UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño e implementación de un intérprete para un lenguaje de especificación de sistemas P

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Cantero Delgado, Jesús

Tutor/a: Sempere Luna, José María

CURSO ACADÉMICO: 2022/2023

Resumen

En este trabajo se ha desarrollado un intérprete de sistemas P utilizando Python, con el objetivo de simular el comportamiento de las membranas y sus interacciones. El enfoque se basa en los principios biológicos de las células y sus membranas. Se han investigado los fundamentos teóricos, diseñado la sintaxis del sistema P, desarrollado el intérprete, realizado pruebas exhaustivas y evaluado su capacidad para resolver problemas. Este trabajo contribuye al avance de la computación con membranas y proporciona habilidades técnicas en este campo.

Palabras clave: intérprete de sistemas P, computación celular, Python, sintaxis, resolución de problemas, membranas.

Abstract

In this work, an interpreter for P systems has been developed using Python, aiming to simulate the behavior of membranes and their interactions. The approach is based on the biological principles of cells and their membranes. The theoretical foundations have been investigated, the syntax of the P system has been designed, the interpreter has been developed, exhaustive testing has been conducted, and its problem-solving capabilities have been evaluated. This work contributes to the advancement of membrane computing and provides technical skills in this field.

Keywords: P system interpreter, cellular computing, Python, syntax, problem-solving, membranes.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

En 1998 Gheorge Păun (1) introdujo la computación celular con membranas como un campo de investigación prometedor dentro de la rama de computación natural (2). Este paradigma se basa en el estudio de los sistemas biológicos, particularmente en la estructura y funcionamiento de las células y sus membranas, para desarrollar nuevos modelos y algoritmos computacionales.

En los últimos años, la computación celular con membranas ha despertado un gran interés debido a su capacidad para resolver problemas complejos de manera eficiente y su potencial aplicación en diversos campos, como la optimización, la bioinformática, la inteligencia artificial y la computación cuántica. La idea fundamental detrás de esta disciplina es aprovechar las características y propiedades de las membranas biológicas, que actúan como estructuras de control y procesamiento en las células, para diseñar sistemas computacionales innovadores y más eficientes. Uno de los modelos más prominentes en este ámbito es el sistema P, que ha demostrado ser altamente efectivo en la resolución de problemas complejos.

Un sistema P es una abstracción computacional que se basa en las características y procesos que ocurren en las células biológicas. Se utiliza para representar sistemas distribuidos y paralelos en los que varias membranas interactúan entre sí mediante reglas de comunicación y transporte de sustancias. Estas membranas actúan como estructuras de control y procesamiento, similar a cómo las células biológicas utilizan sus membranas para realizar funciones vitales.

En un sistema P, las membranas se organizan en forma jerárquica, donde cada nivel representa una segmentación de procesos. Cada membrana puede contener objetos organizados en multiconjuntos, que representan las sustancias químicas o señales que se comunican y se procesan dentro del sistema. El sistema P define un conjunto de reglas de evolución que especifican cómo las membranas se dividen, cómo se comunican y cómo se procesan los multiconjuntos.

La relación entre el sistema P y la computación celular con membranas es estrecha, ya que el sistema P es uno de los modelos fundamentales utilizados en esta disciplina. La computación con membranas se basa en la observación de cómo las células biológicas realizan procesos complejos de manera eficiente y cómo pueden utilizarse esos principios para resolver problemas computacionales.

Nuestro trabajo se basará en los sistemas P de transición que es una variante específica de los sistemas P en la cual las células y las membranas pueden realizar transiciones o transformaciones mediante reglas de evolución. Estas reglas determinan cómo los objetos contenidos en las células y las membranas pueden evolucionar y cambiar de estado a lo largo del tiempo.

En un sistema P de transición, las reglas de evolución pueden especificar diferentes tipos de transformaciones, como la creación o destrucción de objetos, la transferencia de objetos entre células o membranas, y la interacción entre objetos. Estas reglas se aplican de manera concurrente y no determinista, lo que permite una amplia gama de comportamientos y procesos computacionales.

A partir de ahora cuando nos refiramos a sistemas P, estos serán de “transición” aunque no lo especifiquemos explícitamente.

En este trabajo de final de grado, nos enfocaremos en el desarrollo de un intérprete capaz de procesar sistemas P utilizando la computación celular con membranas. Nuestro objetivo será diseñar un algoritmo que permita simular el comportamiento de las membranas y las interacciones entre ellas, siguiendo las reglas y restricciones del sistema.

Al aprovechar las características y los principios biológicos de las células y sus membranas, buscamos contribuir al avance de la computación celular con membranas y explorar su potencial en la resolución eficiente de problemas computacionales.

1.1 Motivación

La realización de este trabajo ha comportado varias justificaciones y motivaciones significativas, estas son:

1. Exploración de un nuevo paradigma: La computación celular con membranas y el sistema P representan un paradigma innovador que combina la biología y la informática. El desarrollo de un intérprete para procesar sistemas P permite sumergirse en este enfoque novedoso y comprender cómo los principios biológicos pueden aplicarse para resolver problemas computacionales complejos.
2. Resolución eficiente de problemas complejos: El sistema P ha demostrado ser altamente efectivo para resolver problemas complejos debido a su capacidad para modelar sistemas distribuidos y paralelos. Al desarrollar un intérprete que

pueda procesar sistemas P, se podrá aprovechar este modelo para abordar desafíos computacionales que de otra manera serían difíciles de resolver con enfoques tradicionales.

3. Potencial aplicación en diversas áreas: La computación celular con membranas, y en particular el sistema P, tiene aplicaciones potenciales en campos como la optimización, la bioinformática, la inteligencia artificial y la computación cuántica. Al desarrollar un intérprete para sistemas P, se estarán sentando las bases para futuras investigaciones y aplicaciones en estas áreas.
4. Contribución al avance de la computación con membranas: Al desarrollar un intérprete para procesar sistemas P, se contribuirá al avance de la computación con membranas al proporcionar una implementación práctica y funcional de este modelo. Esto puede ayudar a ampliar el conocimiento y la comprensión de esta disciplina, así como a fomentar nuevas investigaciones y desarrollos en el campo.
5. Desarrollo de habilidades técnicas: El desarrollo de un intérprete para sistemas P implica el dominio de conceptos teóricos y habilidades prácticas relacionadas con la computación celular con membranas, la programación y el diseño de algoritmos eficientes. Realizar este trabajo permitirá adquirir y fortalecer estas habilidades técnicas, lo cual es valioso para una carrera en ingeniería informática y campos afines.

1.2 Objetivos

A continuación, detallamos los objetivos del presente trabajo:

- Investigar y comprender en profundidad los fundamentos teóricos de la computación con membranas y el sistema P.
- Diseñar una sintaxis de un sistema P para poder ser interpretado y posteriormente procesado.
- Diseñar y desarrollar un intérprete eficiente que pueda procesar sistemas P utilizando la computación celular con membranas.
- Realizar pruebas exhaustivas del intérprete para verificar su precisión y capacidad para resolver problemas complejos.
- Evaluar la capacidad del intérprete para resolver problemas específicos mediante el procesamiento de sistemas P durante un número determinado de iteraciones.

- Realizar recomendaciones para futuras mejoras y aplicaciones del intérprete desarrollado, proponiendo posibles extensiones o adaptaciones del modelo de computación con membranas.

1.3 Estado del arte

La computación con membranas, también conocida como computación celular con membranas, ha sido objeto de amplia investigación en los últimos años. Esta disciplina combina conceptos de biología y ciencias de la computación para desarrollar modelos y algoritmos basados en las propiedades y procesos de las células biológicas. Uno de los modelos más destacados en este campo es el sistema P, propuesto por Păun en (1).

El sistema P se basa en la estructura y el comportamiento de las células biológicas y utiliza membranas como unidades de procesamiento y comunicación. A lo largo de los años, se han realizado numerosos avances en la comprensión y aplicación de los sistemas P, así como en su integración con otros enfoques computacionales.

Desde su origen se han llevado a cabo investigaciones en varias áreas relacionadas con la computación con membranas y los sistemas P. Algunos de los temas más relevantes abordados incluyen:

1. Modelado y simulación: Se han desarrollado diversos modelos y herramientas de simulación para representar y analizar sistemas P. Estos modelos incluyen extensiones del sistema P original, como sistemas P con cooperación, sistemas P probabilísticos, entre otros. Las herramientas de simulación permiten estudiar el comportamiento de los sistemas P en diferentes escenarios y analizar su eficiencia y capacidad de resolución de problemas (3).
2. Aplicaciones y resolución de problemas: Los sistemas P han demostrado ser efectivos en la resolución de diversos problemas computacionales, como la solución de ecuaciones, el procesamiento de imágenes, la optimización combinatoria, entre otros. Se han propuesto y aplicado algoritmos específicos basados en sistemas P para abordar estos problemas, logrando resultados prometedores en términos de eficiencia y rendimiento. En (4) se presentan algunas de las líneas de investigación abiertas en el área, centrándose en los problemas de segmentación, esqueletización y aspectos algebraico-topológicos de las imágenes .
3. Relación con otros modelos de computación: La computación con membranas se ha explorado en relación con otros modelos y paradigmas de computación,

como los autómatas celulares, la computación cuántica y la programación genética. Estas investigaciones buscan identificar las similitudes y diferencias entre los diferentes enfoques y explorar su complementariedad para resolver problemas complejos. Por ejemplo, en (5) se utiliza la computación con membranas para simular fenómenos descritos por la mecánica cuántica o en (6) definiendo un nuevo autómata que utiliza los conceptos de la computación con membranas y se rige por los principios de la computación cuántica.

4. Aspectos teóricos y fundamentales: Se han realizado estudios teóricos y analíticos para comprender mejor los fundamentos de la computación con membranas y los sistemas P. Estos estudios incluyen análisis de complejidad, demostraciones formales de propiedades y teoremas relacionados con los sistemas P, y exploración de las capacidades computacionales de los sistemas P en términos de computabilidad y complejidad (7).

En resumen, la computación con membranas muestra un campo de investigación dinámico y en constante evolución. Se han realizado avances significativos en áreas como modelado y simulación, aplicaciones y resolución de problemas, relación con otros modelos de computación y aspectos teóricos y fundamentales. Estos avances han permitido una mejor comprensión de los sistemas P y su potencial para resolver problemas complejos de manera eficiente.

Además de los avances mencionados en el estado del arte de la computación celular con membranas, es importante destacar la contribución de eventos y organizaciones que han promovido el desarrollo y la difusión de esta disciplina.

El Conference on Membrane Computing (CMC) es uno de los eventos más destacados en el campo de la computación con membranas. Inicialmente conocido como Workshop on Membrane Computing (WMC), este congreso internacional se ha llevado a cabo desde el año 2000. Después de su décima edición, el evento cambió su nombre a Conference on Membrane Computing (CMC), consolidándose como un espacio de encuentro y discusión para investigadores y expertos en el área.

Asimismo, existe una versión asiática de este congreso denominada Asian Conference on Membrane Computing (ACMC), que ha brindado una plataforma específica para la comunidad de investigadores en Asia interesados en la computación con membranas.

La International Membrane Computing Society (IMCS) es una organización fundada en 2016 con el objetivo de fortalecer la cooperación internacional en el campo de la computación con membranas. Esta sociedad ha promovido la colaboración y el intercambio de conocimientos entre investigadores de todo el mundo. Entre las

iniciativas de la IMCS, se destaca la publicación de la revista Journal of Membrane Computing (JMC), que se ha convertido en un proyecto importante de la sociedad. JMC proporciona un foro para la publicación de investigaciones originales y de alta calidad en todos los aspectos de la computación con membranas.

La revista JMC, respaldada por la IMCS y con el apoyo de Springer-Verlag, desempeña un papel crucial en el desarrollo y la difusión de los avances en la computación con membranas. La revista publica artículos de investigación, artículos de revisión, comunicaciones cortas y tutoriales que abarcan una amplia gama de temas, desde sistemas P de tipo celular, sistemas P de tipo tejido, sistemas P neuronales con emisión de pulsos y otros tipos de sistemas P, hasta algoritmos de membrana, complejidad computacional, investigación interdisciplinaria que combina la computación con membranas con la computación evolutiva y las redes neuronales, así como aplicaciones en optimización y modelado de biosistemas, e implementaciones de computación con membranas con nanotecnología. Esta amplia cobertura promueve el intercambio de ideas entre los estudios biológicos y tecnológicos, impulsando el avance de una comunidad interdisciplinaria interesada en la inteligencia computacional inspirada en la biología.

1.4 Estructura de la memoria

La organización del contenido del presente documento versará sobre una introducción a la computación con membranas y a los sistemas P. Comprenderemos cuales son los campos de investigación de esta área, así como la justificación de la elección de trabajar sobre este ámbito. Una vez realizada una breve introducción pasaremos a detallar el proceso de diseño y programación de nuestro de intérprete de sistemas P de transición. En primer lugar, definiremos unos conceptos previos sobre sistemas P y teoría de autómatas y lenguajes.

En segundo lugar, describiremos la sintaxis del lenguaje mediante la cual se codificarán los sistemas P que serán procesados por el intérprete.

Posteriormente desgranaremos la aplicación en sí para ahondar en detalles técnicos en el desarrollo de la aplicación.

Para finalizar, mostraremos los casos de prueba realizados para comprobar la potencia de los analizadores del intérprete y los ejemplos empleados y procesados con la correspondiente visualización de sus resultados.

2 Computación con membranas y sistemas P. Definiciones básicas

2.1 Las células y las máquinas

Para poder empezar a entrar en detalle con sistemas P necesitamos definir y entender algunos conceptos básicos tanto del área de biología referente a la célula como de teoría de autómatas y lenguajes formales.

