotos. Para ello, habría que establecer algún protocolo que permita dejar a un simulador escuchando por un canal de comunicaciones y enviarle los trabajos para recibir después las computaciones simuladas. Este protocolo se podría estandarizar e incluir en el marco de P-Lingua y, de esta manera, sería posible enviar la definición de un sistema P a un simulador remoto de manera automática (mediante la correspondiente traducción al formato que acepta el simulador) así como recibir los resultados del mismo. Con esto se podría ganar la eficiencia de simuladores ejecutados sobre arquitecturas de alto rendimiento sin renunciar a la sencillez de definir los sistemas P en P-Lingua, ni tener que cargar de manera manual el fichero que acepta el simulador remoto.

Por añadidura, este protocolo podría estar controlado por un servidor que contenga un listado de simuladores remotos y que se encargue de distribuir la carga de trabajo de manera más equilibrada. Además, sería posible establecer prioridades entre los diferentes trabajos de simulación y que para cada simulador existiera una cola de trabajos.

El protocolo se podría extender comunicando de manera jerárquica los servidores, con el fin de poder establecer diferentes subredes de simulación.

Mejoras en las herramientas asociadas a P-Lingua

La biblioteca pLinguaCore está desarrollada en Java y permite el análisis de ficheros de texto que definen sistemas P, la simulación de computaciones y la exportación a otros formatos. Así mismo, existen aplicaciones para la línea de comandos (usando la biblioteca) que permiten la compilación y la simulación de ficheros que definen sistemas P.

Se propone:

- **"Portar" a C/C++ parte del código de pLinguaCore**

  Es posible incluir código C/C++ dentro de un proyecto Java para acelerar la ejecución de las partes de código que requieran mayor velocidad de cómputo. Para mejorar la eficiencia en términos de tiempo de los simuladores desarrollados dentro de la biblioteca, se propone exportar el código de algunos simuladores a C/C++, e implementar el protocolo adecuado para comunicar estos nuevos simuladores con la biblioteca pLinguaCore.

- **Mejora de los métodos de análisis de ficheros**

  De igual manera, existe una gran variedad de posibles mejoras de eficiencia del analizador léxico/sintáctico incorporado en pLinguaCore. Se propone el estudio de las posibles mejoras tales como el uso de nuevas herramientas de generación de analizadores, por ejemplo AntLR.

- **Creación de un *interprete de órdenes* para P-Lingua**

  La funcionalidad ofrecida por pLinguaCore puede ser accedida actualmente a través de la línea de comandos, cargando un fichero P-Lingua y ejecutando simulaciones. Se propone la creación de un interprete de órdenes que permita al usuario la definición de un sistema P, ya sea cargando ficheros P-Lingua o escribiendo directamente las instrucciones, y la simulación paso a paso de manera interactiva. Esta nueva herramienta facilitará a los investigadores la creación de nuevos sistemas P y el estudio de las propiedades de diversos modelos.

Mejoras en la modelización y simulación de ecosistemas

Aplicar el marco de modelización a otros casos

Los sistemas P multientorno funcional-probabilísticos con membranas activas parecen ser un marco adecuado para la modelización de ecosistemas, debido a las estrategias probabilísticas, así como a la modularidad y al carácter discreto de este sistema computacional.

Las herramientas software desarrolladas en la memoria han servido para facilitar la validación experimental y para producir aplicaciones orientadas a la experimentación virtual, con el objetivo final de servir de asistente informático al experto en el planteamiento y filtrado de hipótesis plausibles.

Como propuesta de línea futura de investigación, se propone aplicar el marco de modelización junto con las herramientas desarrolladas a otros casos de estudio, no solamente ecosistemas, sino cualquier proceso dinámico y complejo de la vida real en donde la estocasticidad y la interacción de manera dinámica entre un gran número de factores sean ingredientes fundamentales.

En este sentido, existen estudios que relacionan algunos procesos económicos con el modelo presa–depredador de Lotka–Volterra. Una interesante línea futura sería aplicar el marco de modelización basado en sistemas P probabilísticos y sus herramientas de software a la modelización y simulación computacional de ciertos procesos económicos.

