Universidad Nacional Autónoma de México

Facultad de Ciencias

Sistemas de Metabolismo-Reparación: una implementación exploratoria en programación orientada a objetos

T E S I S

que para obtener el título de:

Licenciado en Ciencias de la Computación

P R E S E N T A :

Enrique Francisco Soto Astorga

TUTOR

Dr. Gustavo de la Cruz Martínez

Ciudad Universitaria, CD. MX., 2022

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Terræ, vera mater universa et æterna, ex qua omnes gnati sumus,
in extremis ad quam revertemur nihilominus.

Para Toñita y Crepita, que la tierra les sea leve.

Gratiæ agendæ

Agradezco al Dr. Gustavo de la Cruz Martínez por haber guiado este trabajo de tesis a buen término con calidez humana y rigor académico, así como por haber ayudado en mi camino en una de las materias que con mayor fuerza estudié en esta Facultad.

Agradezco al M. en C. Everardo Gustavo Robredo Esquivelzeta, mentor y amigo, por haber abierto mis ojos e intuiciones al trabajo del eminente Dr. Robert Rosen, por haber sembrado la semilla del anti-reduccionismo en mi pensamiento, y por las innumerables charlas que hemos sostenido a lo largo de los años.

Igualmente agradezco al M. en C. Sergio Hernández López, al M. en I. Gerardo Avilés Rosas, y al M. en C. Oscar Hernández Constantino por haber aceptado ser jurado para este trabajo de tesis, así como por haber formado parte de mi camino por esta Facultad en la forma más humana y rigurosa.

Finalmente, agradeceré siempre al Dr. Robert Rosen por haber abierto, un poco más, la puerta hacia el conocimiento final de nuestra existencia.

Nuncupatio

Consagro este trabajo a mi abuela, a mis padres y a todos quienes convivieron en aquella gran casa, fuesen animales humanos o no-humanos;

A Amellalli, por mostrarme el camino con su amor y compañía, y por ser mi pilar y razón en esta tierra de fatalidades;

A Daniel, José, Tredok, René y Nazareth, por su paso conmigo en esta Facultad, por su amistad y presencia o ausencia;

A Fernando, por su amistad inquebrantable y cariño innegable que me sostuvo tanto tiempo;

A todos los gatos con los que vivimos, y en especial a Crepita y Toñita que, antes de morir, me acompañaron pacientemente durante la escritura de esta tesis.

Al planeta, un mundo en profundo sufrimiento, que nos lleva a sus espaldas en el frío y silencioso espacio.

Índice general

Gratiæ agendæ v

Nuncupatio vii

1. Introducción **1**

I. Motivación 1

II. Objetivo y alcances 2

III. Estructura sintética de esta tesis 3

2. Antecedentes **5**

2.1. ¿Qué es la vida? 5

2.1.1. Un poco de historia 6

2.1.2. Biología molecular 7

2.2. Biología relacional 8

2.3. Sistemas de Metabolismo-Reparación 16

2.4. Teoría de la computación y el trabajo de Rosen 20

3. Exploraciones computacionales: tres casos de estudio **29**

3.1. Los *Sistemas-(M,R)* como Máquinas-X (Palmer y cols., 2016) 29

3.2. Los *Sistemas-(M,R)* en álgebras de procesos (Gatherer & Galpin, 2013) 34

3.3. Los *Sistemas-(M,R) (Zhang y cols., 2016)* en UML 39

4. El sistema-(M,R) como diagrama UML **43**

4.1. Orientación a Objetos 43

4.2. Lenguaje Unificado de Modelado (UML) 45

4.2.1. Diagrama de Clase 45

4.2.2. Diagrama de Actividad 48

4.2.3. Diagrama de Comunicación 49

4.2.4. Diagrama de Máquina de Estados 50

4.3. El $Sistema-(M,R)$ como diagrama UML: la traducción de Zhang y cols. (2016) . . . 51
4.3.1. El diagrama de clases de un $Sistema-(M,R)$ . . . 51
4.3.2. El diagrama de actividades de un $Sistema-(M,R)$ . . . 54
4.3.3. Los diagramas de comunicación de un $Sistema-(M,R)$ . . . 55
4.3.4. Los diagramas de máquina de estados de un $Sistema-(M,R)$ 56
4.4. Las Máquinas-X y los $Sistemas-(M,R)$ en UML . . . 60

5. Implementación en Python (Py3K) 63
5.1. Problema . . . 63
5.2. Objetivo de la fase experimental . . . 63
5.3. Hipótesis . . . 64
5.4. Metodología: implementación en Python . . . 64
5.5. Desarrollo . . . 65
5.5.1. Paso 1: Análisis y Programación . . . 65
5.5.2. El programa y sus adecuaciones . . . 67
5.5.3. Paso 3: Simulaciones . . . 76

6. Análisis exploratorio de la implementación 81
6.1. Resultado de las simulaciones . . . 81
6.2. Análisis de resultados y de la implementación . . . 85

7. Conclusiones 91
7.1. Generales sobre esta tesis . . . 91
7.2. Conclusiones . . . 94
7.2.1. Las corrupciones de los Sistemas-(M,R) en el fondo de un programa . . . 94
7.2.2. La hipótesis del continuo Vida-Mente . . . 96
7.3. El estado del arte y trabajo futuro . . . 97

Apéndice A. El Sistema-(M,R) en Python 101
A.1. Clase Biomolecula . . . 101
A.2. Clase Substrato . . . 101
A.3. Clase Enzima . . . 101
A.4. Clase A . . . 102
A.5. Clase B . . . 103
A.6. Clase F . . . 104

A.7. Clase Beta 105
A.8. Clase FPrima 107
A.9. Clase Phi 109
A.10.MAIN para conectar clases y correr simulaciones 112

Apéndice B. Algunos resultados de las simulaciones **119**

Bibliografía **125**
Referencias 125

Índice de figuras

2.1. La taxonomía de la vida. 20

3.1. Gráfica de promedio de simulaciones estocásticas en Bio-PEPA 38

4.1. Ejemplo de una clase en UML 46
4.2. Ejemplo de relaciones en un diagrama de clases en UML 46
4.3. Ejemplo de diagrama de actividades en UML 49
4.4. Ejemplo de diagrama de comunicación en UML 50
4.5. Ejemplo de diagrama de estados en UML 51
4.6. El diagrama de clases de un $Sistema - (M, R)$ 53
4.7. Diagrama de actividad de un $Sistema - (M, R)$ 54
4.8. El diagrama de comunicación de un $Sistema - (M, R)$ 55
4.9. El diagrama de comunicación de un $Sistema - (M, R)$ manipulado topológicamente. 56
4.10. El diagrama total de máquina de estado para un $Sistema - (M, R)$. 58
4.11. Los diagramas de máquinas de estado para un $Sistema - (M, R)$ . . 59
4.12. El diagrama de clases extendido para un $Sistema - (M, R)$ 60
4.13. El diagrama de clases de un $Sistema - (M, R)$ como Máquina-X . . . 62