Una célula es la unidad estructural y funcional básica de los organismos vivos. Constituye la unidad más pequeña capaz de llevar a cabo todas las funciones vitales necesarias para la supervivencia y el funcionamiento adecuado de un organismo.

Aunque existen diferentes tipos de células con características específicas, la mayoría de las células eucariotas comparten algunas partes comunes:

- Membrana celular: También conocida como membrana plasmática, es una estructura delgada y flexible que rodea la célula y la separa del entorno externo. Actúa como una barrera selectiva que controla el flujo de sustancias dentro y fuera de la célula, permitiendo el intercambio de nutrientes y desechos.
- Citoplasma: Es el fluido gelatinoso que llena el espacio entre la membrana celular y el núcleo. Contiene varias estructuras y orgánulos celulares, como el retículo endoplasmático, el aparato de Golgi, las mitocondrias y los ribosomas. El citoplasma alberga numerosas reacciones químicas esenciales para el funcionamiento celular.
- Núcleo: Es el centro de control de la célula y contiene el material genético, como el ADN, que contiene las instrucciones necesarias para la síntesis de proteínas y la reproducción celular. El núcleo está rodeado por una membrana nuclear que regula el intercambio de material entre el núcleo y el citoplasma.
- Orgánulos celulares: Son estructuras especializadas que realizan funciones específicas dentro de la célula. Algunos ejemplos comunes incluyen el retículo endoplasmático, que está involucrado en la síntesis de proteínas y lípidos, el aparato de Golgi, que procesa y clasifica las moléculas para su distribución, y las mitocondrias, que generan energía a través de la respiración celular.

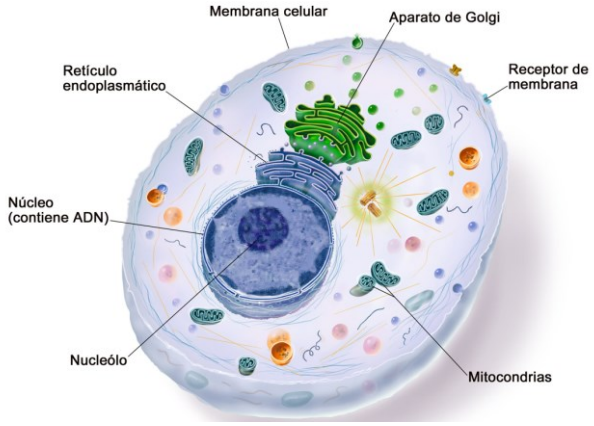


Figura 1: Partes de la célula

Estas son solo algunas de las partes principales de una célula. Cabe mencionar que hay diferencias significativas entre los diferentes tipos de células, como las células procariotas y las células eucariotas, que poseen características estructurales y funcionales distintas.

Podemos establecer una relación entre una célula y una máquina al considerar ciertas similitudes en términos de estructura y función. Si bien son entidades muy diferentes, estas son algunas formas en las que podemos establecer una analogía:

- Estructura organizada: Tanto las células como las máquinas tienen una estructura organizada que les permite llevar a cabo funciones específicas. En el caso de las células, tienen diferentes orgánulos y compartimentos que desempeñan roles especializados. De manera similar, las máquinas están compuestas por componentes y piezas que trabajan en conjunto para lograr una tarea determinada.

- Funciones específicas: Tanto las células como las máquinas tienen funciones específicas que desempeñar. Las células realizan funciones vitales para el organismo, como la síntesis de proteínas, la producción de energía y la reproducción celular. Las máquinas, por su parte, están diseñadas para realizar tareas específicas, como producir energía, transportar materiales o realizar cálculos.

- Interacciones y comunicación: Tanto en las células como en las máquinas, existe la necesidad de interacciones y comunicación entre diferentes componentes para lograr un objetivo común. En las células, esto implica la comunicación entre

orgánulos y la transferencia de señales químicas. En las máquinas, puede implicar la transmisión de información a través de cables, circuitos o redes.

- Mantenimiento y reparación: Tanto las células como las máquinas requieren mantenimiento y reparación para garantizar su correcto funcionamiento. Las células tienen mecanismos de reparación del ADN y procesos de eliminación de desechos. Las máquinas, por su parte, requieren mantenimiento regular, limpieza y reparación de piezas dañadas.

2.2 Conceptos de teoría de autómatas y lenguajes formales

En este apartado presentamos nociones básicas de lenguajes formales basándonos en (8).

Un *alfabeto* es un conjunto de símbolos finito y no vacío. Por convención, utilizamos el símbolo Σ para designar un alfabeto.

Una *cadena* es una secuencia finita y ordenada de símbolos pertenecientes a un alfabeto. Por ejemplo, en el alfabeto de los números binarios = $\{0,1\}$, la cadena $\mathrm{x} = 01100$ estaría formada por los símbolos de dicho alfabeto.

La *cadena vacía* es aquella representada por la aparición de ningún símbolo del alfabeto. Se suele representar por λ y puede construirse sobre cualquier alfabeto.

En determinadas ocasiones resulta interesante poder clasificar las cadenas según el número de posiciones que ocupan todos los símbolos de la cadena. Para una cadena x, esta clasificación se denomina *longitud de una cadena* y se denota por $|x|$. Por ejemplo, dado el alfabeto $\Sigma = \{0,1\}$, para la cadena $x = 01100, |x| = 5$ *y* $|\lambda| = 0$.

Sea Σ un alfabeto, el conjunto de todas las cadenas de una determinada longitud sobre Σ se puede expresar mediante una notación exponencial. Se define Σ^n como el conjunto de cadenas de longitud n, donde cada uno de los símbolos pertenece a Σ. Sea el alfabeto $\Sigma = \{0,1\}$, podemos obtener $\Sigma^1 = \{0,1\}, \Sigma^2 = \{00,01,10,11\}$, etc.

Se establece un convenio para designar al de todas las cadenas sobre un determinado alfabeto Σ, como Σ^*, denominado cierre de Kleene. Conviene advertir, que en Σ^* también está incluida la cadena vacía. Asimismo, se define Σ^+como el conjunto formado por Σ^* excluyendo la cadena vacía, $\Sigma^+ = \Sigma^* - \{\lambda\}$.

Sean u, v dos cadenas cualesquiera sobre un determinado alfabeto Σ, la *concatenación* denotada por uv es una copia de los símbolos de pertenecientes a u seguidos de los símbolos pertenecientes a v.

Sea Σ un alfabeto, un lenguaje L es un conjunto de cadenas seleccionadas sobre dicho alfabeto, i.e., tanto $L \subseteq \Sigma^*$.

Dado el alfabeto $\Sigma = \{a_1, \dots, a_n\}$ y la cadena $w \in \Sigma^*$, denotaremos por $|x|_{aj}$ al número de símbolos a_j que contiene la cadena w.

Se define la proyección de Parikh como una correspondencia $\Psi = (|w|_{a1}, |w|_{a2}, \dots, |w|_{an})$. Del mismo modo se puede extender a lenguajes como $\Psi(L) = \{\Psi(w): w \in L\}$.

2.3 Conceptos de sistemas P

A continuación, seguimos con definiciones y conceptos básicos sobre estructuras de membranas y sistemas P.

Tal y como hemos anticipado en apartados anteriores fue Păun en [1] quien introdujo la computación celular con membranas inspirado tanto en la estructura como el funcionamiento de las células.

Las partes que componen los modelos computacionales sobre los que se apoya esta área son la estructura de membranas, que representa las vesículas que componen la célula. Esta tiene una membrana superior que es la piel y que separa al resto de membranas del entorno. A su vez, cada membrana puede contener un conjunto de objetos, los cuales pueden evolucionar en base a unas reglas que actúan sobre cada membrana. Por lo que respecta a la relación de los componentes que acabamos de presentar del modelo computacional con la estructura celular, los objetos representan las sustancias químicas y las reglas representan las reacciones químicas que se producen dentro de las membranas. Si varias reglas compiten entre sí para poder ser aplicadas sobre un objeto se elegirá solo una de ellas de forma aleatoria, la cual, producirá nuevos objetos comunicándolos con membranas adyacentes si fuera el caso. Todas las reglas que se pueden ejecutar sobre los objetos y que no se interpongan entre sí en un determinado instante, se lanzarán forma simultánea y a su vez, cada una de las membranas harán sus ejecuciones de reglas simultáneamente.

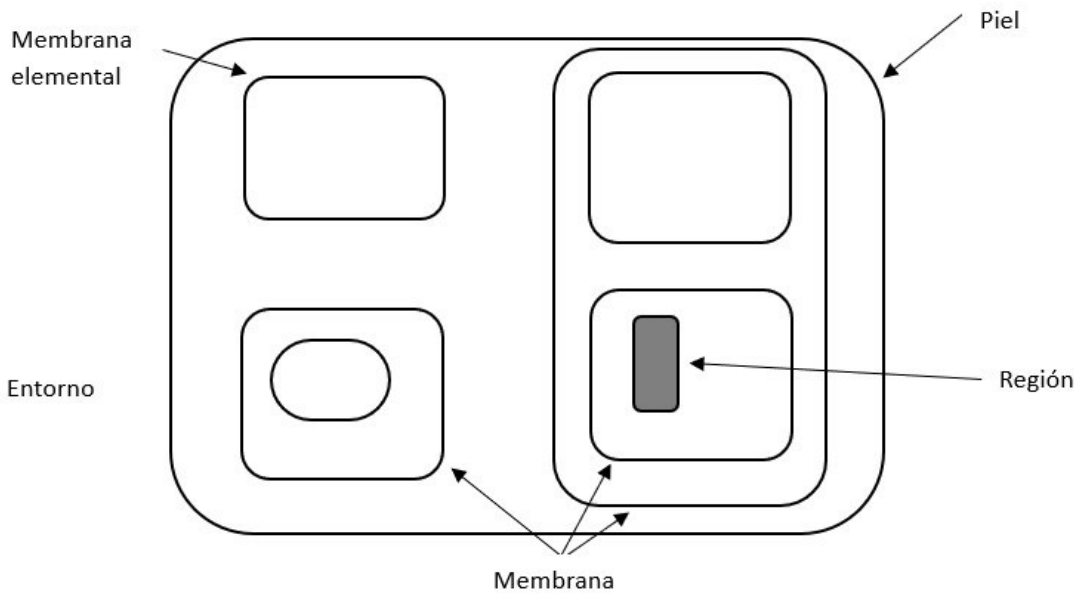


Figura 2: Estructura de membranas

A continuación, necesitamos introducir un concepto imprescindible para poder definir los sistemas P sobre los que vamos a realizar nuestro estudio [1]. Sea U un conjunto cualquiera, un *multiconjunto* sobre U es una aplicación $M: U \rightarrow N$, donde N es el conjunto de los números naturales. Para cada $a \in U$, la multiplicidad de a se denota como $M(a)$.

Mientras que en un conjunto tradicional cada elemento aparece como máximo una vez, en un multiconjunto, los elementos pueden aparecer múltiples veces. La multiplicidad indica cuántas veces aparece un elemento en el multiconjunto. Por ejemplo, en un multiconjunto $\{a, a, b, c, c, c\}$, el elemento *'a'* tiene una multiplicidad de 2, el elemento *'b'* tiene una multiplicidad de 1 y el elemento *'c'* tiene una multiplicidad de 3.

A diferencia de los conjuntos tradicionales, en los multiconjuntos el orden de los elementos no es relevante. Dos multiconjuntos son iguales si contienen los mismos elementos con las mismas multiplicidades, independientemente del orden en que se presenten.

En el apartado anterior vimos cómo una cadena era una secuencia finita de símbolos sobre cierto alfabeto. Por tanto, cualquier cadena $x \in V^*$ define un multiconjunto sobre los símbolos de V que se puede denotar como $m(x) = \{(a, |x|_a \mid a \in V\}$. Por ejemplo, la cadena $x = abbcc$ define el multiconjunto $m(x) = \{(a, 1), (b, 2), (c, 2)\}$. Alternativamente $m(x) = \Psi(x)$.

Seguimos avanzando con más definiciones matemáticas para poder representar la disposición de las membranas en la estructura, ya que habrá membranas que estén contenidas en otras o simplemente serán vecinas unas con otras.

Consideremos el lenguaje MS sobre el alfabeto $\{[,]\}$ cuyas cadenas se definen de forma recursiva como sigue:

1. $[] \in MS$;

2. Si $\mu_1 \ldots, \mu_n \in MS, \mathrm{n} \geq 1,$ entonces $[\mu_1 \ldots \mu_n] \in MS$;

3. No hay nada más en MS.

Consideremos ahora la relación sobre los elementos de MS: $x \sim y$ si y sólo si podemos escribir dos cadenas con la forma $x = \mu_1\mu_2\mu_3\mu_4, y = \mu_1\mu_3\mu_2\mu_4,$ para cada $\mu_1\mu_4 \in MS\ y\ \mu_2\mu_3 \in MS$, i.e., dos pares de corchetes al mismo nivel que se intercambian junto con sus contenidos. Esta relación es de equivalencia ya que es reflexiva, simétrica y transitiva y, por tanto, denotamos por $\overline{MS}$ al conjunto de las clases de equivalencia de MS con respecto a la relación $\sim$ cuyos elementos son denominados *estructuras de membranas*.

Cada par de corchetes [,] que aparezca en una estructura de membranas denotaría una *membrana*. El número de membranas en una estructura de membranas μ se denomina *grado* de μ y se denota por $deg(\mu)$ y la membrana más externa es llamada *piel*. Una membrana que no contiene a ninguna membrana en su interior (cuya representación es [ ]) es una membrana *elemental*.