Mejoras en la herramienta *EcoSim 2.0*

EcoSim 2.0 es una familia de software basado en P-Lingua para la validación experimental y experimentación virtual de los modelos definidos. Existe una versión específica para cada ecosistema modelizado, de tal manera que cada una de estas versiones mantiene una interfaz gráfica de usuario (GUI) desarrollada *ad hoc*, siendo el elemento común la biblioteca *pLinguaCore* como motor de simulación.

El usuario final de la aplicación es el *usuario ecólogo* y la aplicación se

comporta para él como una *caja negra*, realizando las simulaciones de manera transparente. Este usuario tan sólo interactúa con la interfaz de la aplicación.

El desarrollo de una nueva versión de EcoSim 2.0 para otro modelo de ecosistema conlleva lanzar un proyecto informático para la creación del nuevo GUI, siguiendo todas las fases del ciclo de vida del software (especificación funcional, análisis, diseño, codificación, pruebas, puesta en servicio, etc.). Este proceso contrasta con la enorme agilidad que ofrece P-Lingua para definir el sistema P del modelo. Por otra parte, el proceso actual conduce a la producción de muchas versiones de una familia de software, incrementando el coste de mantenimiento.

Con el fin de agilizar el desarrollo de nuevas aplicaciones, se propone desarrollar una nueva herramienta (*EcoSim 3.0*) que permita al *usuario diseñador*, de manera relativamente sencilla, la creación de la interfaz de usuario. Para ello, se propone establecer un método de definición de interfaces de usuario basado en ficheros Excel o XML. Así, el *usuario diseñador* no solamente se encargaría de utilizar *EcoSim 3.0* para especificar, depurar y validar experimentalmente el modelo, sino también para el diseño y la creación de una interfaz gráfica de usuario para la aplicación final, que se entregará al *usuario ecólogo*.

A este nivel de abstracción y teniendo en cuenta que el marco de modelización se puede extender a otros procesos de la vida real, conviene resaltar que *EcoSim 3.0* podría servir como herramienta en otras ramas de la ciencia distintas a la Ecología. Si esto es así, se recomienda cambiar el nombre de *EcoSim 3.0* por otro más genérico y cambiar el término de *usuario ecólogo* por *usuario experto*.

También sería conveniente incluir en esta nueva aplicación la integración con otros simuladores eficientes a través del protocolo de comunicaciones descrito anteriormente. De esta manera, la interfaz podría comunicarse con simuladores remotos y ejecutar simulaciones de sistemas P, con el fin de interpretar posteriormente los resultados y mostrarlos de manera comprensible al *usuario experto*.

Bibliografía

[1] G. Acampora, V. Loia. A Proposal of Multi-Agent Simulation System for Membrane Computing devices. *IEEE Congress on Evolutionary Computation (CEC 2007)*, 2007, 4100–4107.

[2] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, **266** (1994), 1021–1024.

[3] I.I. Ardelean, M. Cavaliere. Modelling biological processes by using a probabilistic P system software. *Natural Computing*, **2**, 2 (2003), 173–197.

[4] A. Arkin, J. Ross, H.H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected Escherichia coli cells. *Genetics*, **149**, 4 (1998), 1633–1648.

[5] F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F. de Mingo. A Recursive Algorithm for Describing Evolution in Transition P Systems. *Pre–Proceedings of Workshop on Membrane Computing*, 2001, 19–30.

[6] F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F de Mingo. Structures and Bio-Language to Simulate Transition P Systems on Digital Computers. *Lecture Notes in Computer Science*, 2235 (2001), 1–16.

[7] F. Arroyo, C. Luengo, A.V. Baranda, L.F.de Mingo. A Software Simulation of Transition P systems in Haskell. *Lecture Notes in Computer Science*, 2597 (2003), 19–32.

[8] D. Balbontín-Noval, M.J. Pérez-Jiménez, F. Sancho-Caparrini. A MzScheme Implementation of Transition P Systems. *Lecture Notes in Computer Science*, 2597 (2003), 58–73.