6.1. Promedio de experimentos usando parámetros de Zhang y cols. (2016) 81
6.2. Promedio de experimentos usando parámetros de Zhang y cols. (2016) añadiendo estocasticidad 82
6.3. Promedio de experimentos usando parámetros de Zhang y cols. (2016) con reusos estocásticos 83
6.4. Promedio de experimentos usando parámetros de Zhang y cols. (2016) usando remoción de enzimas y fallas en catálisis aleatorias. 84
6.5. Promedio de experimentos usando parámetros de Zhang y cols. (2016) con condiciones iniciales aleatorias 85

B.1. Experimento con parámetros de Zhang y cols. (2016) 120
B.2. Experimento usando parámetros de Zhang y cols. (2016) añadiendo estocasticidad 121
B.3. Promedio de experimentos usando parámetros de Zhang y cols. (2016) con producción simultánea de pares de biomoléculas 122
B.4. Experimento con alimentación aleatoria 123

Índice de tablas

3.1. Funciones de transición para una Máquina-X de flujo 32
3.2. Funciones de transición parciales de una Máquina-X de flujo 32
3.3. Relaciones de comunicación en una Máquina-X Comunicante 33

4.1. Relaciones para diagramas de clases en UML 47
4.2. Notación para diagramas de actividades en UML. 48

1 Introducción

I. Motivación

Los sistemas de metabolismo-reparación (o metabolismo-reemplazo), abreviados como $Sistemas - (M, R)$, son un formalismo propuesto por el biólogo-matemático y biofísico Robert Rosen (1934-1998) para modelar la mínima expresión viable de los organismos vivos y que sería un paso esencial para lograr la culminación de uno de los viajes intelectuales más relevantes para Rosen: responder la pregunta sobre qué es la vida. La más conocida descripción de los sistemas-(M,R) se encuentra publicada en el libro Life Itself (Rosen, 1991), aunque el formalismo ya había sido explorado y descrito previamente por Rosen a lo largo de su carrera.

Un sistema-(M,R) está compuesto, a su vez, por dos subsistemas: el metabólico (M) y el de reparación o reemplazo de partes (R). Esta forma de describir organismos destaca que la vida no es en principio un fenómeno material (que tenga una causa material), sino funcional (que tiene una causa eficiente).

Entre sus múltiples áreas de estudio, es de interés para la Ciencia de la Computación explorar modelos de la vida y llevarlos al espacio de lo computable para procurar los progresos en la Vida Artificial e Inteligencia Artificial. Así, la relevancia computacional de los sistemas-(M,R) está en su supuesta incomputabilidad. Según Rosen, al ser los organismos necesariamente estructuras cerradas bajo causas eficientes, estos tendrán modelos incomputables por lo que el mínimo sistema que exhibiere propiedades vivientes tendrá que ser, también, incomputable (Rosen, 1991). Esto tiene implicaciones relevantes no sólo para la Biología y la Filosofía, sino para el estudio de la Vida Artificial. Hasta el momento, una demostración formal sobre la computabilidad o incomputabilidad de los sistemas-(M,R) no ha sido producida fuera de las mismas que Rosen exhibió. No obstante esto, varios grupos de investigadores han promovido la búsqueda de esta demostración mediante enfoques prácticos bien conocidos para los científicos de la computación: la instanciación, en este caso particular, de un

sistema-(M,R) en una computadora clásica. Para progresar en este sentido, el presente trabajo parte de la publicación realizada por Zhang y cols. (2016) en la cual se desarrolló un esfuerzo para lograr traducir el formalismo del sistema-(M,R) a un diagrama UML que, según argumentan los autores, bastaría para identificar al sistema-(M,R) como Turing-computable debido a la compatibilidad de UML con lenguajes formales orientados a objetos. En ese mismo trabajo se evita deliberadamente producir una implementación del diagrama UML resultante temiendo que, haciendo esto, se producirían corrupciones al modelo. Debido a esto, se deja como un *datum* el que tal implementación podría o no ser beneficiosa para el análisis.
En este trabajo se realizará una visita a los elementos que componen al modelo de Rosen, así como a algunos resultados de investigaciones recientes sobre la naturaleza computacional de los sistemas-(M,R). Finalmente, para convertir ese *datum* en un *factum*, se realizará una implementación a partir del diagrama UML producido por Zhang et. al (2016) para descubrir si existen o no corrupciones que alejen al sistema-(M,R) diagramado en UML del objeto de estudio original. El interés de ejecutar este paso de materialización (o instanciación) es el promover el entendimiento del formalismo de Rosen en el ámbito computacional.

II. Objetivo y alcances

Objetivo general

Continuar el trabajo de exploración de los sistemas-(M,R) de Robert Rosen, desde la perspectiva computacional, mediante la implementación del diagrama UML derivado por Zhang y cols. (2016).

Objetivos particulares

Realizar una revisión de la teoría existente en torno a los Sistemas-(M,R) de Robert Rosen y sus implicaciones computacionales, así como de trabajos existentes que se han centrado en implementar o instanciar tales sistemas en diversos *frameworks*. Esto anterior con el objetivo de establecer el marco dentro del cual se realizará el análisis del experimento de implementación.

Realizar una revisión computacional sobre la publicación de Zhang y cols. (2016) —como piedra angular de este trabajo —para proporcionar una implementación que permita realizar un análisis sobre las cualidades de este sistema-(M,R) ya en código.

Alcances

Es importante destacar que, a pesar de la preeminencia biológica de los sistemas-(M,R), el enfoque de este trabajo partirá de la perspectiva computacional (no sólo como técnica y ciencia sino como óptica filosófica), pues el interés en el que se enfoca esta tesis es en el de progresar el entendimiento de los Sistemas-(M,R) con un experimento computacional. Únicamente en la discusión terminal del documento ha de ser contemplado el aspecto transdisciplinario de las consecuencias que de este trabajo emanen y que podrían ser de interés para otras disciplinas.