La *profundidad* de una estructura μ se denota como $dep(\mu)$ y se define recursivamente como sigue:

1. Si $\mu = [\,], \text{entonces } dep(\mu) = 1$;

2. $Si\ x = [\mu_1, \ldots, \mu_n]$ para algún $\mu_1, \ldots, \mu_n \in MS, \text{entonces } \boldsymbol{dep}(\mu) = max\{\boldsymbol{dep}\,(\mu_i) \mid 1 \leq i \leq n\} + 1.$

Para finalizar esta sección introduciremos los elementos principales de los sistemas P básicos de transición. Un sistema P de transición es una tupla

$$\Pi = (V, \mu, w_1, \dots, w_n, (R_1, \rho_1), \dots, (R_n, \rho_n), i_0)$$

1. V es un alfabeto cuyos elementos son objetos.

2. μ es una estructura de membranas de grado n con las membranas etiquetadas en una correspondencia uno a uno con los elementos de cierto conjunto Λ. Habitualmente se utilizarán las etiquetas *1, 2, …, n* .

3. w_i, $1 \leq i \leq n$, son cadenas de V^* que representan multiconjuntos sobre V asociados con las regiones $1,2, \dots, n$ de μ.

4. R_i, $1 \leq i \leq n$ son conjuntos finitos de reglas de evolución sobre V asociadas a las regiones $1,2, \dots, \mathrm{n}$ de μ. Una regla siempre vendrá acompañada de objetos con los items *here, out* o *in* excepto si la regla es de disolución de membrana.

5. ρ_i es una relación de orden parcial sobre $R_{i,}$, $1 \leq i \leq n$ que determina la prioridad sobre las reglas de R_i.

6. i_0 identifica a la membrana de salida de Π.

A continuación, hacemos hincapié sobre algunos de los aspectos que acabamos de abordar respecto a la especificación de un sistema P.

Una regla de evolución es un par (u, v) que habitualmente se escribe como $\mathrm{u} \longrightarrow v$ donde u es una cadena sobre V y $v = v'$ o $\mathrm{v} = \mathrm{v}'\ \delta$ donde v'es una cadena sobre:

$$(V \times \{here, out\}) \cup (V \times \{in_j \mid 1 \leq j \leq n\})$$

y δ es un símbolo especial que no pertenece a V.

Una vez reflejados los componentes que definen un sistema P, resulta interesante describir el proceso de computación del sistema en base a las reglas y qué restricciones se imponen según el estado del sistema.

A continuación, listamos algunos puntos semánticos asociados a todo sistema P:

- El sistema debe contener una etiqueta asociada a la membrana piel y esta nunca puede ser disuelta mediante alguna regla.
- La membrana de salida nunca puede ser disuelta.
- Si $u \longrightarrow v$ pertenece al conjunto de reglas que se aplican sobre una membrana i y el multiconjunto u aparece en dicha membrana, entonces:

    - El multiconjunto *u* se elimina de la membrana i.
    - Si $(\mathrm{v}_1,\ \mathrm{here})$ aparece en v, se anade v_1 a la membrana i.
    - Si $(v_1,\ out)$ aparece en v, se anade v_1 a la membrana padre de i (al entorno si i es la piel).
    - Si $\left(v_1,\ in_j\right)$ aparece en v, se anade el v_1 a la membrana j (siempre que j sea un hijo de i).
    - Si δ aparece en v, la membrana i se disuelve y sus objetos pasan al primer antecesor no disuelto siempre que no sea la piel.

Es importante reiterar, tal y como introdujimos anteriormente, que las reglas se ejecutan en paralelo, en forma maximal y no determinista, lo que implica que si se dos reglas son aplicables para la membrana en el paso de ejecución se seleccionará la más prioritaria en el caso de que dicha prioridad se haya establecido y si no, se elegirá de forma aleatoria. El número de veces que se aplica la regla elegida será el máximo posible.

Un paso de computación se considera de parada si no hay reglas que se puedan aplicar. El resultado de una computación de parada estará codificado en la membrana de salida del sistema.

Para una mayor comprensión, procedemos, mediante un diagrama, ilustrar y explicar la representación de un sistema P.

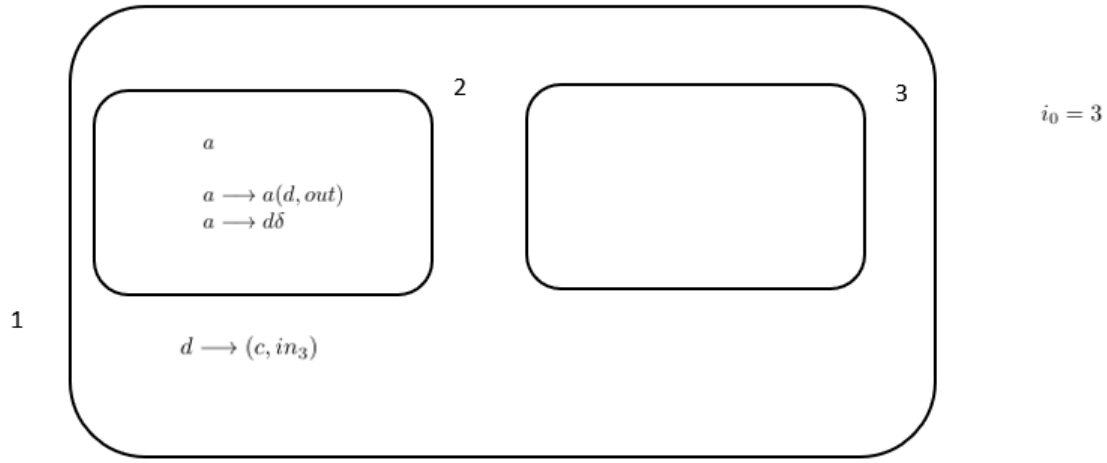


Figura 3: Diagrama sistema P

En este caso, tenemos un sistema de 3 membranas, donde la membrana 1 es la piel y la membrana 3 es la de salida. Respecto al conjunto de reglas, se aplican dos sobre la membrana 2 y una sobre la 1. El proceso de ejecución de reglas se describe de la siguiente manera:

En la membrana uno, cuando se encuentre una '*d'*, esta, desaparecerá de la membrana y se enviará una '*c*' a la membrana 3 por ser una regla de tipo *in*.

Cuando se ejecuten las reglas sobre 2, ya que pueden ejecutarse cualquiera de las dos existentes y no hay ningún criterio de prioridad establecido, entonces solo se procesará una. Si es la primera, la '*a'* desaparecerá y se genera una '*a'*, permaneciendo en la propia membrana (tipo *here*) y una '*d'* se enviará fuera de la membrana (tipo *out*) con lo que pasará a la membrana antecesora, en este caso a la membrana 1.

Si por el contrario se ejecutase la segunda, entonces se elimina la *'a'* y se genera un *'d'* que permanece (tipo *in*) y posteriormente la membrana se disolvería, sus reglas desaparecerían y todos los objetos que estuvieran en dicha membrana pasarían a ser de la membrana 1 por ser su membrana padre.

El sistema P, llegará a un estado de parada cuando ya no puedan aplicarse más reglas, en cuyo caso, se tomarán los objetos permanecientes en la membrana de salida (la 3 en este ejemplo) mediante $\Psi(L)$, siendo x el multiconjunto de la membrana 3. Alternativamente, contarían los objetos que la membrana 3 contiene.

Cabe advertir que en este caso no hay ningún objeto que vaya a ser enviado al entorno, porque no tenemos, por ejemplo, ninguna regla tipo *out* que sea aplicada sobre la membrana piel.

3 Lenguaje de especificación de sistemas P

Para poder programar nuestro intérprete de sistemas P necesitamos primeramente obtener un vocabulario específico que sea capaz de representar cada sistema que queramos que sea interpretado para posteriormente ser procesado.

En nuestro estudio previo hemos investigado qué tipos de lenguajes existían para la especificación de sistemas P y, tras nuestra búsqueda nos hemos detenido en el lenguaje P-Lingua. El P-Lingua es un lenguaje de programación diseñado para describir y especificar los comportamientos de los sistemas P (9).

El P-Lingua, proporciona una forma de representar las reglas y las interacciones entre los objetos contenidos en las membranas, permitiendo simular y analizar el comportamiento de los sistemas P. Se utiliza para definir los sistemas P y realizar experimentos computacionales con ellos, explorando sus propiedades y su capacidad para resolver problemas complejos.

En resumen, el P-Lingua es un lenguaje formal utilizado en la computación con membranas para describir y especificar los sistemas P y sus comportamientos. Proporciona una herramienta para estudiar y analizar estas estructuras y su capacidad computacional. En (9) se puede obtener más información acerca del P-Lingua.

Para poder construir nuestro interprete debíamos pasar por cada una de las fases que requiere cualquier compilador, las cuales incluyen un análisis léxico seguido de un análisis sintáctico para concluir con un análisis semántico.

En primer lugar, hemos diseñado una propuesta de sintaxis para la especificación sistemas P. Dicha propuesta es la que presentamos a continuación:

#Objects <string>	Indica los objetos que se manejan en el sistema
#Membrane <label>	Indica el nombre (<label>) de una membrana

#Membrane <label> objects <string>	Indica los objetos que contiene una membrana inicialmente
#Membrane <label> skin	Indica que una membrana es la más externa
#Membrane <label> out	Indica que una membrana es de salida
#Out infinity	Indica que la salida del sistema es el entorno
#Structure <estructura>	Indica la estructura de membranas. Una estructura es una cadena de paréntesis etiquetados bien parentizada. Por ejemplo , “(1 (2)2 (3)3)1”
#Rule <label> <regla> from <membrane label>	Indica a que membrana se le aplica la regla etiquetada
#Priority <rule label i, rule label j>	Indica que la regla más a la izquierda tiene una prioridad mayor.

En cuanto a la sintaxis para la definición de reglas, comenzamos nuestro trabajo con reglas simples de evolución y estas vienen definidas como sigue:

String_1:String_2

1. **String_1** es un multiconjunto de símbolos sobre un determinado alfabeto

2. **String_2** es una lista de ítems opcionales entre llaves.
3. Los ítems pueden ser pares **{w,z}** donde **w** es un multiconjunto y **z** es o un símbolo "**<**" para especificar si la regla es de tipo *here* o un símbolo "**>**" si de tipo *out* o una **etiqueta de membrana** que hace referencia a una membrana sucesora para especificar que la regla es de tipo *in*.
4. Los ítems pueden ser **{@}** o **{!}** donde el primero representa la cadena vacía y el segundo hace referencia a una disolución de membrana.

Es importante respetar el orden secuencial de cada instrucción tal y como ha sido expuesto anteriormente, esto implica, por ejemplo, que no se pueden definir antes los objetos sobre una membrana, la cual aún no ha sido declarada o los objetos especificarlos después. Toda membrana ha de ser declarada previamente antes de ser utilizada en reglas, asociarle objetos o nombrarla en estructura. De igual manera la prioridad de reglas siempre irá después de especificar el conjunto de reglas.

Otro aspecto para tener en cuenta es que cada símbolo que aparezca en cualquiera de las instrucciones ha de estar establecido en la primera línea junto con la palabra reservada *#Objects*.

Por último, hacer mención que la especificación del sistema P en nuestro lenguaje es sensible a mayúsculas.

Dicho lenguaje deberá ser analizado semánticamente por nuestro intérprete para conseguir respetar la semántica de los sistemas P. Por tanto, los puntos que se han de satisfacer son los siguientes:

1. Para una regla de tipo *in* se ha de comprobar que la etiqueta de membrana a la que hace referencia es una membrana adyacente de la membrana sobre la que se está aplicando la regla.
2. No puede haber una membrana establecida como de salida y también estar declarado el entorno como salida. Ha de haber siempre uno de los dos.
3. Una regla de disolución no puede ser aplicada ni sobre la membrana piel ni sobre la membrana de salida.
4. No puede haber contradicciones de prioridad de ejecución de reglas establecidas en múltiples líneas. Por ejemplo, R1 > R2 ; R2 > R4; R4 > R1.

Para poder tener un mejor entendimiento, procedemos a representar un par de sistemas P utilizando nuestro lenguaje definido anteriormente:

Ejemplo 1

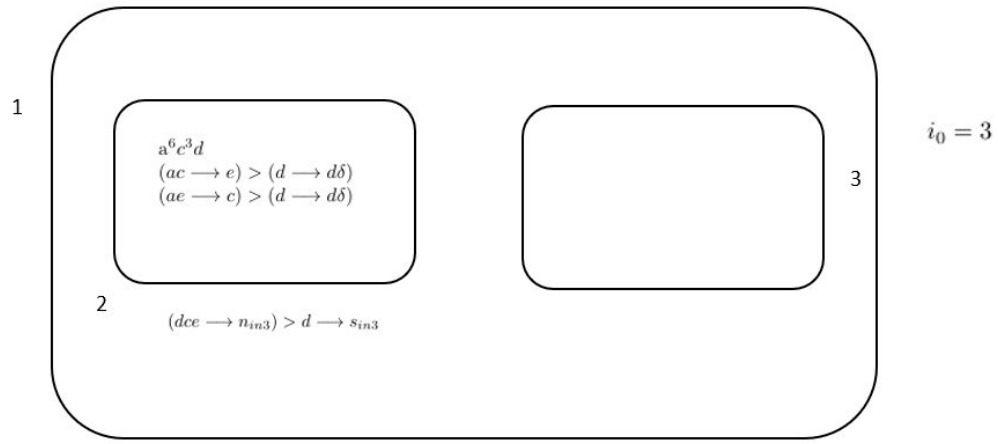


Figura 4: Sistema P ejemplo 1

```
#Objects acdens
#Membrane 1
#Membrane 2
#Membrane 3
#Membrane 2 objects aaaaaacccd
#Membrane 1 skin
#Membrane 3 out
#Structure (1(2)2(3)3)1
#Rule 1 ac:{e,<} from 2
#Rule 2 ae:{c,<} from 2
#Rule 3 d:{d,<}{!} from 2
#Rule 4 dce:{n,3} from 1
#Rule 5 d:{s,3} from 1
#Priority 1,3
#Priority 2,3
#Priority 4,5
```

Figura 5: Código sistema P ejemplo 1

Este sistema determina si el número de *a's* es divisible por el número de *c's*, estableciendo en la membrana de salida una *'s'* o una *'n'*, si es divisible o no, respectivamente. Podemos reemplazar el número de a's y de c's para hacer otras comprobaciones solamente con indicarlo en la línea etiquetada *#Membrane 2 objects*.