[9] A.V. Baranda, F. Arroyo, J. Castellanos, R. Gonzalo. Towards an Electronic Implementation of Membrane Computing: A Formal Description of Nondeterministic Evolution in Transition P Systems. *Lecture Notes in Computer Science*, 2340 (2002), 350–359.

[10] A. Baranda, J. Castellanos, R. Gonzalo, F. Arroyo, L.F. de Mingo. Data Structures for Implementing Transition P Systems in Silico. *Romanian Journal of Information Science and Technology*, **4**, 1–2 (2001), 21–32.

[11] D. Besozzi, G. Ciobanu. A P systems description of the Sodium-Potassium pump. *Lecture Notes in Computer Science*, 3365 (2005), 210–223.

[12] U.S. Bhalla, R. Iyengar. Emergent properties of networks of biological signaling pathways. *Science*, **283** (1999), 381–387.

[13] L. Bianco, F. Fontana, G. Franco, V. Manca: P systems for biological dynamics. *Applications of Membrane Computing*, 2006, pp. 83–128.

[14] R. Blossey, L. Cardelli, A. Phillips. A compositional approach to the stochastic dynamics of gene networks. *Lecture Notes in Computer Science*, 3939 (2006), 99–122.

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

[16] D. Boneh, C. Dunworth, R.J. Lipton. Breaking DES using a molecular computing. *Proceedings of DIMACS workshop on DNA computing*, 1995, 37–66.

[17] R. Borrego-Ropero, D. Díaz-Pernil, M.J. Pérez-Jiménez. Tissue simulator: A graphical tool for tissue P systems. *Proceedings of the International Workshop Automata for Cellular and Molecular Computing*, 2007, 23–34.

[18] C.J. Brown. Population dynamics of the Bearded Vulture Gypaetus barbatus in southern Africa. *African Journal of Ecology*, **35** (1997), 53–63.

[19] M. Cardona, M. A. Colomer, M.J. Pérez–Jiménez, D. Sanuy, A. Margalida. Modelling ecosystems using P systems: The Bearded Vulture, a case of study. *Lecture Notes in Computer Science*, 5391 (2009), 137–156.

[20] M. Cardona, M.A. Colomer, A. Margalida, I. Pérez–Hurtado, M.J. Pérez–Jiménez, D. Sanuy. A P system based model of an ecosystem of some scavenger birds. *Lecture Notes in Computer Science*, 5957 (2010), 182–195.

[21] M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez–Hurtado, M.J. Pérez–Jiménez, D. Sanuy. A Computational Modeling for Ecosystems Based on P Systems. *Natural Computing*, en prensa.

[22] A. Castellini, V. Manca. MetaPlab: A computational framework for metabolic P systems. *Lecture Notes in Computer Science*, 5391 (2009), 157–168.

[23] M. Cavaliere: Evolution-Communication P Systems. *Lecture Notes in Computer Science*, 2597 (2003), 134–145.

[24] M. Cavaliere, I.I. Ardelean. Modelling Respiration in Bacteria and Respiration/Photosynthesis Interaction in Cyanobacteria by Using a P System Simulator. *Applications of Membrane Computing*, 2006, pp. 129–158.

[25] J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulation of P systems with active membranes on CUDA. *Briefings in Bioinformatics*, versión online (`http://dx.doi.org/10.1093/bib/bbp064`).

[26] J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulating

a P system based efficient solution to SAT by using GPUs. *Journal of Logic and Algebraic Programming*, versión online (`http://dx.doi.org/10.1016/j.jlap.2010.03.008`).

[27] J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Implementing P systems parallelism by means of GPUs. *Lecture Notes in Computer Science*, 5957 (2010), 227-241.

[28] S. Cheruku, A. Păun, A, F.J. Romero-Campero, M.J. Pérez-Jiménez, O.H. Ibarra. Simulating FAS-induced apoptosis by using P systems. *Progress in Natural Science*, **17**, 4 (2007), 424–431.

[29] G. Ciobanu, D. Paraschiv: P System Software Simulator. *Fundamenta Informaticae*, **49**, 1–3 (2002), 61–66.