III. Estructura sintética de esta tesis

El presente trabajo abordará aspectos históricos y recientes sobre el trabajo computacional realizado en torno a los sistemas-(M,R) y sobre la ideación de los mismos para poder destacar el valor de la publicación de Zhang y cols. (2016) sobre el cual se sustentará la argumentación central de esta tesis. Para esto, la estructura sintética de el presente trabajo se delimitará de la siguiente manera:

- El primer capítulo es la presente introducción.
- El segundo capítulo funcionará como una revisión de los antecedentes históricos del problema de definir la vida así como del formalismo propuesto por Robert Rosen, sus definiciones (las generadas por el mismo Rosen así como aquellas desarrolladas por otros autores) así como los alcances que este formalismo tiene dentro y fuera de la teoría de la computación. También se introducirán los conceptos teóricos necesarios para la realización del trabajo que no sean parte del trabajo de Rosen (e.g. tecnicalismos o definiciones de la teoría de la computación). Este capítulo, junto con el tercero, permitirá sentar las bases del marco dentro del cual se realizará el análisis del experimento en el quinto capítulo.
- El tercer capítulo contendrá una revisión de tres diferentes trabajos de investigación desarrollados recientemente que se enfocan en la búsqueda de una prueba experimental sobre la Turing-computabilidad de los sistemas-(M,R). Entre estos trabajos se encontrará aquel que ha inspirado este trabajo y que será el sustento para los demás capítulos.

- El cuarto capítulo estará dedicado al análisis del diagrama UML producido por Zhang y cols. (2016). Aprovecharemos para hacer un brevísimo repaso sobre UML y sus aplicaciones para poder entender los alcances del artículo, así como de la teoría auxiliar necesaria para entender con claridad el desarrollo experimental.

- En el quinto capítulo se exhibirá el experimento: la implementación en Python del diagrama UML derivado por Zhang y cols. (2016).

- El sexto capítulo contendrá el análisis exploratorio sobre la naturaleza de la implementación experimental con respecto al formalismo puro de Robert Rosen.

- Finalmente, en el séptimo capítulo, se harán las conclusiones sobre el trabajo y sus alcances. A la par se realizará también una discusión no sólamente computacional sino transdisciplinaria para entender qué horizontes se abren ante aquellos que deseen continuar la investigación sobre el trabajo de Robert Rosen, una figura eminente en la Biomatemática y Biología Relacional.

2 Antecedentes

2.1. ¿Qué es la vida?

La biología es una ciencia natural cuya multitud de tópicos converge en torno al estudio de la vida. Aunque la biología tiene sus raíces en la antigüedad de oriente y occidente (Magner, 2002), su concreción como una disciplina científica fue dándose con el paso del tiempo y de la mano con el desarrollo de la filosofía occidental. Aristóteles, a quien se estudiará brevemente, realizó un estudio sistemático y organizado sobre aspectos de los organismos que inclusive al día de hoy siguen siendo tratados. No obstante la amplitud del campo de estudio de la biología y de sus logros en las décadas recientes, una pregunta sigue eludiendo a los estudiosos de esta disciplina: **¿qué es la vida?**

No existe una sola definición consensuada de lo que puede o no ser la vida, y los intentos por explicar lo que es la vida suelen ser descriptivos (Madigan, Martinko, y Parker, 2002):

> ¿Cuáles son las propiedades esenciales de la vida? ¿Qué diferencia las células de los objetos inanimados? Nuestro concepto de lo que significa «vivo» está determinado por lo que podemos observar hoy en la Tierra o podemos deducir del registro fósil. Pero de lo que sabemos hoy en biología, podemos identificar varias características que son compartidas por la mayoría de los sistemas vivos [...] (p. 3)

Observar el mundo y describirlo es una característica central para el ser humano, y es por ello que el fenómeno de la vida no ha pasado para nada desapercibido para las personas filósofas más prominentes de occidente, y es por ello que, a pesar de

seguir sin respuesta, la pregunta sobre qué es la vida ha alimentado a varios de los debates más importantes de occidente (o bien, se ha visto abordada por las corrientes de pensamiento que circundaron a esos debates).

2.1.1. Un poco de historia

Vale la pena hablar sobre cómo es que la pregunta ha sido abordada en el largo andar de la ciencia y de la filosofía para entender el porqué detrás del modelo que este trabajo ha de estudiar. Haciendo un breve recorrido histórico se puede observar, por ejemplo, que en la filosofía, medicina y religión de la Antigua Grecia, la definición de la vida sería dependiente del *pneuma*, un flujo de aire necesario para que un sistema material pudiese funcionar y sostener a una consciencia, descripción que sería mayormente vitalista a pesar de su componente materialista. De hecho es posible encontrar en Aristóteles la semilla del posterior vitalismo en su tratamiento de los seres vivos mediante su teoría del hilemorfismo: para Aristóteles, el alma es al cuerpo lo que la forma es a la materia, siendo el alma la causa de una cosa viviente (Rosen, 1991).
Durante la Edad Media, cuando algunas de las preguntas de los griegos eran censuradas, el estudio sobre la vida dependió de la defensa de la existencia y justificación de Dios como una causa primera (o primer motor, en términos aristotélicos), mismo que podía ser percibido (y, en términos de Santo Tomás de Aquino, también demostrado) por los efectos que su propia existencia (Tomás de Aquino, 2001). Para San Agustín de Hipona, por ejemplo, lo contrario a Dios es la ausencia de lo que Dios no decrete o decida, por lo que la vida sería una consecuencia directa de la mera existencia (o esencia) de Dios (*i.e.* para San Agustín, lo existente es lo que emana de Dios y de su naturaleza) (Agustín de Hipona, s.f.). No obstante, durante estos mil años en los que la ciencia pudo parecer dormida y en lo que el estudio de la vida estuviese supeditado a la defensa de Dios, los árabes dominaron a la biología y tradujeron textos de Galeno y de los estudios biológicos de Aristóteles. Estos trabajos serían seminales para lo que se realizaría en el renacimiento.
El renacimiento vio aparecer a la anatomía como la principal expresión de la ciencia observacional, pero no avances en cuanto a la definición de la vida. Durante este tiempo se clasificaron plantas y animales, y se dio inicio al primer paso de la revolución científica que proyectaría la especialización de la historia natural y de otras áreas del conocimiento. Es gracias a esto, y a la recuperación del conocimiento de los clásicos, que futuros pensadores se plantearon, finalmente, definir a lo vivo (Hall, 1994).
La modernidad está severamente marcada por el pensamiento dualista de René Des-