Ejemplo 2

En este otro ejemplo el sistema es capaz de generar el conjunto $\{n^2 : \mathrm{n} \geq 1\}$.

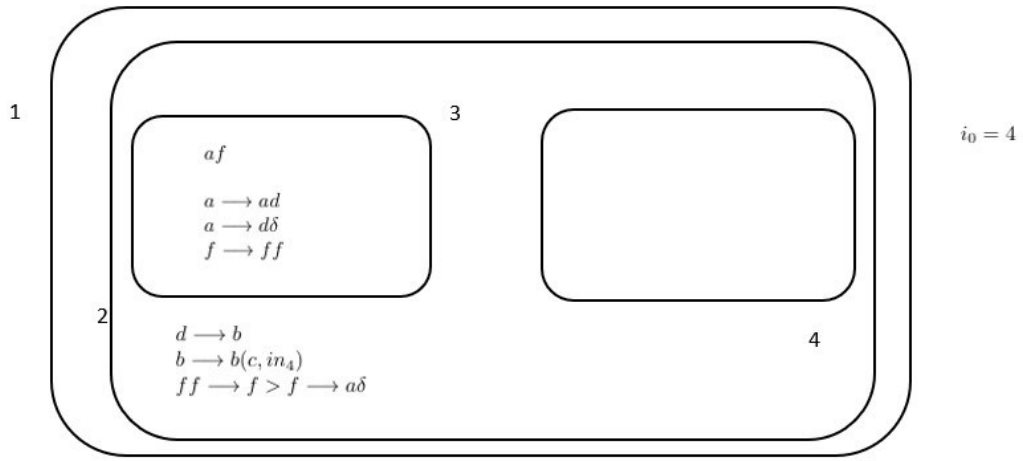


Figura 6: Sistema P ejemplo 2

```
#Objects abcdf
#Membrane 1
#Membrane 2
#Membrane 3
#Membrane 4
#Membrane 3 objects af
#Membrane 1 skin
#Membrane 4 out
#Structure (1(2(3)3(4)4)2)1
#Rule 1 d:{b,<} from 2
#Rule 2 b:{b,<}{c,4} from 2
#Rule 3 ff:{f,<} from 2
#Rule 4 f:{a,<}{!} from 2
#Rule 5 a:{ad,<} from 3
#Rule 6 a:{d,<}{!} from 3
#Rule 7 f:{ff,<} from 3
#Priority 3,4
```

Figura 7: Código sistema ejemplo 2

4 Intérprete del lenguaje

En esta sección abordaremos cómo ha sido el proceso de desarrollo del intérprete. Para ello, la primera decisión fue decidir el lenguaje de programación para a llevar a cabo el trabajo, el cual, fue Python. Utilizar Python para implementar el intérprete de sistemas P presenta varias ventajas.

En primer lugar, diferentes compañeros están trabajando el campo de computación con membranas utilizando Python para sus proyectos. Por otro lado, Python destaca por su sintaxis clara y legible, lo que facilita la comprensión del código y reduce la posibilidad de errores. Además, Python cuenta con una amplia comunidad y abundante documentación, lo que brinda soporte y recursos útiles durante el desarrollo del intérprete. La versatilidad de Python y su amplia gama de bibliotecas especializadas permiten complementar el intérprete de sistemas P con funcionalidades específicas, como procesamiento de imágenes o análisis de datos. Python también ofrece facilidad de integración con otras tecnologías, lo que posibilita combinar sistemas P con enfoques adicionales, como el aprendizaje automático. Por último, la portabilidad y compatibilidad de Python hacen que el intérprete pueda ejecutarse en diferentes sistemas operativos sin mayores modificaciones. En resumen, Python es una elección interesante para implementar el intérprete de sistemas P debido a su sintaxis clara, su comunidad activa, su versatilidad y su facilidad de integración con otras tecnologías.

Antes de continuar describiendo las estructuras que hemos utilizado para nuestro desarrollo hay que tener en cuenta que nuestro interprete contará con dos parámetros de entrada. El primer parámetro será un fichero en texto plano que contendrá las especificaciones del sistema en el lenguaje que hemos visto en la sección anterior. El segundo será un entero positivo para establecer el número de pasos que el sistema P debe ejecutar.

Por lo que respecta al tipo de estructuras utilizadas en el desarrollo, hay que tener en cuenta que había que tener una visión de cómo se iban a recibir las especificaciones y posteriormente como se iba a procesar dicho sistema P. Por ello, a la vez que el intérprete realizaba los respectivos análisis de verificación de especificaciones, se generaba las estructuras necesarias para posteriormente ejecutar el número de veces que fuese necesario.

A continuación, se detalla cómo se ha realizado la programación del intérprete en cuanto a las estructuras y tipos de datos utilizados:

Estructura de pila: Se ha utilizado el concepto de pila para manejar las estructuras de membranas necesarias durante el procesamiento de los sistemas P. Se ha implementado una clase de pila en Python, que incluye métodos para apilar y desapilar elementos. Esta estructura de pila se utiliza para verificar la sintaxis correcta a la hora de declaración de la estructura y así poder ir creando una lista de sucesores para cada membrana y así poder gestionar las operaciones durante la ejecución del intérprete de manera eficiente.

Clases para membranas y reglas: Se han creado clases en Python para representar las membranas y las reglas en los sistemas P. Estas clases contienen atributos y métodos que permiten definir las propiedades y comportamientos de las membranas y las reglas. Por ejemplo, la clase de membranas puede tener atributos como el identificador de la membrana y los objetos contenidos en ella, mientras que la clase de reglas puede tener atributos para la parte izquierda y derecha de la regla.

Prioridades de las reglas: Para manejar las prioridades de las reglas, se ha definido una lista donde cada elemento representa una regla. El orden de los elementos en la lista determina la prioridad, siendo el elemento de menor índice el más prioritario. Esto facilita la gestión de las reglas y su aplicación en el intérprete.

Representación de objetos o multiconjuntos: Los objetos o multiconjuntos se han considerado como cadenas de texto en el intérprete. Esto permite acceder fácilmente a cada posición de la cadena para realizar las operaciones necesarias durante el procesamiento de los sistemas P.

En resumen, el intérprete de sistemas P ha sido programado en Python, utilizando una estructura de pila para gestionar las operaciones, clases para representar las membranas y las reglas, instanciación de objetos, una lista para definir las prioridades de las reglas y la representación de objetos o multiconjuntos como cadenas. Estas decisiones de diseño y programación han permitido implementar un intérprete funcional y eficiente para el procesamiento de sistemas P.

Conviene advertir que, originalmente el intérprete se basaba en una comunicación restringida con el usuario mediante la consola. Pero se ha decidido implementar una pequeña interfaz utilizando la librería *tkinter* para representar el sistema P. Mediante dicha interfaz podremos, en primer lugar, visualizar el estado de la estructura de membranas y sus respectivos objetos. En segundo lugar, podremos ejecutar el número de pasos que hayamos declarado en el parámetro de entrada, bien de una vez o bien ir paso a paso hasta alcanzar el dicho número de pasos. En la siguiente figura se muestra un ejemplo de la interfaz.

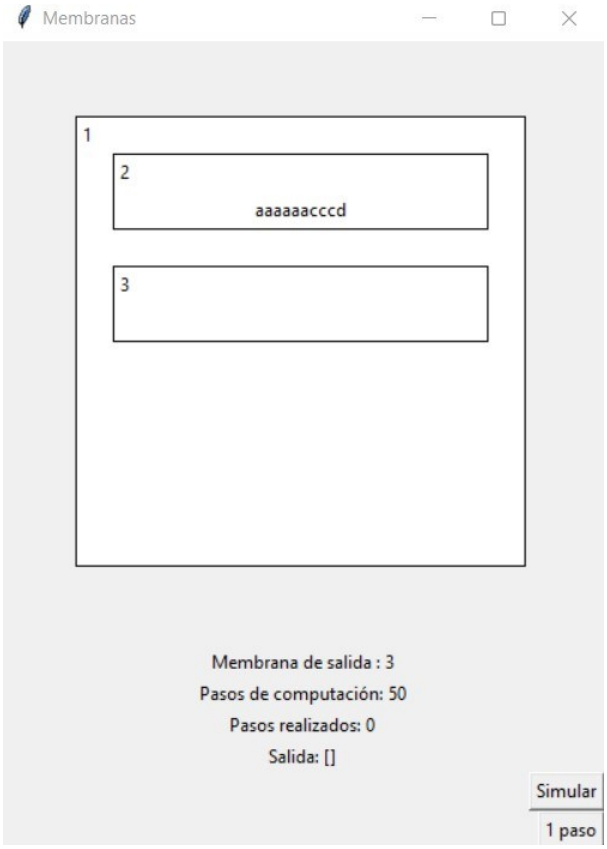


Figura 8: Interfaz del intérprete

4.1 Análisis de las especificaciones

En el intérprete de sistemas P, durante la etapa de análisis de las especificaciones del sistema P, se ha implementado la posibilidad una interacción con el usuario que permite mostrar los errores producidos en dicha etapa. Estos errores son identificados y clasificados mediante un sistema de códigos numéricos y mensajes descriptivos, los cuales brindan información precisa sobre la naturaleza de cada error.

En esta fase, el intérprete procesa línea a línea cada una de las especificaciones propuestas por el sistema P en su entrada de datos y una vez encuentre una de las palabras reservadas al principio de la línea efectúa las correspondientes comprobaciones acorde con el tipo de palabra en cuestión. El proceso es secuencial y en caso de error, se detiene en la línea producente sin continuar más hasta que el usuario solucione el error.

A continuación, se presenta la lista de errores detectados por el intérprete y sus respectivos mensajes descriptivos:

1. Membrana no encontrada.
2. Carácter o parámetro desconocido.
3. Símbolo no encontrado.
4. Número de parámetros incorrecto.
5. Membrana ya definida en estructura, no se puede empezar o cerrar otra vez.
6. Número de separadores de reglas [:] incorrectos.
7. Carácter no permitido.
8. Posición incorrecta del símbolo disolución.
9. Membrana de salida no encontrada.
10. Falta "()" en declaración de estructura.
11. Membrana ya definida.
12. Uso incorrecto de ")" para la membrana.
13. Membrana inicial de estructura sin la propiedad skin.
14. Identificador de regla incorrecto.
15. Regla no encontrada.
16. Identificador de regla duplicado.
17. Regla redundante o prioridad contradictoria.
18. Símbolo "@" duplicado.
19. Disolución no permitida.

20. Salida del sistema ya definida.
21. Número de parámetros incorrecto en la declaración de objetos en regla.
22. Operación de transición duplicada.
23. Operación de transición no permitida.

Cada uno de estos errores se muestra al usuario en caso de que se produzca un problema en la sintaxis o estructura del sistema P y, además, anotando el número de línea donde se ha producido. Estos mensajes de error son fundamentales para que el usuario pueda identificar rápidamente los errores y corregirlos en la etapa de análisis, lo que facilita el desarrollo y la depuración del sistema P.

La interacción entre el intérprete y el usuario, a través de la detección y visualización de errores, garantiza la calidad y la corrección en las especificaciones del sistema P antes de su ejecución. Esto ayuda a mejorar la eficiencia en el proceso de desarrollo y a obtener resultados más confiables en la computación con membranas.

Una vez finalizada la fase de análisis y verificado que las especificaciones del sistema P son correctas, el intérprete procede a construir internamente las estructuras de datos necesarias para representar y procesar cada uno de los elementos del sistema.

Una vez construidas las estructuras de datos, el intérprete está listo para ejecutar el sistema P y realizar las iteraciones especificadas. Durante este proceso, se aplican las reglas de transición y se actualizan los contenidos de las membranas de acuerdo con las interacciones definidas.

4.2 Especificación de estructuras de datos

Antes de describir el procesamiento del programa, es necesario mostrar una representación simplificada de UML de las clases que intervienen en la ejecución, para poder entender mejor posteriormente el proceso:

Membrane
- id - objects - skin - out

- pre - suc - rules - rulesToExec - structure - toOut - toIn - toDisol
+ __init__(id) + execute() + flush()

Tabla 1: Clase Membrana

Atributos:

- id: Representa el identificador de la membrana.
- objects: Cadena de texto que almacena los objetos contenidos en la membrana.
- skin: Indica si la membrana es una membrana exterior (skin).
- out: Indica si la membrana es una membrana de salida (out).
- pre: Lista que almacena los identificadores de las membranas predecesoras.
- suc: Lista que almacena los identificadores de las membranas sucesoras.
- rules: Lista que contiene las reglas asociadas a la membrana.
- rulesToExec: Lista que almacena los índices de las reglas a ejecutar.
- structure: Variable que apunta a la estructura global del sistema.
- toOut: Lista que almacena los objetos a enviar a las membranas de salida.
- toIn: Lista que almacena los objetos a enviar a otras membranas.
- toDisol: Indica si se debe disolver la membrana.