[30] G. Ciobanu, G. Wenyuan. P Systems Running on a Cluster of Computers. *Lecture Notes in Computer Science*, 2933 (2004), 123–139.

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

[32] A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. Cellular Solutions of Some Numerical NP-Complete Problems: A Prolog Implementation. *Molecular Computational Models: Unconventional Approaches*, 2005, 115–149.

[33] A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F. Sancho-Caparrini. A Prolog Simulator for Deterministic P Systems with Active Membranes. *New Generation Computing*, **22**, 4 (2004), 349–364.

[34] A. Cordón-Franco, M.A. Gutiérrez Naranjo, M.J. Pérez Jiménez, A. Riscos Núñez, F. Sancho-Caparrini. Implementing in Prolog an effective cellular solution to the knapsack problem. *Lecture Notes in Computer Science*, 2933 (2004), 140–152.

[35] D.W. Corne, P. Frisco. Dynamics of HIV infection studied with Cellular Automata and conformon-P systems. *BioSystems*, **91**, 3 (2008), 531–544.

[36] D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. A uniform family of tissue P system with cell division solving 3-COL in a linear time. *Theoretical Computer Science*, **404**, 1–2 (2008), 76–87.

[37] L. Dib, Z. Guessoum, N. Bonnet, M.T. Laskri. Multi-agent system simulating tumoral cells migration. *Lecture Notes in Artificial Intelligence*, 3809 (2005), 624–632.

[38] J.A. Donazar. *Los buitres ibéricos: biología y conservación* (J.M. Reyero ed.), Madrid, Spain, 1993.

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

[40] F. Fontana, L. Bianco, V. Manca. P systems and the modelling of biochemical oscillations. *Lecture Notes in Computer Science*, 3850 (2005), 199–208.

[41] P. Frisco, R.T. Gibson. A simulator and an evolution program for conformon-P systems. *TAPS, Workshop on Theory and Applications of P Systems*, 2005, 427–430.

[42] P. Frisco, S. Ji. Conformons-P systems. *Lecture Notes in Computer Science*, 2568 (2003), 291–301.

[43] M.R. Garey, D.S. Johnson. *Computers and intractability*, W.H. Freeman and Company, New York, (1979).

[44] A. Georgiou, M. Gheorghe: Generative Devices Used in Graphics. *Preproceedings of the Workshop on Membrane Computing*, 2003, 266–272.

[45] A. Georgiou, M. Gheorghe, F. Bernardini: Membrane–Based Devices Used in Computer Graphics. *Applications of Membrane Computing*, 2006, pp. 253–282.

[46] R. Gershoni, E. Keinan, Gh. Păun, R. Puran, T. Ratner, S. Shoshani. Research topics arising from the (Planned) P systems implementation experiment in technion. *Proceedings of the 6th Brainstorming Week on Membrane Computing*, 2008, 183–192.

[47] M. Gheorghe, C. Martín-Vide, V. Mitrana, M.J. Pérez–Jiménez. An agent based approach of collective foraging. *Lecture Notes in Computer Science*, 2686 (2003), 638–645.

[48] M.A. Gibson, J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry*, **104**, 25 (2000), 1876–1889.

[49] D.T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, **22** (1976), 403–434.

[50] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, **81** (1977), 2340–2361.

[51] D.T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, **188** (1992), 404–425.

[52] D.T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics*, **115** (2001), 1716–1733.

[53] D.T. Gillespie, L. Petzold. Improved leap-size selection for accelerated stochastic simulation. *Journal of Chemical Physics*, **119** (2003), 8229–8234.

[54] P.J.E. Goss, J. Peccoud. Quantitative modelling of stochastic systems in molecular biology by using stochastic Petri nets. *Proceedings of the*

National Academy of Sciences of the United States of America, **95** (1998), 6750–6755.

[55] M.A. Gutiérrez–Naranjo, D. Ramírez-Martínez: A Software Tool for Dealing with Spiking Neural P Systems. *Fifth Brainstorming Week on Membrane Computing*, 2007, 299–313.

[56] M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, D. Ramírez-Martínez. A software tool for verification of spiking neural P systems. *Natural Computing*, **7**, 4 (2008), 485–497.