cartes, quien desarrolló una perspectiva mecanicista de la vida, describiéndola no por su función sino por su sustrato (material). Descartes intenta separar la vida animal de la vida racional argumentando que mientras los animales son meros autómatas que responden a su entorno y a los estímulos con los que éste les provee, los seres racionales poseen una naturaleza distinta que depende de la separación de la mente y el cuerpo — «el fantasma en la máquina», diría Gilbert Ryle (Ryle, 2021) (1905/2011) — y cuyas explicaciones sobre los fenómenos biológicos darían paso al reduccionismo en la biología (Rosen, 1991), mientras que Kant construyó una dialéctica entre la teleología del funcionamiento de los organismos y un entendimiento mecanicista sobre sus causas, basándose en la mecánica newtoniana (Goy, 2015). El argumento mecanicista se vio atacado por vitalistas como la filósofa Margaret Cavendish que definió a la vida en términos de grados de movimiento y postuló que la materia misma tendría un grado de conocimiento (Cavendish y Lilley, 1994). La postura vitalista que, además de trascender a los siglos por su origen variopinto, no descansaba sobre una sola idea para definir al requerimiento mínimo para la vida (o fuerza vital universal a todos los seres vivos), fue perdiendo *momentum* por los avances de la química orgánica e inorgánica, la bioquímica y, en general, por la adopción del mecanicismo en casi toda la ciencia normal/convencional (Bedau y Cleland, 2010).

En tiempos recientes (a partir del siglo XX), el cisma entre el mecanicismo y el vitalismo se perdió de vista. Es durante este período contemporáneo en el que decenas de científicos, filósofos y pensadores han intentado definir la vida sin lograr, todavía, un consenso. Varios investigadores han realizado sondeos al respecto a estos intentos de definición, muchos de los cuales no han resonado entre la comunidad científica (Malaterre y Chartier, 2021). Mientras tanto, otras definiciones de vida han sido más estudiadas en campos adjuntos a la biología, como es el caso de la Teoría de la Autopoiesis propuesta por los biólogos (Maturana y Varela, 1980), misma que ha sido explorada en el ámbito de la Ciencia Cognitiva (en particular, el de la cognición de las 4E). Esta definición de vida será auxiliar para explicar las implicaciones computacionales del modelo que este trabajo estudiará.

2.1.2. Biología molecular

A partir del triunfo de la perspectiva mecanicista en las ciencias se dieron a lugar varios intentos por describir los procesos que subyacieren a los fenómenos físicos. Uno de estos fenómenos es el de la herencia de los carácteres. Para mediados del siglo XX, los físicos y los químicos empezaron a interesarse por resolver el enigma sobre

los mecanismos de la herencia. Erwin Schrödinger (Schrödinger, 2021) postuló, por ejemplo, que los fenómenos cuánticos podrían dar lugar a la naturaleza estable pero mutable de los genes. A partir del descubrimiento de la doble hélice estructural del A.D.N. y una fuerte motivación experimental, esta disciplina se establecería como la que estudiaría los mecanismos de comunicación, replicación y regulación celular a nivel molecular. La biología molecular definiría a los organismos vivos como aquellos con capacidad metabólica y de almacenamiento de información hereditaria replicable (Madigan y cols., 2002).

La biología molecular representa un caso relevante de reduccionismo en la biología. Este reduccionismo intentaría describir a todos los organismos vivos a partir de los mecanismos de las interacciones químicas estudiadas por la biología molecular, y representa un caso relevante de reduccionismo en la biología. Este reduccionismo intentaría describir a todos los organismos vivos a partir de los mecanismos de las interacciones moleculares de intercambio de información regidas por las leyes físicas.

2.2. Biología relacional

La biología molecular postula que la estructura material de los organismos define sus funciones y cuya organización puede ser obtenida a partir del estudio analítico de sus partes, deshaciendo cualquier contexto organizativo superior y dejando únicamente al fenómeno fisicoquímico como sujeto de estudio (Rosen, 1991). Este postulado reduccionista se ve confrontado por una escuela de pensamiento eminentemente matemática fundada por Nicolas Rashevsky y posteriormente extendida por Robert Rosen, como se verá más adelante. Rashevsky fundó en la década de 1950 los estudios de la Biología Relacional (Rashevsky, 1954); ésta serviría como una perspectiva que se enfocaría no en los principios materiales de los organismos sino en la organización como originante de los aspectos estructurales de los mismos (Rosen, 1991)[1]. El enfoque de Rashevsky contrastaría diametralmente con el reduccionista al ser principalmente uno algebraico y funcionalista, y no mecanicista y analítico. Además, Rashevsky habría distinguido una suerte de universalidad común a todos los organismos (una homología biológica) y sus comportamientos — desligado de su material formante —, buscando describir

[1]Nicolas Rashevsky es también el padre y fundador de la Biología Matemática, antes Matemática Biofísica, así como el fundador del Boletín de Biología Matemática, antes Boletín de Matemática Biofísica, mismo que hoy sigue siendo publicado. Fundó el primer Comité de Biología Matemática, en la Universidad de Chicago, el cual gestionó el primer programa educativo en el mundo que otorgaría doctorados en Biología Matemática (Shmailov, 2016)

matemáticamente esta universalidad integrada (como un formalismo común). Se puede decir que la perspectiva de Rashevsky sería holista (A. H. Louie, 2019).

A partir de ahora se hace una construcción de los rudimentos de la biología relacional según la descripción de Aloisius Louie (A. H. Louie, 2019).

Axioma 2.2.1 (de especificación) *Para todo conjunto U y un enunciado $p(x)$ (sobre x) existe un conjunto P cuyos elementos son exactamente aquellos $x \in U$ para los cuales el enunciado $p(x)$ es* `TRUE`.

El conjunto es, pues, definido por las propiedades que sus elementos satisfagan. Véase que este conjunto puede escribirse como $P = \{x \in U \mid p(x)\}$. El complemento de este conjunto puede ser escrito como $\bar{P} = \{x \in U \mid \neg p(x)\}$, lo que significa que $\bar{P} = \{x \in U \mid x \notin P\} = U \sim P$. Véase también que si $P = U$ y $\bar{P} = \varnothing$, o bien si $P = \varnothing$ y $\bar{P} = U$, la propiedad $p(x)$ no definirá subconjuntos nuevos en U.
No obstante, para el trabajo de la biología relacional, se operará suponiendo que tanto P como $\bar{P}$ son no-vacíos por la siguiente especificación: el universo será llamado N (el universo de todos los sistemas naturales) y el subconjunto de interés será L (el de los organismos vivos), de tal suerte que $L \subset N$. Esto implica directamente que tanto L como $\bar{L}$ serán ambos no-vacíos (porque hay sistemas naturales vivos y no-vivos) por lo que $\bar{L} = N \sim L$. El enunciado de interés será el que se definirá como $l(x)$: un organismo está vivo (o un sistema natural es un organismo viviente) si y sólo si la propiedad $l(x)$ es `TRUE`. Así pues, se puede escribir:

$$L = \{x \in N \mid l(x)\}$$

Obsérvese ahora que si consideramos el par de conjuntos $\langle L, \bar{L} \rangle$ se define una partición de N de forma que $\forall x : [l(x) \vee \neg l(x)] \wedge [\neg(l(x) \wedge \neg l(x))]$.