Métodos:

- init(id): Constructor de la clase Membrane. Inicializa los atributos de la membrana.
- execute(): Método que ejecuta las reglas asociadas a la membrana y actualiza los objetos y las acciones a realizar.
- flush(): Método que realiza las acciones de enviar objetos a otras membranas.

Rule
- id - leftSide - rightSide - priority - disol - mb
+ __init__(id)

Tabla 2: Clase Rule

Atributos:

- id: Representa el identificador de la regla.
- leftSide: Cadena de texto que representa el lado izquierdo de la regla.
- rightSide: Lista que almacena los elementos del lado derecho de la regla.
- priority: Variable que almacena la prioridad de la regla.
- disol: Indica si la regla provoca la disolución de una membrana.
- mb: Variable que referencia a la membrana asociada a la regla.

Métodos:

- init(id): Constructor de la clase Rule. Inicializa los atributos de la regla.

Stack
- items
+ __init__(id) + push(x) + pop() + isEmpty() + checkIn(x) + top()

Tabla 3: Clase Stack

Atributos:

- items: Lista que almacena los elementos de la pila.

Métodos:

- init(): Constructor de la clase Pila. Crea una pila vacía.
- push(x): Agrega el elemento x a la pila.
- pop(): Devuelve el elemento tope y lo elimina de la pila.
- isEmpty(): Devuelve True si la pila está vacía, False si no.
- checkIn(x): Devuelve True si el elemento x está en la pila, False si no.
- top(): Devuelve el elemento en la cima de la pila.

En la figura siguiente podemos observar el diagrama de clases en UML para nuestro intérprete.

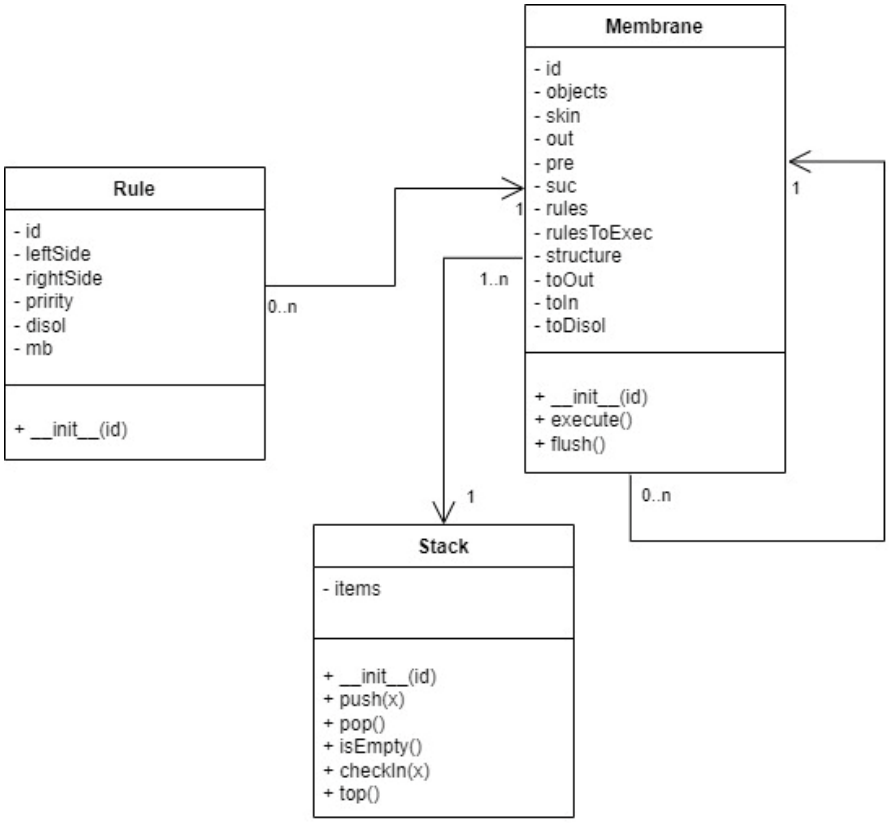


Figura 9: Diagrama de clases UML

4.3 Funcionamiento interno

Tras la presentación de los ingredientes principales del interprete, estamos en disposición de mostrar brevemente cómo funciona internamente nuestro código.

1. Recibe como parámetro el nombre de un archivo que contiene la definición del sistema de membranas en un formato específico.
2. Lee el archivo y analiza su contenido para construir el sistema de membranas correspondiente.
    - Abre el archivo y lee su contenido.
    - Analiza el contenido para identificar las membranas, objetos y reglas definidas en el sistema.
    - Crea instancias de objetos y reglas según la definición del archivo.
    - Almacena las instancias en las estructuras de datos correspondientes, como listas o diccionarios.
3. Ejecuta las reglas del sistema en un ciclo hasta que no haya más reglas a ejecutar:
    - Itera sobre todas las membranas del sistema y verifica si hay reglas a ejecutar en cada una.
    - Si hay reglas a ejecutar en una membrana, selecciona una regla y realiza las siguientes acciones:
        - Verifica la existencia del multiconjunto de objetos de la parte izquierda de la regla en la membrana asociada. Si no existe, la regla no puede aplicarse y se pasa a la siguiente regla.
        - Si varias reglas pueden aplicarse y compiten entre sí y no hay ninguna prioridad establecida sobre dichas reglas, se añadirá al conjunto de reglas pendientes de ejecución de la membrana una de ellas de manera aleatoria. Dicha aleatoriedad queda establecida por la función *randint* de la librería *random*, la cual devuelve un número aleatorio contenido entre dos valores que previamente hayan sido establecidos en los argumentos de la función. En caso de que las reglas que compiten tengan establecidas alguna prioridad declarada, entonces se añadiría la más prioritaria dicho conjunto.

4. Para cada membrana en el sistema, se lanza un método propio de la membrana, el cual procesa las reglas pendientes de ejecución que se han añadido en el paso anterior:
    - Verifica si alguna de las reglas pendientes de ejecución de la membrana es una regla de disolución. Si es de disolución, se marca la membrana como "pendiente de disolución". Esto significa que la membrana se disolverá una vez que se ejecuten el resto de las reglas pendientes.
    - Mientras haya reglas pendientes de ejecución:
        - Aplica la regla al contenido de la membrana según el lado izquierdo y derecho de la regla.
        - Actualiza el estado de las membranas involucradas en la regla:
        - Se eliminan objetos de la membrana de origen y se transforman en otros y permanecen en dicha en membrana o se envían a otras según la sintaxis de la regla.
        - Elimina la regla ejecutada de la lista de reglas pendientes de ejecución.
    - Si no hay más reglas pendientes de ejecución para la membrana, se considera que la membrana ha terminado de ejecutar todas sus reglas.
5. Para cada membrana que tenga la marca "pendiente de disolución", se le aplicará dicho proceso y la membrana padre heredará los objetos de esta. Además, los sucesores de la membrana disuelta pasarán a ser sucesores de la membrana predecesora.

Desde el punto 3 al 5 sería lo que equivaldría a un paso de ejecución, con lo que son estos los puntos que se repetirían en bucle hasta completar en número de pasos deseados en la simulación.

5 Casos de uso y ejemplos

Este capítulo tiene como objetivo presentar los casos de uso y ejemplos que ilustran las funcionalidades del interprete, destacando especialmente su capacidad para detectar e informar errores durante la fase de análisis. Además, se proporcionarán ejemplos de simulaciones de sistemas P para demostrar cómo se pueden lograr los objetivos planteados y cómo el intérprete puede ayudar en la consecución de estos objetivos.

El intérprete cuenta con un sólido mecanismo de detección de errores, tanto sintácticos como semánticos, durante la fase de análisis de un sistema P. Esta capacidad resulta fundamental, ya que permite identificar y corregir problemas en la estructura y lógica del sistema antes de ejecutarlo, ahorrando tiempo y evitando errores costosos. En el siguiente subapartado, se presentarán diferentes escenarios que ilustran estos tipos de errores y cómo el intérprete los detecta y reporta.

5.1 Detección de errores de especificación

Antes de que se realice el análisis sintáctico y semántico de un sistema P, se lleva a cabo el análisis léxico. Este proceso se encarga de descomponer el código fuente en una secuencia de tokens, identificando palabras clave, identificadores, operadores y otros elementos. Durante el análisis léxico, el programa verifica que los tokens sean válidos y estén correctamente estructurados. En caso de encontrar un error léxico, como una palabra clave incorrecta o un carácter inválido, el intérprete lo detectará y proporcionará un mensaje de error descriptivo.

Los errores sintácticos son aquellos relacionados con la estructura y gramática del sistema. Estos errores pueden incluir el uso incorrecto de palabras clave, la falta de elementos obligatorios o el uso inadecuado de operadores. En nuestro caso, se examina minuciosamente la estructura del sistema y se emiten mensajes de error claros y descriptivos en caso de detectar una sintaxis incorrecta.

Los errores semánticos se refieren a problemas en la lógica y coherencia del sistema. Estos errores pueden incluir reglas contradictorias, incoherencias en la especificación de objetos o membranas, o reglas que no se pueden aplicar debido a restricciones lógicas. El intérprete de Sistemas P realiza un análisis semántico exhaustivo para detectar este tipo de errores y proporcionar información precisa sobre su origen y naturaleza.

A continuación, se muestran algunos ejemplos que ilustran cada uno de los tipos de errores que podemos encontrar a la hora de especificar un sistema P, según su naturaleza sea tipo léxico, sintáctico o semántico, y como el programa es capaz de detectarlo. La batería de casos la realizaremos sobre el sistema ejemplo 1 que presentemos en secciones anteriores. Sobre dicho caso mostraremos como modificaremos parte de las especificaciones para comprobar la eficacia de la detección de errores del intérprete.

Los ejemplos ilustrados serán del fichero de las especificaciones del sistema P incluyendo el correspondiente error en dicho fichero y su posterior salida por consola del propio intérprete notificando el error.

En primer lugar, podemos apreciar como el intérprete detecta errores al no encontrar una palabra reservada al principio de línea en este caso #Membrane en la línea 3.

```
#Objects acdens
#Membrane 1
#membrane 2
#Membrane 3
#Membrane 2 objects aaaaaacccd
#Membrane 1 skin
#Membrane 3 out
#Structure (1(2)2(3)3)1
#Rule 1 ac:{e,<} from 2
#Rule 2 ae:{c,<} from 2
#Rule 3 d:{d,<}{!} from 2
#Rule 4 dce:{n,3} from 1
#Rule 5 d:{s,3} from 1
#Priority 1,3
#Priority 2,3
#Priority 4,5
```

```
In [2]: runfile('D:/python/pinterpreter.py', wdir='D:/python')
Error en la linea 3. Carácter o parámetro desconocido.
```

Figura 10: Caso error 1

En este caso hemos intercambiado los “:” por “;” .

```
#Objects acdens
#Membrane 1
#Membrane 2
#Membrane 3
#Membrane 2 objects aaaaaacccd
#Membrane 1 skin
#Membrane 3 out
#Structure (1(2)2(3)3)1
#Rule 1 ac;{e,<} from 2
#Rule 2 ae:{c,<} from 2
#Rule 3 d:{d,<}{!} from 2
#Rule 4 dce:{n,3} from 1
#Rule 5 d:{s,3} from 1
#Priority 1,3
#Priority 2,3
#Priority 4,5
```

```
In [4]: runfile('D:/python/pinterpreter.py', wdir='D:/python')
Error en la linea 9. Número de separadores de reglas [:] incorrectos.
```

Figura 11: Caso error 2

Para este caso hemos optado por no cerrar una llave en la línea 9.

```
#Objects acdens
#Membrane 1
#Membrane 2
#Membrane 3
#Membrane 2 objects aaaaaacccd
#Membrane 1 skin
#Membrane 3 out
#Structure (1(2)2(3)3)1
#Rule 1 ac:{e,< from 2
#Rule 2 ae:{c,<} from 2
#Rule 3 d:{d,<}{!} from 2
#Rule 4 dce:{n,3} from 1
#Rule 5 d:{s,3} from 1
#Priority 1,3
#Priority 2,3
#Priority 4,5
```

```
In [10]: runfile('D:/python/pinterpreter.py', wdir='D:/python')
Error en la linea 9. Número de parámetros incorrecto.
```

Figura 12: Caso error 3

Ya que el orden sí que importa al especificar los objetos del sistema en la última línea cuando en la 4 se menciona que la membrana 2 tiene los objetos aaaaaacccd, realmente el intérprete ni tiene constancia de los citados objetos.

```
#Membrane 1
#Membrane 2
#Membrane 3
#Membrane 2 objects aaaaaacccd
#Membrane 1 skin
#Membrane 3 out
#Structure (1(2)2(3)3)1
#Rule 1 ac:{e,<} from 2
#Rule 2 ae:{c,<} from 2
#Rule 3 d:{d,<}{!} from 2
#Rule 4 dce:{n,3} from 1
#Rule 5 d:{s,3} from 1
#Priority 1,3
#Priority 2,3
#Priority 4,5
#Objects acdens
```

```
In [11]: runfile('D:/python/pinterpreter.py', wdir='D:/python')
Error en la linea 4. Símbolo no encontrado.
```

Figura 13: Caso error 4

Para concluir esta parte de casos de error detectados mencionaremos un par de errores semánticos. El primero de ellos es ponerle la propiedad de piel a la membrana 2 cuando según la estructura de membranas declarada en la línea 8, la membrana más externa es la 1. El otro error tiene que ver con las cláusulas de prioridades, donde es contradictorio que la regla 2 sea más prioritaria que la 1 cuando previamente se ha establecido la regla 1 es más prioritaria que la 3 y esta, a su vez, más prioritaria que la 2, Por tanto, no es posible que la 2 sea más prioritaria que la 1.

```
#Objects acdens
#Membrane 1
#Membrane 2
#Membrane 3
#Membrane 2 objects aaaaaacccd
#Membrane 2 skin
#Membrane 3 out
#Structure (1(2)2(3)3)1
#Rule 1 ac:{e,<} from 2
#Rule 2 ae:{c,<} from 2
#Rule 3 d:{d,<}{!} from 2
#Rule 4 dce:{n,3} from 1
#Rule 5 d:{s,3} from 1
#Priority 1,3
#Priority 2,3
#Priority 4,5
```

```
In [12]: runfile('D:/python/pinterpreter.py', wdir='D:/python')
Error en la linea 8. Membrana inicial de estructura sin la propiedad skin.
```