[57] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. A Fast P System for Finding Balanced 2-partition. *Soft Computing*, **9**, 9 (2005), 673–678.

[58] M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez. Towards a programming language in cellular computing. *Electronic Notes in Theoretical Computer Science*, **123** (2005), 93–110.

[59] J. Hartmanis, P.M. Lewis, R.E. Stearn. Hierarchies of memory limited computations. *Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logical Design*, 1965, 179–190.

[60] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, **49** (1987), 737–759.

[61] T. Head. Aqueous simulations of membrane computations. *Romanian Journal of Information Science and Technology*, **4** (2001), 1–2.

[62] M. Holcombe, M. Gheorghe, N. Talbot. A hybrid machine model of rice blast fungus, Magnaphorte Grisea. *BioSystems*, **68**, 2–3 (2003), 223–228.

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

[64] R. Impagliazzo. A personal view of average-case complexity. En *10th IEEE Annual Conference on Structure in Complexity Theory*, 1995, 143–147.

[65] D. Jackson, M. Holcombe, F. Ratnieks. Trail geometry gives polarity to ant foraiging networks. *Nature*, **432** (2004), 907–909.

[66] D. Jackson, M. Holcombe, F. Ratnieks. Coupled computational simulation and empirical research into the foragin system of Pharaoh's ant. *BioSystems*, **76** (2004), 101–114.

[67] H.A. Jenner, J.W. Whitehouse, C.J.L. Taylos, M. Khalanski. Cooling water management in European power stations. Biology ans Control of fouling. *Hidroecol. Appl*, **10** (1998), 1–2.

[68] R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, Plenum Press, (1972), 85–104.

[69] L. Levin. Average case complete problems. *SIAM Journal Computing*, **15** (1986), 285–286.

[70] P.M. Lewis, R.E. Stearn, J. Hartmanis. 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.

[71] J.B. Lingrel, T. Kuntzweiler. Na^+, K^+–ATPase. *Annual Review of Biochemistry*, **269**, 31 (1994), 19659–19662.

[72] M. Maliţa. Membrane Computing in Prolog. *Pre-proceedings of the Workshop on Multiset Processing*, 2000, 159–175.

[73] A. Margalida, D. García, A. Cortés–Avizanda. Factors influencing the breeding density of Bearded Vultures, Egyptian Vultures and Eurasian Griffon Vultures in Catalonia (NE Spain): management implications. *Animal Biodiversity and Conservation*, **30** (2007), 189–200.

[74] A. Margalida, S. Mañosa, J. Bertran, D. García. Biases in studying the diet of the Bearded Vulture. *The Journal of Wildlife Management*, **71** (2006), 1621–1625.

[75] A. Margalida, J. Bertran, J. Boudet. Assesing the diet of nestling Bearded Vultures: a comparison between direct observation methosds. *Journal of Field Ornithology*, **76** (2005), 40–45.

[76] A. Margalida, J.Bertran, R. Heredia. Diet and food preferences of the endangered Bearded Vulture Gypaetus barbatus: a basis for their conservation. *Ibis*, **151** (2009), 235–243.

[77] M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. A P-Lingua based simulator for Tissue P systems. *Journal of Logic and Algebraic Programming*, versión online (`http://dx.doi.org/10.1016/j.jlap.2010.03.009`).

[78] W.S. McCulloch, W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5** (1943), 115–133.

[79] T.C. Meng, S. Somani, P. Dhar. Modelling and simulation of biological systems with stochasticity. *In Silico Biology*, **4** (2004), 293–309.

[80] R. Milner. Communication and mobile systems: The π-calculus. *Cambridge University Press.*, 1999.

[81] M. Minsky, S. Papert. *Perceptions*, Cambridge, M.A. MIT Press, 1970.

[82] I.A. Nepomuceno-Chamorro: A Java simulator for basic transition P systems. *Proceedings of the Second Brainstorming Week on Membrane Computing*, 2004, 309–315.