Ahora bien, sea $P = \{x \mid p(x)\}$ y $Q = \{x \mid q(x)\}$. Se dice que $P \subset Q$ si y sólo si p es suficiente para q, y también si y sólo si q es necesaria para p. I.e.:

$$P \subset Q \iff \forall x, p(x) \Rightarrow q(x) \tag{2.1}$$

Para el estudio de la vida, se busca acotar al enunciado $l(x)$ que describa precisamente las condiciones necesarias y suficientes para que un sistema natural sea un

organismo vivo. En el caso particular de los organismos, buscar condiciones de suficiencia es más complicado que buscar condiciones de necesidad (e.g. si se remueve la acidez necesaria para las hortensias, éstas mueren). Por lo tanto, si el enunciado $l(x)$ tiene como necesidad a $q(x)$, entonces $L \subset Q$. Si por otro lado $p(x)$ es condición suficiente para $l(x)$, entonces $P \subset L$. Para lograr construir el conjunto que integre las condiciones necesarias (de forma independiente) y suficientes (de forma conjunta) para la vida, se busca un conjunto de condiciones necesarias individuales de la forma:

$$l \Rightarrow q_1, l \Rightarrow q_2, l \Rightarrow q_3, ... = L \subset Q_1, L \subset Q_2, L \subset Q_3, ... \tag{2.2}$$

Para después construir una cadena de intersecciones descendientes de la forma:

$$Q_1 \supset Q_1 \cap Q_2 \supset Q_1 \cap Q_2 \cap Q_3 \supset ... \supset L = \bigcap_{i=1}^{n} Q_i = L \tag{2.3}$$

Esta cadena convergerá idealmente, en el límite, en L, aunque en la práctica podría ser posible tener que restringir la búsqueda a un número finito de superconjuntos para las condiciones necesarias $q_1, q_2, ..., q_n$ (i.e. $Q_1 \supset L, Q_2 \supset L, ..., Q_n \supset L$ de forma que la intersección $L = \bigcap_{i=1}^{n} Q_i$ sea *lo más cercano* a L cuanto se pueda. De ahí que sea posible definir:

$$l = \bigwedge_{i=1}^{n} q_i \tag{2.4}$$

como la **suficiencia conjunta** que todos los sistemas naturales (que son organismos) deben cumplir.

Teorema 2.2.1 (Fundamental de la Biología Relacional) *Un sistema natural N es un organismo si y sólo si se encuentra clausurado bajo causas eficientes.*[2]

Este teorema descansa en el trabajo realizado a lo largo de la vida de Robert Rosen, y muestra la conexión de tres propiedades interesantes cuyas relaciones se exploraron en una publicación de Rosen (Rosen, 1972) así como en sus ensayos publicados póstumamente (Rosen, 2000):

[2]Este teorema también es referido como el Axioma de Rosen (A. H. Louie, 2013), aunque no queda claro el porqué de intercambiar el término de teorema por el de axioma.

$q_1 = x$ es impredicativo

$q_2 = x$ es anticipatorio

$q_3 = x$ es cerrado bajo causas eficientes

Por lo que se puede entender que $l \Rightarrow q_1, l \Rightarrow q_2, l \Rightarrow q_3$. No obstante, existe un orden secuencial de implicaciones, de forma $l \Rightarrow q_3 \Rightarrow q_2 \Rightarrow q_1$, lo cual quiere decir que la propiedad q_3 es lo más cercano que se tiene para L. $L = Q_3$ es otra forma de ver el teorema recién enunciado.

Ahora se dará un boceto sobre las definiciones de anticipación, impredicatividad y clausura bajo causas eficientes, mismas que serán breves y meramente referenciales al encontrarse justo en la frontera del alcance de este trabajo. Estas definiciones y teoremas emanan, también del trabajo de (A. H. Louie, 2019):

Definición 2.2.1 (Anticipación) *Un sistema natural es anticipatorio si y sólamente si:*

1. *Contiene un modelo*[3] *interno impredicativo (modelo de sí mismo y su ambiente), y*

2. *De acuerdo al modelo, el sistema natural ejecuta acciones antecedentes al modelo.*

Axioma 2.2.2 (de la Anticipación) *La vida anticipa (es anticipatoria).*

En torno a la propiedad de impredicatividad, es necesario primero describir qué es ésta en términos de la lógica matemática. Aunque no existe una definición unívoca, se dice que algo es impredicativo si hace referencia a sí mismo, o si la definición de cuantificación de un objeto hace referencia al objeto mismo o a un conjunto que contiene al objeto (Rosen, 1991). Así, definiciones auto-referenciales serán impredicativas, mientras que las predicativas (como las del cálculo de predicados) pueden construir teorías estratificadas. Un buen ejemplo de una construcción impredicativa es la Paradoja de Russell.

[3]Un modelo, para Rosen, contiene todas las estructuras de implicación lógica que coinciden con las de la realización física de la cosa que se modeló (Rosen, 1991, 2000).

En la teoría de Robert Rosen, los sistemas naturales que sean impredicativos se llamarán *sistemas complejos*, mientras que aquellos que no sean impredicativos se llamarán *sistemas simples o mecánicos.*

Teorema 2.2.2 *Un sistema natural que sea anticipatorio debe ser un sistema natural impredicativo, pero no todos los sistemas naturales impredicativos serán sistemas naturales anticipatorios.*

En el texto del que derivan estas definiciones, teoremas y axiomas, Aloisius Louie pone de ejemplo a los virus como sistemas naturales impredicativos que no son anticipatorios porque necesitan a un hospedero para poder hacer uso del sistema anticipatorio que es la célula en sí misma (A. H. Louie, 2019).