Figura 14: Caso error 5

```
#Objects acdens
#Membrane 1
#Membrane 2
#Membrane 3
#Membrane 2 objects aaaaaacccd
#Membrane 1 skin
#Membrane 3 out
#Structure (1(2)2(3)3)1
#Rule 1 ac:{e,<} from 2
#Rule 2 ae:{c,<} from 2
#Rule 3 d:{d,<}{!} from 2
#Rule 4 dce:{n,3} from 1
#Rule 5 d:{s,3} from 1
#Priority 1,3
#Priority 3,2
#Priority 4,5
#Priority 2,1
```

```
In [166]: runfile('D:/python/pinterpreter.py', wdir='D:/python')
Error en la linea 17. Prioridad repetida o contradictoria.
```

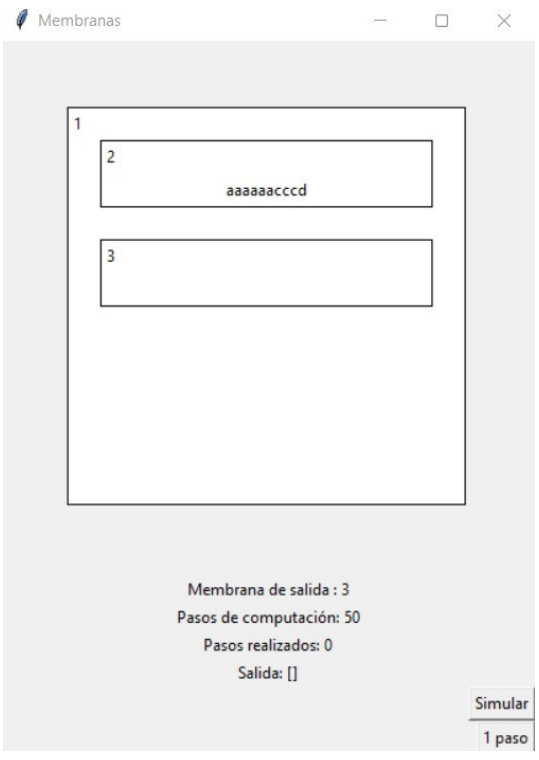
Figura 15: Caso error 6

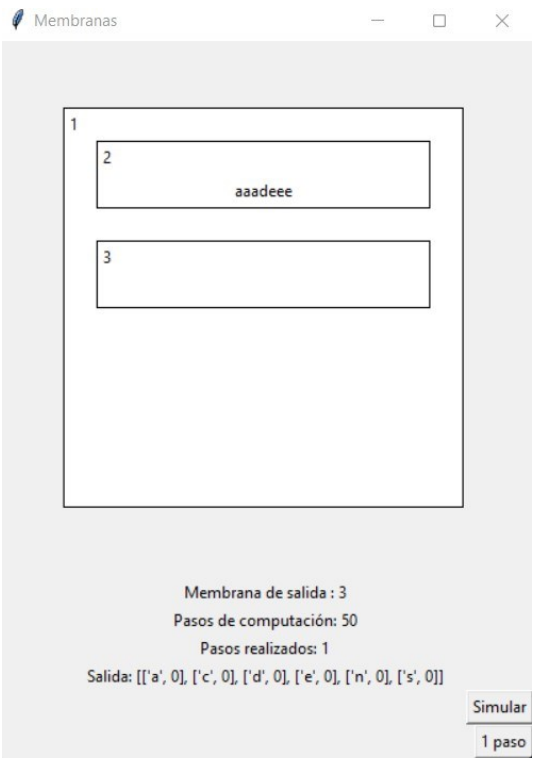
5.2 Simulación de sistemas P

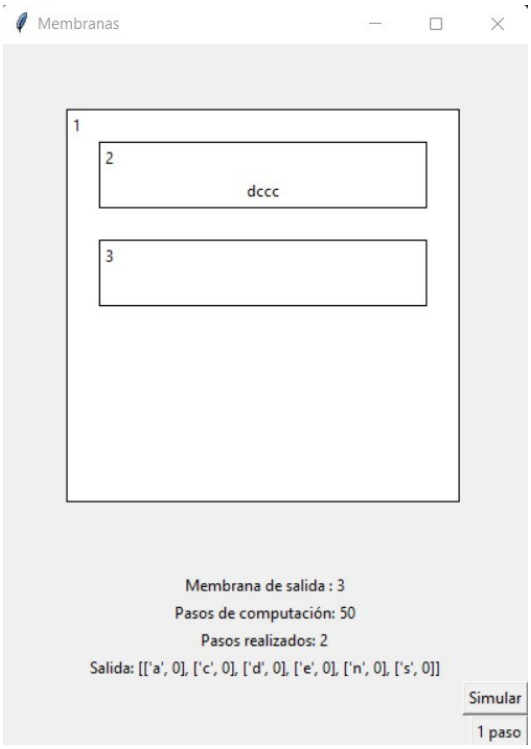
En este último apartado de casos de uso, procedemos a mostrar cómo el intérprete es capaz de procesar el sistema el número de pasos que otorguemos por parámetros. Para ello, construiremos diferentes sistemas P y continuaremos con su posterior simulación. Al margen de que la aplicación nos permite ejecutar todas las iteraciones de golpe que se le haya pasado por parámetros, en los siguientes ejemplos obtendremos el resultado final mediante la simulación paso a paso del intérprete para que nos permita observar cómo se producen las transiciones en cada uno de los pasos.

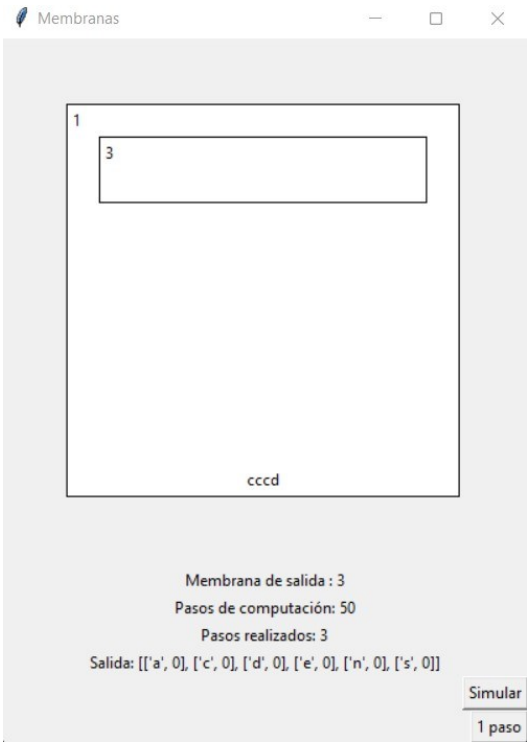
En primer lugar, comenzamos por simular el sistema con el que hemos estado trabajando en apartados anteriores cuya referencia inicial correspondía al sistema ejemplo 1. Este sistema P identifica un criterio de divisibilidad de la expresión $a^n c^k d$, concretamente en la membrana de salida obtendremos una $'s'$si n es divisible por k o un 'n' si no lo es. Seguidamente, ejemplificamos las dos alternativas:

Simulación 1









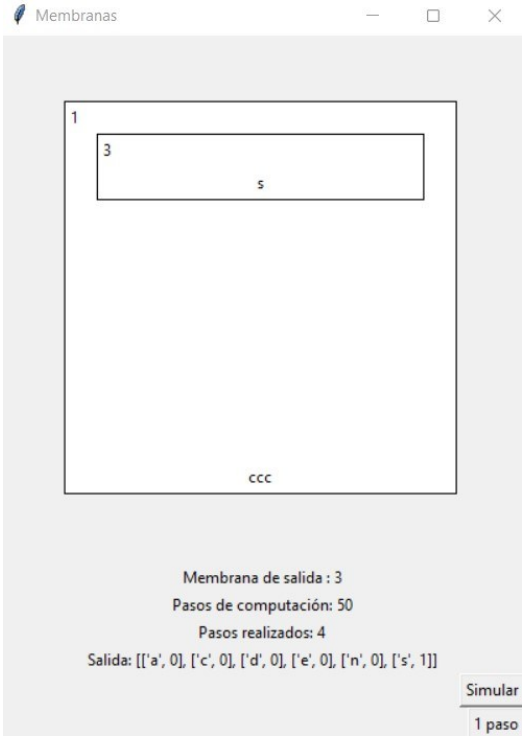


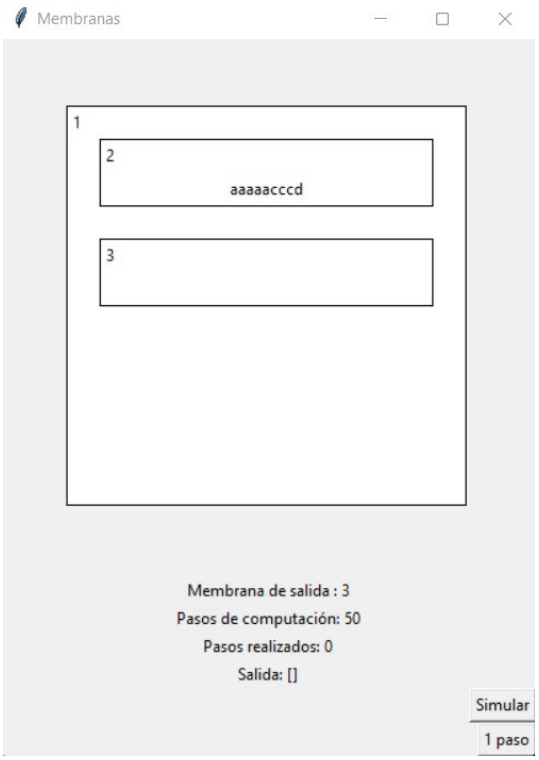
Figura 16: Simulación 1

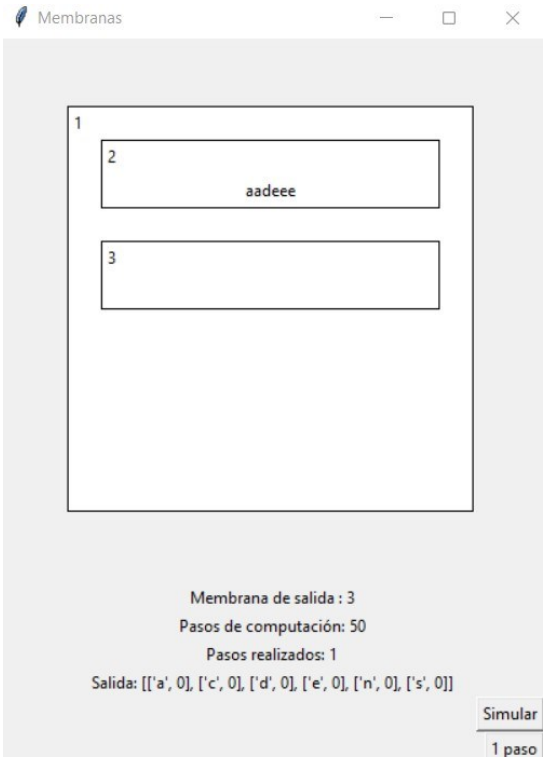
La tabla de los objetos en las membranas respecto al paso de ejecución es:

Paso	Membrana 1	Membrana 2	Membrana 3
0		a^6c^3d	
1		a^3de^3	
2		dc^3	
3	c^3d	Disuelta	
4	c^3		s