[83] V. Nguyen, D. Kearney, G.Gioiosa. A Region-Oriented Hardware Implementation for Membrane Computing Applications and Its Integration into Reconfig-P. *Lecture Notes in Computer Science*, 5957 (2010), 385-409.

[84] T.Y. Nishida. Membrane Algorithms. *Lecture Notes in Computer Science*, 3850 (2006), 55–66.

[85] Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* Nr. 208, 1998.

[86] Gh. Păun. P systems with active membranes: Attacking **NP**–complete problems. *Journal of Automata, Languages and Combinatorics*, **6**, 1 (2001), 75–90.

[87] Gh. Păun. Further research topics about P systems. *Pre-Proceedings of Workshop on Membrane Computing*, 2001, 243–250.

[88] G. Păun. *Membrane Computing: An Introduction.* Springer-Verlag, 2002.

[89] M.J. Pérez-Jiménez, F.J. Romero-Campero. A CLIPS simulator for recognizer P systems with active membranes. *Proceedings of the Second Brainstorming Week on Membrane Computing*, 2004, 387–413.

[90] M.J. Pérez-Jiménez, F.J. Romero-Campero. An Efficient Family of P Systems for Packing Items into Bins. *Journal of Universal Computer Science*, **10**, 5, 2004, 650–670.

[91] M.J. Pérez-Jiménez, F.J. Romero-Campero. Attacking the Common Algorithmic Problem by recognizer P systems. *Lecture Notes in Computer Science*, 3354 (2005), 304–315.

[92] M.J. Pérez-Jiménez, F.J. Romero-Campero. A study of the robustness of the EGFR signalling cascade using continuous membrane systems. *Lecture Notes in Computer Science*, 3561 (2005), 268–278.

[93] M.J. Pérez-Jiménez, F.J. Romero-Campero. P systems, a new computational modelling tool for systems biology, *Transactions on Computational Systems Biology VI, LNBI*, **4220** (2006), 176–197.

[94] M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini. A polynomial complexity class in P systems using membrane division. *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, 2003, 284–294.

[95] D. Pescini,D. Besozzi, G. Mauri, C. Zandron. Dynamical probabilistic P systems, *International Journal of Foundations of Computer Science*, **17**, 1 (2006), 183–195.

[96] B. Petreska, C. Teuscher: A Reconfigurable Hardware Membrane System. *Lecture Notes in Computer Science*, 2933 (2004), 269–285.

[97] A. Phillips, L. Cardelli. A correct abstract machine for the stochastic Pi-calculus. *Electronical Notes in Theoretical Computer Science*, (2004).

[98] M. Pogson, R. Smallwood, R., E. Qwarnstrom, M. Holcombe. Formal agent-based of intracellular chemical interactions. *BioSystems*, **85**, 1 (2006), 37–45.

[99] C. Priami, A. Regev, E. Shapiro, W. Silverman. Application of a stochastic Name-Passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, **80** (2001), 25–31.

[100] V.N. Reddy, M.N. Liebman, M.L. Mavrouniotis. Qualitative analysis of biochemical reaction systems. *Computers in Biology & Medicine*, **26**, 1 (1996), 9–24.

[101] A. Regev, E. Shapiro. The π-calculus as an abstraction for biomolecular systems. *Modelling in Molecular Biology*,2004, pp. 219–266.

[102] A. Regev, E. Shapiro. Cellular abstractions: Cells as computation, *Nature*, **419** (2002), 343.

[103] A. Riscos-Núñez: *Cellular programming: efficient resolution of numerical NP-complete problems*. Ph.D. Thesis. University of Seville, 2004.

[104] E. Rivero-Gil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez. Graphics and P Systems: Experiments with JPLANT. *Proceedings of the Sixth Brainstorming Week on Membrane Computing*, 2008, 241–254.

[105] F.J. Romero-Campero, M.J. Pérez-Jiménez. Modelling gene expression control using P systems: The Lac Operon, a case study. *BioSystems*, **91**, 3 (2008), 438–457.

[106] F.J. Romero–Campero, M.J. Pérez–Jiménez. A model of the quorum sensing system in Vibrio Fischeri using P systems. *Artificial Life*, **14**, 1 (2008), 95–109.