Teorema 2.2.3 (de Rosen) *Un organismo debe ser un sistema natural impredicativo, pero no todos los sistemas naturales impredicativos serán organismos.*

Sea $f : A \to B$ un mapeo del conjunto A al conjunto B, i.e. $f \in H(A, B)$, donde $H(A, B) \subset B^A$ es un conjunto de mapeos de A a B. Véase que la caza de elementos $f : a \mapsto b$ tal que $a \in A$ y $b \in B$ puede dibujarse como un diagrama relacional de la forma:

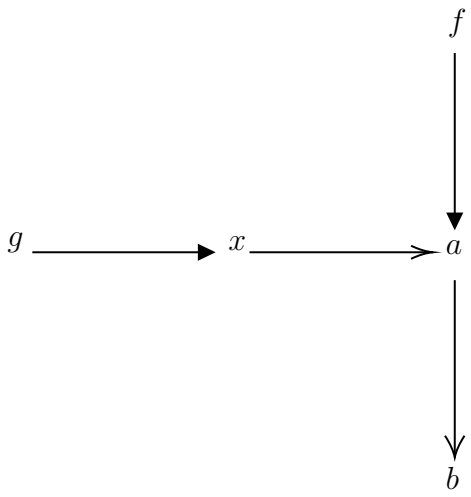
$$f \longrightarrow a \longrightarrow b$$

Que es el diagrama relacional (o de flujo) desde una causa material $a \in A$ hasta la causa final $b \in B$. La flecha rellena indica una inducción de — o una restricción sobre — el flujo por parte de un procesador f, quien representa la causa eficiente. Así, entonces:

$$\langle procesador,\ flujo \rangle\ = f \longrightarrow a \longrightarrow b$$

De forma que en el caso anterior donde las causas están nombradas (a, f, b), se dice que f *implica* b (o bien, que b *es consecuencia de* f), i.e. $f \vdash b$.

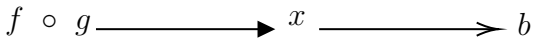
Véase ahora que un diagrama relacional es una red de implicaciones/consecuencias, e.g.:



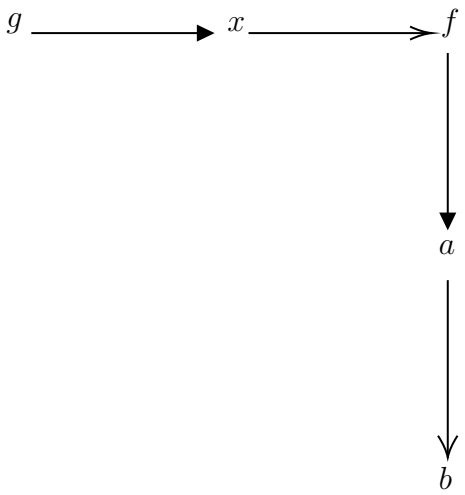
Es el diagrama relacional para la siguiente especificación:

$$f \in H(A, B)$$
$$g \in H(X, A)$$
$$f : a \mapsto b, g : x \mapsto a$$
$$f \vdash b, g \vdash a$$

Por lo que es posible observar que $f \circ g \in H(X, B)$ con $f \circ g : x \mapsto b$ (donde $\circ$ es la composición de funciones). En este caso, la causa final de g es la causa material de f, y se puede representar de la siguiente forma en la que $f \circ g \vdash b$:



Lo cual es llamado una *composición secuencial*. Nótese que cuando aquello que es implicado es después usado como causa material, se dice que tal implicación es una *implicación material*. Ahora véase también el siguiente diagrama relacional:



Que es el diagrama que resulta de la especificación:

$$f \in H(A, B)$$
$$g \in H(X, H(A, B))$$
$$f : a \mapsto b, g : x \mapsto f$$
$$g \vdash f \vdash b$$

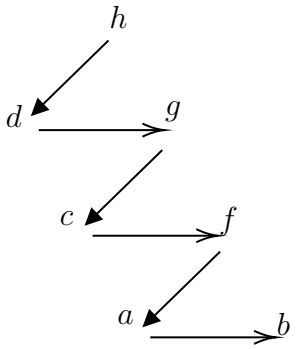
En este caso, la causa final de g es la causa eficiente de f. Cuando aquello que es implicado es una causa eficiente, se dice que se trata de una *implicación funcional*. Esta composición es llamada *composición jerárquica*. A partir de estos dos conceptos sobre los diagramas relacionales se puede hacer una visita a situaciones más interesantes. En particular, aquella que ocurre cuando dos o más composiciones jerárquicas se encuentran en un ciclo.

Definición 2.2.2 (Ciclo de causación eficiente) *Un ciclo de causación eficiente es un ciclo (camino cerrado) de implicaciones causales que contiene dos ó más causas eficientes.*

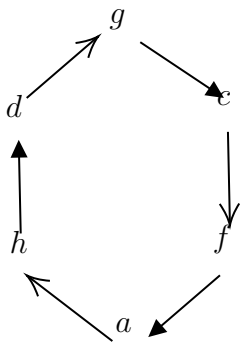
Por ejemplo, téngase en cuenta la siguiente especificación simple:

$$f \in H(A, B)$$
$$g \in H(C, H(A, B))$$
$$h \in H(D, H(C, H(A, B)))$$

Que es representada en el siguiente diagrama relacional:



Supóngase ahora que existe una identificación isomorfa entre b y h, de forma que el diagrama resultante sería:



Cuyas implicaciones pueden ser representadas por la gráfica dirigida $C = \{g \vdash f, h \vdash g, f \vdash h\}$. Por lo tanto, se define:

Definición 2.2.3 *Un ciclo jerárquico es el diagrama relacional construido (usando un grafo dirigido) a partir de un ciclo de implicaciones eficientes.*

Lema 2.2.3.1 *Un sistema natural N tiene un modelo que contiene un ciclo jerárquico si y sólamente si tiene un ciclo de causación eficiente.*

A partir del ciclo jerárquico es posible enunciar algunos postulados de relevancia en la biología relacional al ser, en sí mismos, los bloques de construcción para el modelo que es de interés para este trabajo.

Teorema 2.2.4 *Un sistema natural N es impredicativo (complejo) si y sólamente si tiene un ciclo de causación eficiente. Un sistema natural N será predicativo si y sólamente si no tiene un ciclo de causación eficiente.*

Corolario 2.2.4.1 *Un sistema natural N será impredicativo si y sólamente si tiene un modelo que contenga un ciclo jerárquico.*

Definición 2.2.4 *Un sistema natural N está* ***cerrado bajo causas eficientes*** *si todas sus causas eficientes son internamente implicadas.*

Definición 2.2.5 *Una* ***clausura bajo causas eficientes*** *de un sistema natural N ocurre cuando el sistema natural tiene un modelo en el que todas las causas eficientes en su estructura de implicación causal son componentes de un ciclo jerárquico.*

Definición 2.2.6 (para sistemas formales) *Una* ***clausura bajo causas eficientes*** *de un sistema formal $\langle S, k(S)\rangle$ (donde S es un conjunto equipado con una colección $k(S)$ de mapeos) ocurre cuando el sistema formal tiene todos sus mapeos contenidos en un ciclo jerárquico.*

Esto es el preámbulo necesario, a partir de la biología relacional, para hacer la descripción puntual del modelo que interesa a este trabajo.