Tabla 4 Transiciones paso a paso de la simulación 1

Simulación 2





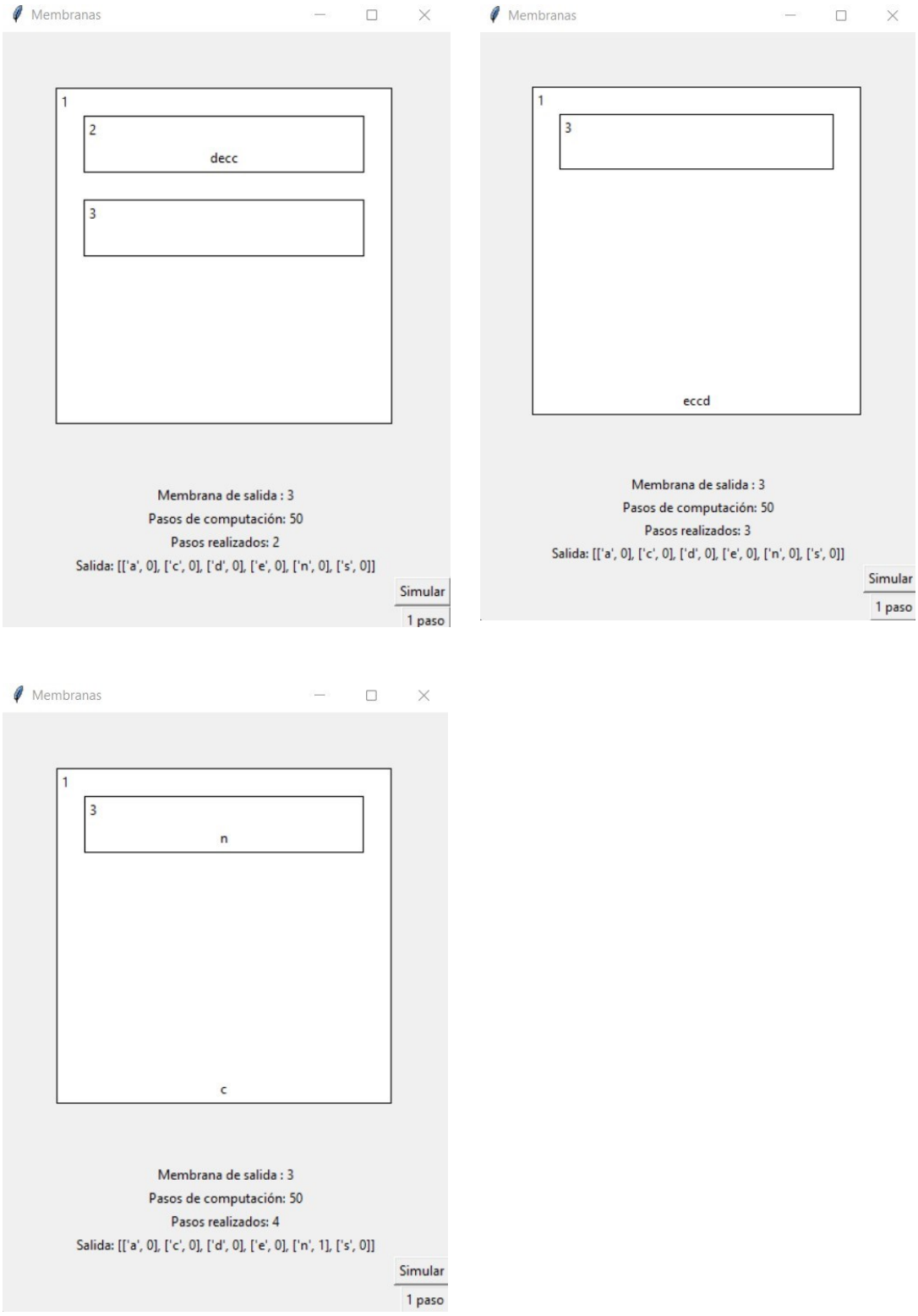


Figura 17: Simulación 2

La tabla de los objetos en las membranas respecto al paso de ejecución es:

Paso	Membrana 1	Membrana 2	Membrana 3
0		a^5c^3d	
1		a^2de^3	
2		dec^2	
3	ec^2d	Disuelta	
4	c	Disuelta	n

Tabla 5: Transiciones paso a paso de la simulación 2

Simulación 3

En este último ejemplo actuaremos sobre el sistema que mostramos como ejemplo 2 en secciones anteriores donde el sistema era capaz de generar el conjunto $\{n^2 : n \geq 1\}$ y cuya representación era la siguiente:

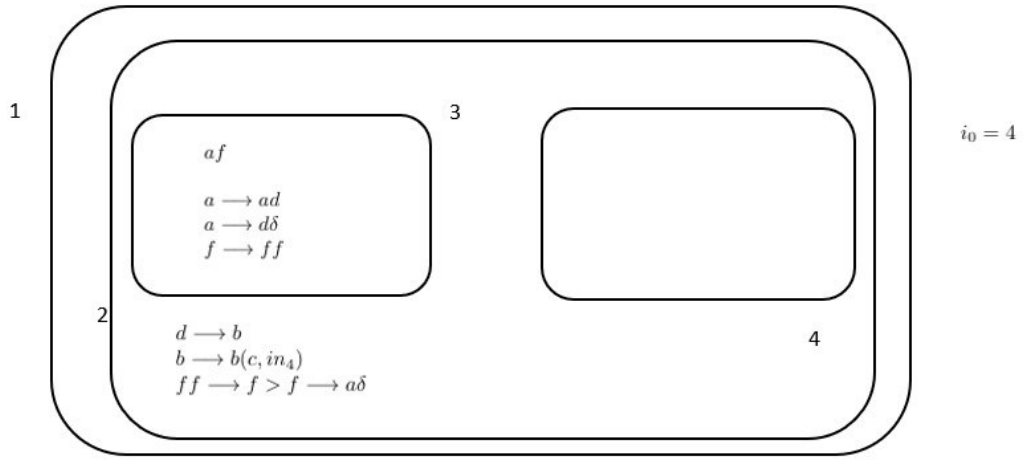
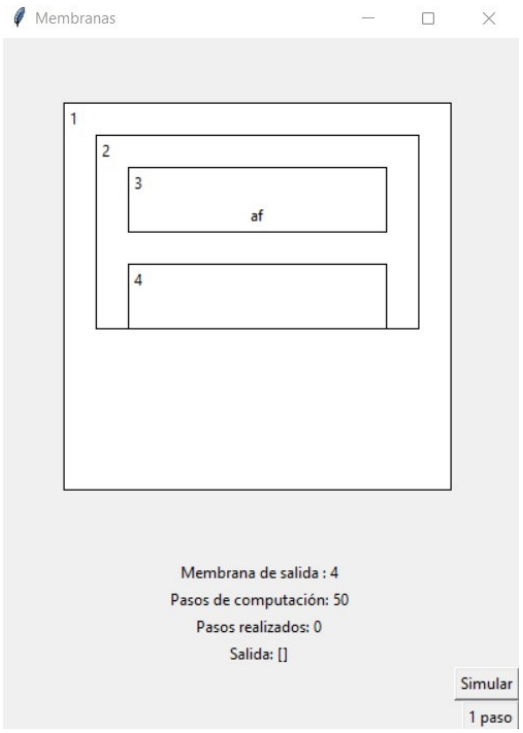
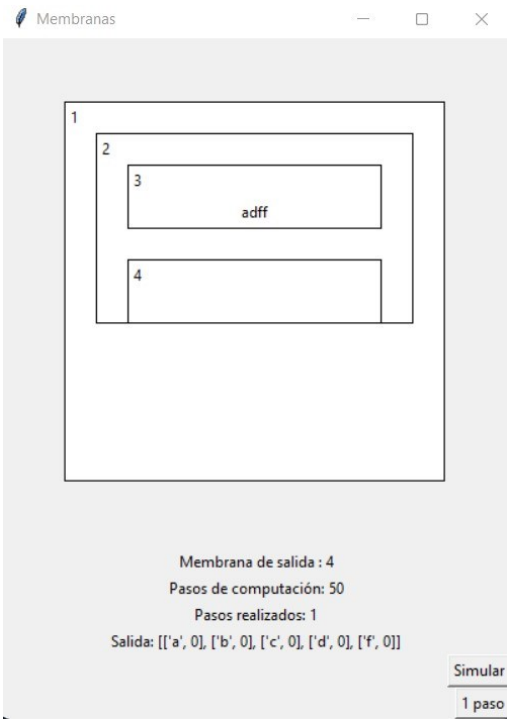
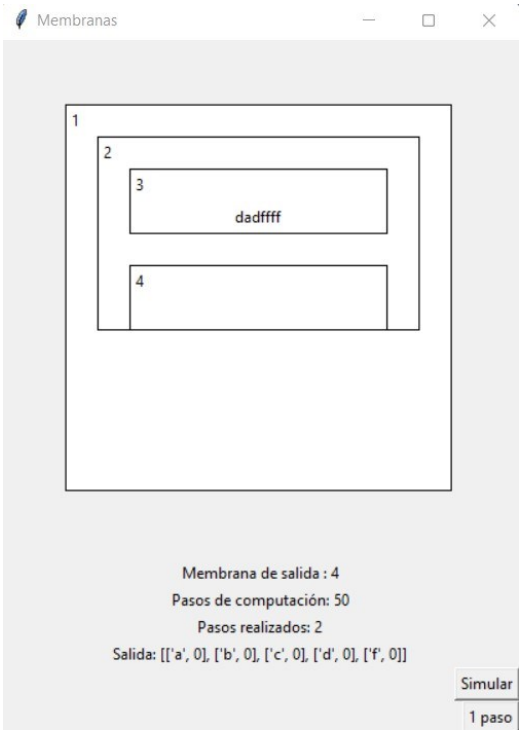


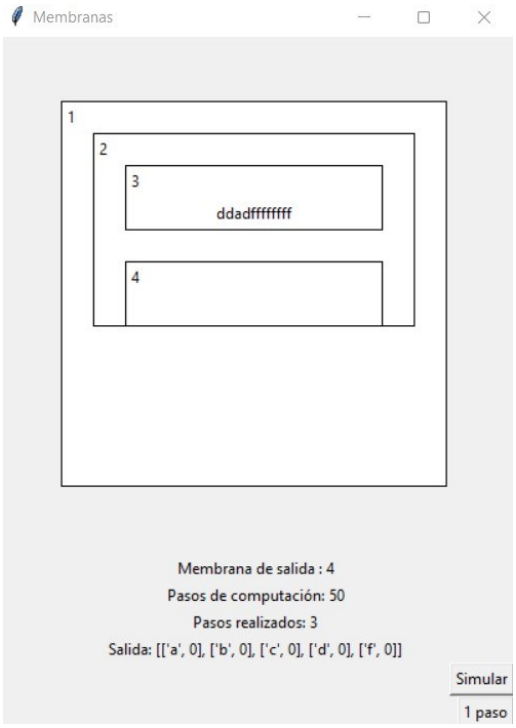
Figura 18: Diagrama sistema simulación 3

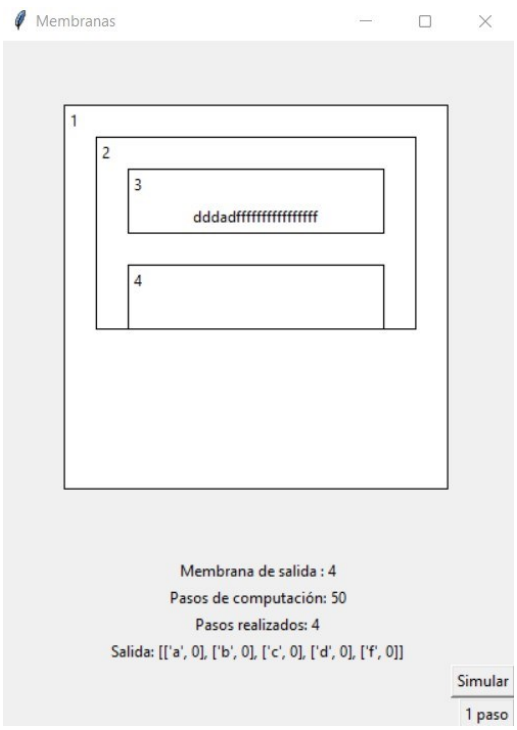
Y sus pasos de simulación:

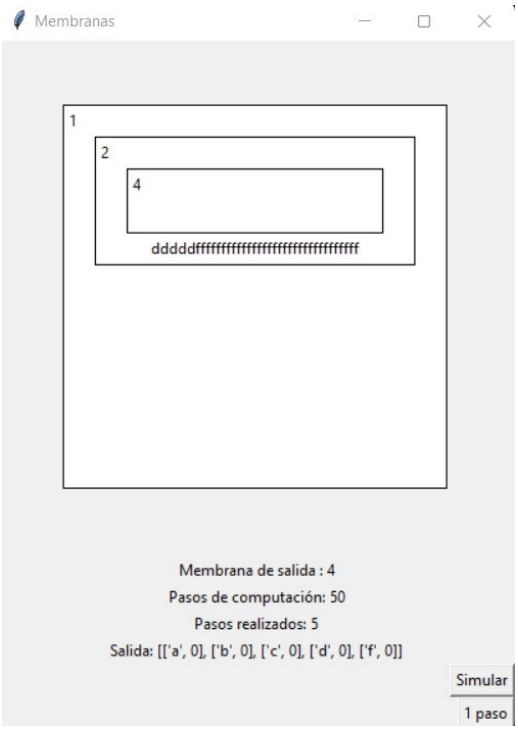

Membranas
1
2
3
af
4
Membrana de salida : 4
Pasos de computación: 50
Pasos realizados: 0
Salida: []
Simular
1 paso

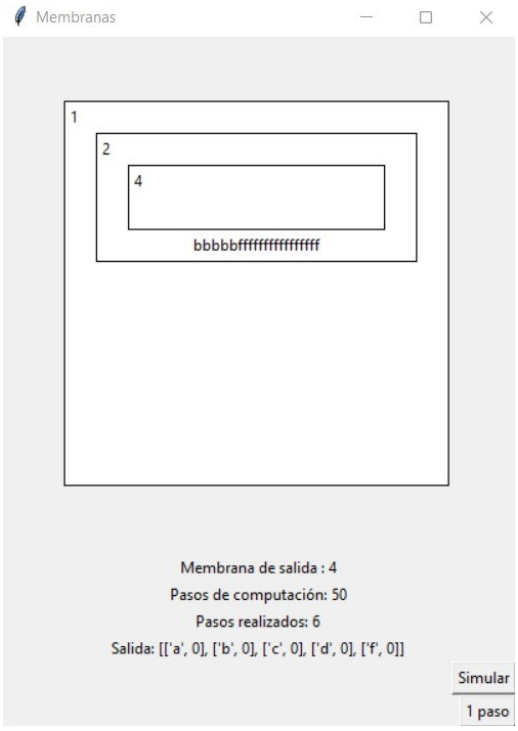

Membranas
1
2
3
adff
4
Membrana de salida : 4
Pasos de computación: 50
Pasos realizados: 1
Salida: [['a', 0], ['b', 0], ['c', 0], ['d', 0], ['f', 0]]
Simular
1 paso