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

[108] C. Ruíz Altaba, P.J. Jiménez, M.A. López. El temido mejillón cebra empieza a invadir los ríos españoles desde el curso bajo del río Ebro. *Quercus*, 188 (2001), 50–51.

[109] S. Sedwards, T. Mazza: Cyto–Sym: A Formal Language Model and Stochastic Simulator of Membrane-Enclosed Biochemical Processes. *Bioinformatics*, **23**, 20 (2007), 2800–2802.

[110] C. Sunyer. El periodo de emancipación en el quebrantahuesos: consideraciones sobre su conservación. El quebrantahuesos (Gypaetus barbatus) en los Pirineos, Colección Técnica, (1991), pp. 47–65.

[111] Y. Suzuki, H. Tanaka. On a LISP implementation of a class of P systems. *Romanian Journal of Information Science and Technology*, 3, 2 (2000), 173–186.

[112] Y. Suzuki, Y. Fujiwara, H. Tanaka, J. Takabayashi. Artificial life applications of a class of P systems: Abstract rewriting systems on multisets. *Lecture Notes in Computer Science*, 2235 (2001), 299–346.

[113] Y. Suzuki, J. Takabayashi, H. Tanaka. Adaptive behavior in a tritrophic interactions consisting of plants, herbivors and carnivores. *Sixth International Conference on the Simulation of Adaptive Behavior (SAB2000)*, 2000, 11–15.

[114] A. Syropoulos, E.G. Mamatas, P.C. Allilomes, K.T. Sotiriades. A Distributed Simulation of Transition P Systems. *Lecture Notes in Computer Science*, 2933 (2004), 357–368.

[115] R. Torres. *Conservación de recursos genéticos ovinos en la raza Xisqueta: caracterización estructural, racial y gestión de la diversidad en programas "in situ".* Tesis Doctoral. Universitat Autónoma de Barcelona, Barcelona (2006).

[116] A.M. Turing. On computable numbers, with an application to the Enstcheidungsproblem. *Proceedings of the London Mathematical Society*, **42** (1936), 230–265; correcciones ibid. **43** (1936), 544–546.

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

[118] A. Palau, I. Cía, D. Fargas, M. Bardina, S. Massuti. Resultados preliminares sobre ecología básica y distribución del mejillón cebra en el embalse de Riba–Roja (río Ebro). *UPH Ebro-Pirineos (ENDESA Generación), Dirección de Medio Ambiente y Desarrollo Sostenible (ENDESA Servicios)* (2003).

[119] D.C. Walker, J. Southgate, G. Hill, M. Holcombe, D.R. Hose, S.M. Wood, S. MacNeil, R.H. Smallwood. The epitheliome: modelling the social behaviour of cells. *BioSystems*, **76**, 1–3 (2004), 89–100.

[120] Christof Teuscher's P systems web page
`http://www.teuscher.ch/psystems`

[121] Models of Natural Computing – University of Verona
`http://mnc.sci.univr.it`

[122] Natural Computing Group of the Technical University of Madrid
http://www.lpsi.eui.upm.es/nncg

[123] Research Group on Natural Computing – University of Seville
http://www.gcn.us.es

[124] GNU Scientific Library http://www.gnu.org/software/gsl

[125] GNU GPL License http://www.gnu.org/licenses/gpl.html

[126] P systems web page http://ppage.psystems.eu

[127] P System Modelling Framework at the University of Sheffield
http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/PSystemMF.htm

[128] The Java web page http://www.java.com/

[129] The Colt library website http://acs.lbl.gov/ hoschek/colt/

[130] The JDom library website http://www.jdom.org/

[131] The JFreeChart library website http://www.jfree.org/jfreechart/

[132] The Java Swing classes
http://java.sun.com/javase/6/docs/technotes/guides/swing/

[133] The Message Passing Interface (MPI) standard
http://www-unix.mcs.anl.gov/mpi

[134] The pLinguaCore library website
http://www.p-lingua.org/wiki/index.php/PLinguaCore

[135] The Stochastic Pi-Machine http://www.doc.ic.ac.uk/~anp/spim/