2.3. Sistemas de Metabolismo-Reparación

Robert Rosen (1934-1998), estudiante de Nicolas Rashevsky, resume su perspectiva sobre los organismos de manera tajante: los organismos no son máquinas, y no contienen modelos simulables (Rosen, 1991) (para su argumento, Rosen utiliza de manera fundacional a la clausura bajo causas eficientes arriba mencionada). Rosen comentó, en contra del dualismo de René Descartes que reduce a toda la vida animal a autómatas, que lo único que a su parecer era objetivo estudiar eran los ciclos cerrados de causación, mismos que están prohibidos en cualquier mecanismo (Rosen, 1993). Para Rosen, lo relevante en el estudio de los organismos no era el aspecto material, sino la organización:

> El cuerpo humano cambia de forma completa sus materiales en aproximadamente 8 semanas mediante el metabolismo, la replicación y la reparación. No obstante, tú sigues siendo tú, con todos tus recuerdos y personalidad [...] Si la ciencia insiste en cazar partículas, las seguirán a través de un organismo y no notarán el organismo. [4].

Rosen propuso una definición para los organismos (y en general, para todos los sistemas complejos) que pudiese responder la pregunta sobre qué es la vida de manera satisfactoria y universal.

Retomando el trabajo expuesto por Aloisius Louie (A. H. Louie, 2019), se desarrollan los conceptos necesarios para exponer el formalismo de Rosen: los $Sistemas - (M, R)$. Para esto, primero se definirán los dos eminentísimos términos utilizados en el nombre del formalismo.

Definición 2.3.1 (Concepto de Aparato Metabólico) *Un organismo es, en principio, un sistema abierto a causas materiales y debe interactuar con su contexto para poder obtener materiales que le provean sustento. Se dice entonces que, sine quā nōn, un organismo debe contener un aparato metabólico. Este aparato sería, en términos formales y generales, el encargado de la causación material* $f : x \vdash y$.

Definición 2.3.2 (Concepto de Reparación) *Una acción ejecutada para generar otra acción: Rosen define al término reparación dentro de la biología relacional como el proceso jerárquico para el cual la salida de un mapeo es también un mapeo. No debe confundirse con el término utilizado en la biología molecular, cuya fuerza semántica es aquella de la reparación bioquímica de una molécula de ADN. La reparación es equivalente con una causación funcional.*

Definición 2.3.3 (Metabolismo) *El metabolismo es un sistema de entrada-salida conectado como componente a una red. El componente del metabolismo es un sistema formal de la forma* $M_i = \langle A_i, H(A_i, B_i)\rangle$.

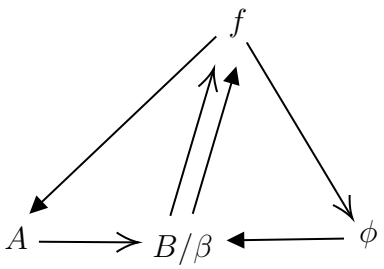
Definición 2.3.4 (Reparación) *La reparación es un sistema de entrada-salida conectado como componente a una red. El componente de la reparación es un sistema formal de la forma* $R_i = \langle Y_i, H(Y_i, H(A_i, B_i))\rangle$.

[4]Dicho por Robert Rosen a su hija, Judith Rosen, según se relató en la ahora difunta página web *Autobiographical Reminiscences of Robert Rosen*, gestionada por Judith Rosen. El dicho se corroboró mediante mensajería privada con Judith Rosen vía la red social Twitter

Definición 2.3.5 (Red de Metabolismo-Reparación) *Una Red-(M,R) es una colección finita de pares de componentes de metabolismo y reparación* $\{(M_i, R_i) \mid i \in I\}$*, conectados en una red. Las salidas del componente de reparación* R_i *son observables en* $H(A_i, B_i)$ *de su correspondiente componente de metabolismo* M_i*. Los componentes de metabolismo podrían o no estar conectados entre sí mediante sus entradas y salidas. Los componentes de reparación deben recibir al menos una entrada que emana desde las salidas de los componentes de metabolismo. Estas conexiones son las requeridas pero podrían no ser las únicas tanto entre componentes como entre el sistema y su contexto.*

Definición 2.3.6 (Sistema de Metabolismo-Reparación) *Un Sistema-(M,R) es una Red-(M,R) que está clausurada bajo causas eficientes.*

Rosen produce en *Life Itself* (Rosen, 1991) el diagrama que representa a la anterior especificación formal donde, nuevamente, flechas rellenas representan causas materiales mientras que las flechas simples representan causas eficientes:



Donde igualmente se pueden interpretar las causas materiales como requerimientos catalíticos para (las transformaciones químicas en) el sistema, mientras que las causas eficientes pueden interpretarse como reacciones productivas (o catalíticas) al interior del sistema. Véase entonces que dentro de este diagrama, todos los componentes catalíticos (ϕ, f, B) son producciones materiales, y todas sus implicaciones causales están internamente contenidas en un ciclo infinito, por ejemplo: si $f : A \mapsto B$, y posteriormente $B : f \mapsto \phi$, y posteriormente $\phi : B \mapsto f$, y se repite ahora el ciclo. De esta forma es fácil ver que las implicaciones dentro de este diagrama relacional serán: $f \vdash B \vdash \phi \vdash f \vdash$ .... En este ciclo, f podría ser una enzima, y ϕ sería el sistema de reemplazo o resíntesis del catalizador f (Rosen, 1991). Algunos autores (Letelier, Soto-Andrade, Guíñez Abarzúa, Cornish-Bowden, y Luz Cárdenas, 2006) han indicado correctamente que no es completamente obvio el porqué de que B sea la causa eficiente del reemplazo enzimático, y han discutido cómo Rosen supone una

propiedad b ó β de los productos metabólicos de B, evitando así una regresión infinita cuya profundización yace fuera del alcance de este trabajo pero que merece ser mencionada para el ojo curioso que note la no-obviedad de la naturaleza causal-eficiente de B, así como por el papel que dicha distinción jugará en capítulos subsecuentes. Para claridad, en este diagrama se incluye también β como una propiedad dual de B.