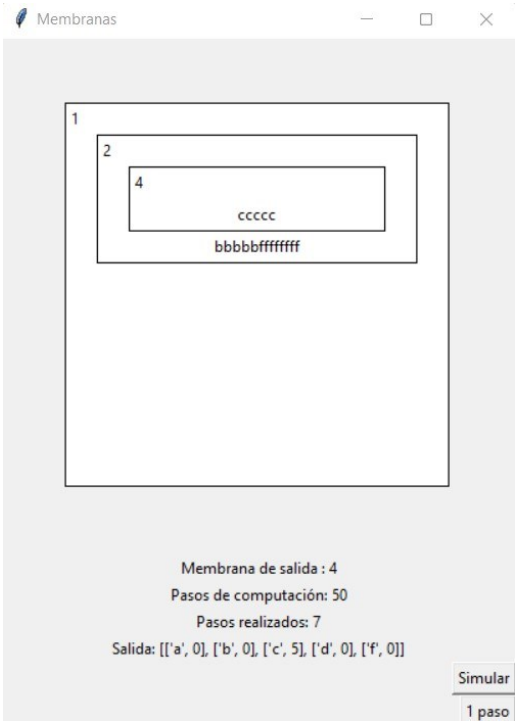

Membranas
1
2
3
dadffff
4
Membrana de salida : 4
Pasos de computación: 50
Pasos realizados: 2
Salida: [['a', 0], ['b', 0], ['c', 0], ['d', 0], ['f', 0]]
Simular
1 paso


Membranas
1
2
3
ddadfffffffff
4
Membrana de salida : 4
Pasos de computación: 50
Pasos realizados: 3
Salida: [['a', 0], ['b', 0], ['c', 0], ['d', 0], ['f', 0]]
Simular
1 paso


Membranas
1
2
3
dddadfffffffffffffffff
4
Membrana de salida : 4
Pasos de computación: 50
Pasos realizados: 4
Salida: [['a', 0], ['b', 0], ['c', 0], ['d', 0], ['f', 0]]
Simular
1 paso


Membranas
1
2
4
dddddfffffffffffffffffffffffffffff
Membrana de salida : 4
Pasos de computación: 50
Pasos realizados: 5
Salida: [['a', 0], ['b', 0], ['c', 0], ['d', 0], ['f', 0]]
Simular
1 paso


Membranas
1
2
4
bbbbbfffffffffffffffff
Membrana de salida : 4
Pasos de computación: 50
Pasos realizados: 6
Salida: [['a', 0], ['b', 0], ['c', 0], ['d', 0], ['f', 0]]
Simular
1 paso


Membranas
1
2
4
ccccc
bbbbbfffffffff
Membrana de salida : 4
Pasos de computación: 50
Pasos realizados: 7
Salida: [['a', 0], ['b', 0], ['c', 5], ['d', 0], ['f', 0]]
Simular
1 paso

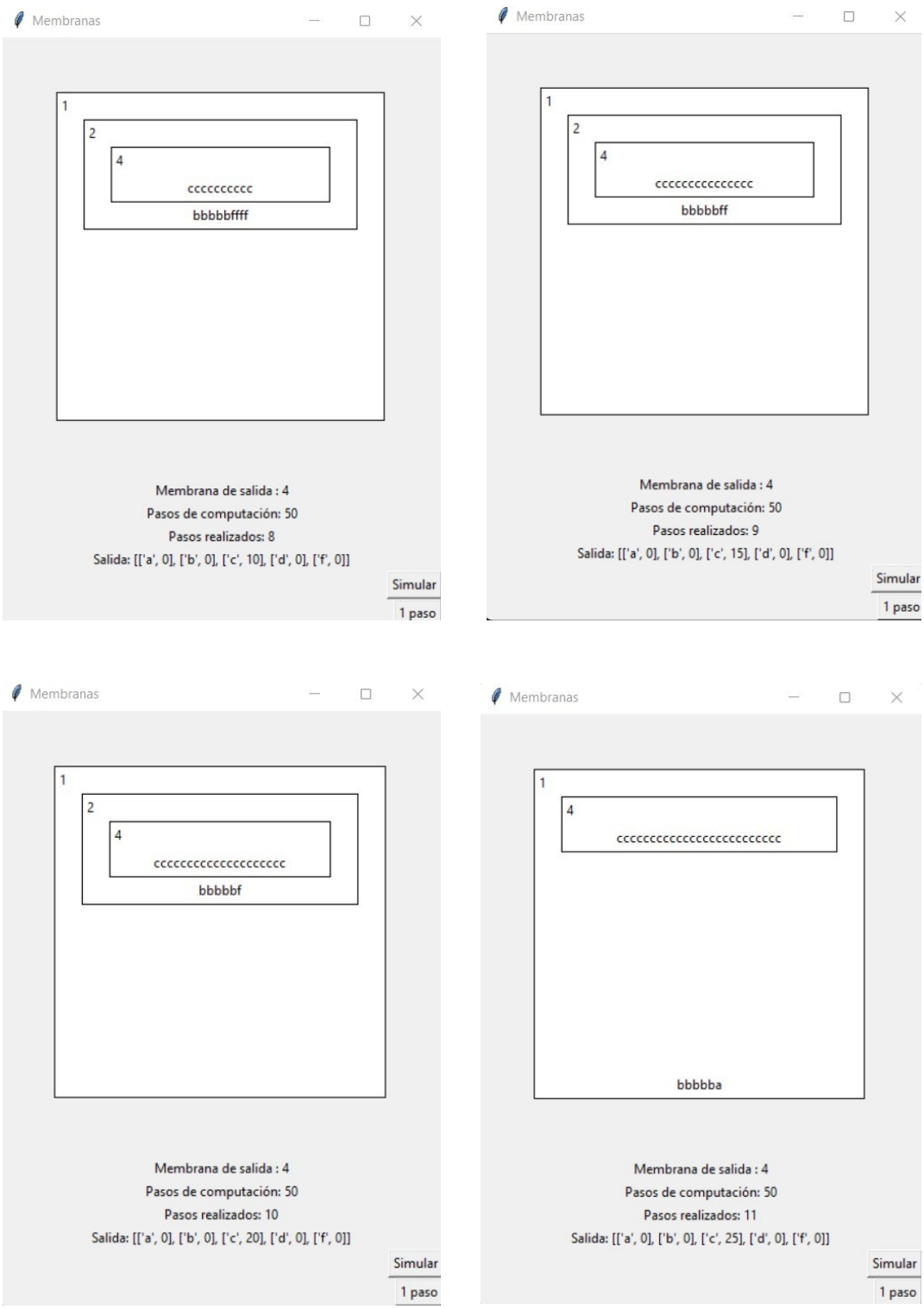


Figura 19: Simulación 3

La tabla de estados de objetos respecto a los pasos es:

Paso	Membrana 1	Membrana 2	Membrana 3	Membrana 4
0			af	
1			adf^2	
2			ad^2f^4	
3			ad^3f^8	
4			ad^4f^{16}	
5		d^5f^32	Disuelta	
6		b^5f^16	Disuelta	
7		b^5f^8	Disuelta	c^5
8		b^5f^4	Disuelta	c^{10}
9		b^5f^2	Disuelta	c^{15}
10		b^5f	Disuelta	c^{20}
11	b^5a	Disuelta	Disuelta	c^{25}

Tabla 6: Transiciones paso a paso de la simulación 3

6 Conclusiones

En el presente trabajo de fin de grado se ha abordado el desarrollo de un intérprete de sistemas P, un lenguaje de especificación orientado a la modelización y simulación de sistemas P. A lo largo del desarrollo del proyecto, se han alcanzado los objetivos planteados y se han obtenido resultados satisfactorios.

En primer lugar, se ha especificado una sintaxis para representar sistemas P y posteriormente diseñar e implementar un intérprete funcional que permite ejecutar dichos sistemas utilizando su correspondiente sintaxis. El intérprete ha sido desarrollado siguiendo una metodología iterativa e incremental, lo que ha permitido una evolución gradual y un mejor control de calidad en cada etapa del proceso.

El intérprete cuenta con las funcionalidades básicas del lenguaje, incluyendo la declaración de membranas, estructura de membranas, reglas de evolución, así como la posibilidad de generar prioridades de ejecución entre ellas. Además, se ha incorporado la validación de todos los tipos y el manejo de errores, brindando al usuario mensajes informativos en caso de detectar problemas durante la ejecución del programa.

Adicionalmente, se ha creado una interfaz para proporcionar un entorno amigable y de visualización en la que se ha incluido la opción de ejecución del sistema paso a paso, lo cual proporciona al usuario un mayor control y seguimiento del flujo de ejecución, así como la posibilidad de observar el estado de los objetos y las membranas en cada paso.

Si bien el intérprete ha logrado cumplir con los objetivos establecidos, se identifican posibles líneas futuras de trabajo para mejorar y expandir su funcionalidad. Entre ellas se encuentra la ampliación de las capacidades del lenguaje, explorando la incorporación de nuevas instrucciones y funcionalidades que permitan una mayor flexibilidad y expresividad en la modelización de sistemas P.

Además, se podría trabajar en la optimización del rendimiento del intérprete, implementando técnicas de optimización y evaluando posibles puntos de mejora en el código.

Otra línea futura de trabajo sería explorar la integración del intérprete con otras herramientas de simulación o análisis de sistemas, con el objetivo de ampliar su utilidad y facilitar la interoperabilidad con otros entornos de trabajo. En conclusión, el desarrollo de este intérprete de sistemas P ha sido un paso importante para acercar a los usuarios al lenguaje y permitirles aprovechar sus capacidades en la modelización y simulación de sistemas. A través de futuras mejoras y ampliaciones, se espera seguir potenciando

el uso de sistemas P y contribuir al avance en el campo de la computación celular con membranas.

7 Bibliografía

1. **PĂUN, Gheorghe.** Computing with membranes. *Journal of Computer and System Sciences.* 2000, Vol. 61, 1, págs. 108-143.

2. **DE CASTRO, Leandro Nunes.** *Fundamentals of natural computing: basic concepts, algorithms, and applications.* CRC Press, 2006. s.l. : CRC Press, 2006.

3. **ZHANG, Gexiang et al.** An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems.* 2014, Vol. 24, 5, pág. 1440006.

4. **DÍAZ-PERNIL, Daniel, GUTIÉRREZ-NARANJO, Miguel A. y PENG, Hong.** Membrane computing and image processing: a short survey. *Journal of Membrane Computing.* 2019, Vol. 1, págs. 58-73.

5. **INOUYE, Jon.** Quantum Simulation Using Membrane Computing. *En METMBS.* 2004, págs. 403-409.

6. **GIANNAKIS, Konstantinos, et al.** Qm automata: a new class of restricted quantum membrane automata. *GeNeDis 2016: Computational Biology and Bioinformatics.* s.l. : Publishing, Springer International, 2017. págs. 193-204.

7. **IBARRA, Oscar H.** On the computational complexity of membrane systems. *Theoretical Computer Science.* 2004, Vol. 320, 1, págs. 89-109.

8. **HOPCROFT, John E., MOTWANI, Rajeev y ULLMAN, Jeffrey D.** *Teoría de autómatas, lenguajes y computación.* s.l. : Addison Wesley, 2008.

9. **http://www.p-lingua.org/wiki/index.php/Main_Page. [En línea] [Citado el: 14 de junio de 2023.]**

8 Anexos

8.1 OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**		**X**		
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**			**X**	
ODS 9. **Industria, innovación e infraestructuras.**		**X**		
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**		**X**		

En el presente trabajo de fin de grado, se ha abordado el desarrollo de un intérprete de sistemas P, un lenguaje de especificación orientado a la modelización y simulación de sistemas P. Aunque el trabajo en sí no aborda directamente múltiples aspectos de los Objetivos de Desarrollo Sostenible (ODS) establecidos en la Agenda 2030 de las Naciones Unidas, se puede identificar la posible relación con algunos de ellos. A través de un análisis cuidadoso, podemos destacar aquellos ODS que podrían estar vinculados al desarrollo de este intérprete, teniendo en cuenta sus implicaciones potenciales y el contexto más amplio en el que podría aplicarse.

El desarrollo del intérprete de sistemas P en este TFG podría estar relacionado con los siguientes ODS:

ODS 4: Educación de calidad, ya que el intérprete proporciona una interfaz amigable y de visualización que puede ayudar a los usuarios a comprender y utilizar el lenguaje de manera efectiva. Esto puede contribuir a mejorar la educación y el aprendizaje en el campo de la computación celular.

ODS 8: Trabajo decente y crecimiento económico, ya que desarrollo del intérprete de sistemas P, puede contribuir al crecimiento económico al impulsar la creación de empleo en el sector de la computación y la tecnología. Además, al proporcionar una herramienta útil para la modelización y simulación de sistemas, el intérprete puede ayudar a mejorar la eficiencia y la productividad en diversos sectores de la economía.

ODS 9: Industria, Innovación e Infraestructura, ya que la creación de esta herramienta representa una contribución a la innovación en el campo de la computación celular con membranas, facilitando la modelización y simulación de sistemas y fomentando el avance en esta área.

ODS 17: Alianzas para lograr los objetivos, ya que se plantea la posible integración del intérprete con otras herramientas de simulación o análisis de sistemas. Esto fomentaría la colaboración y la creación de alianzas entre diferentes entornos de trabajo, permitiendo el intercambio de conocimientos y experiencias.

Si bien el desarrollo del intérprete de sistemas P en este TFG no abarca todos los aspectos de los ODS de manera exhaustiva, se destaca la importancia de reconocer las posibles conexiones y oportunidades de mejora que podrían surgir en futuras iteraciones. A través de la ampliación de las capacidades del lenguaje, la optimización del rendimiento y la exploración de la integración con otras herramientas, el intérprete podría contribuir aún más al avance en el campo de la computación celular con membranas y a la consecución de los ODS relacionados.