Sean:

$$m(x) = x \text{ una } Red - (M, R)$$
$$r(x) = x \text{ un } Sistema - (M, R)$$
$$M = \{x \in N : m(x)\} \text{ una } Red - (M, R)$$
$$R = \{x \in N : r(x)\} \text{ un } Sistema - (M, R)$$

Nótese que $M \sim N \neq \varnothing$ porque no todos los sistemas son $Redes - (M, R)$. No obstante Aloisius Louie ha demostrado que un sistema anticipatorio contiene las implicaciones necesarias tales que para cada $i \in I$ se debe tener que $R_i \vdash M_i$ (de una $Red - (M, R)$), por lo que $A \subset M$ (A. H.-Y. Louie, 2009).

Teorema 2.3.1 *Un sistema anticipatorio es una* $Red - (M, R)$.

Teorema 2.3.2 *Un sistema clausurado bajo causas eficientes es un* $Sistema-(M, R)$, *y vice-versa.*

Con estos postulados, es posible dar el paso hacia el resultado más relevante del trabajo de Robert Rosen. Nuevamente, con sustento en lo desarrollado por Aloisius Louie (A. H. Louie, 2019), se exhiben las características de suficiencia que presentan los $Sistemas - (M, R)$ para el problema de la vida.

Definición 2.3.7 (de la célula) *Una célula es, en su expresión mínimamente viable, una estructura material que es la realización*[5] *de un* $Sistema - (M, R)$.

Teorema 2.3.3 *Un* $Sistema - (M, R)$ *es un sistema anticipatorio.*

[5]Para Rosen, la realización física de un organismo se refiere a la decodificiación de una abstracción (o estructura abstracta) acerca del organismo que reasigna propiedades físicas y químicas al organismo en cuestión. (Rosen, 1991)

Es posible en este punto hacerse la pregunta sobre si los $Sistemas - (M, R)$ son suficientes para caracterizar la vida. Rosen argumentó que cualquier sistema material que tuviese como modelo relacional al grafo que describe un $Sistema - (M, R)$ sería un organismo, y que de hecho tal sistema material (como por ejemplo lo es una célula) sería capaz de llevar a cabo la realización material del sistema (Rosen, 1991).

Postulado 2.3.1 (de la Vida) *Un sistema natural N es un organismo si y sólamente si éste es la realización material de un $Sistema - (M, R)$.*

Como consecuencia, y trabajando bajo el marco de Rosen y el álgebra relacional, un $Sistema - (M, R)$ sería el modelo de la vida, y podría responder a la pregunta sobre qué es la vida. En la figura 2.1 se exhibe la taxonomía que incluye diversos sistemas, misma que implica una jerarquía que conduce a los $Sistemas - (M, R)$. Nótese en ella la jerarquía entre la impredicatividad, la anticipación y los modelos de la vida.

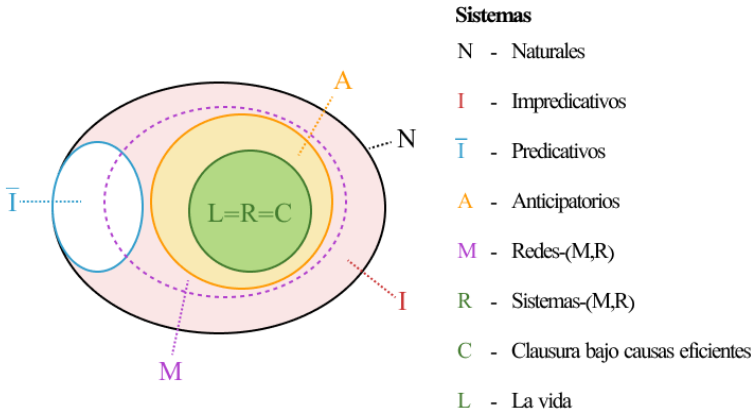


Figura 2.1: La taxonomía de los diversos *systemata*. Adaptada a partir de "Relational Biology" por A.H. Louie, 2019, In: Poli R. (eds) *Handbook of Anticipation*. Springer, Cham., p. 216

2.4. Teoría de la computación y el trabajo de Rosen

Más allá de las consecuencias que tendría el modelo de Rosen sobre los mismísimos fundamentos de la biología, existen aspectos que tocan a la Teoría de la Computación de manera muy relevante para estudios actuales abordados no sólo por la academia sino también por la industria: Rosen argumenta que los $Sistemas - (M, R)$ son incomputables y que esto propone un obstáculo para el desarrollo de la Vida Artificial.

Para tratar este punto, se abordará el concepto de computabilidad de manera constructiva, llevándolo en término al entendimiento de computabilidad de Rosen.

Primero, se considera la definición conceptual de problema (Garey y Johnson, 2009). Un **problema**, en primera instancia, es una pregunta que requiere una respuesta y que puede depender de varios parámetros o variables con valores sin especificar. Un problema se expresará mediante una descripción general de sus parámetros, y mediante un enunciado sobre las propiedades que deben ser satisfechas por la respuesta. Para un problema existirán ejemplares, que se obtendrán especificando valores particulares para los parámetros del problema. Para resolver problemas se pueden plantear algoritmos. Un **algoritmo** es un procedimiento especificado paso a paso de manera finita. Se dice que un algoritmo soluciona un problema Π si el procedimiento es aplicable a cualquier ejemplar de Π y además garantiza que siempre producirá una solución para cualquier ejemplar, es decir, que será un procedimiento efectivo (que produzca algo, en el mismo tenor que la causa eficiente estudiada anteriormente).

Es sensato preguntarse si los algoritmos son procedimientos que podrían ser ejecutados por otro medio que no sea el manual. Esta inquietud nace posiblemente, aunque no únicamente, en la llamada "Tesis de Babbage", atribuida a Charles Babbage por el estudiante y amigo de Turing, Robin Gandy, como el inicio del linaje de las máquinas de cálculo (Gandy, 1988).[6] *Ibidem*, Gandy identificó una variedad de propuestas para construir máquinas calculadoras universales, aunque sin un énfasis en iteración y transferencias condicionales.

A partir de su interés por el *Entscheidungsproblem* de David Hilbert, Alan Turing se hizo la pregunta sobre qué es un cómputo (o, en su tiempo, un "proceso mecánico"). Haciendo esto, Turing analizó a un humano idealizado (*computātor*) para producir una concepción intuitiva sobre "funciones producidas por un procedimiento mecánico". En el mismo trabajo, Turing produjo la descripción de una máquina automática (a la que llamó *A-Machine*) y demostró la equivalencia entre el humano idealizado y su máquina automática. Finalmente, Turing propuso una forma universal de la máquina automática que podría simular a cualquier otra (Turing, 1937)[7] y estableció

[6]La tesis atribuida a Babbage por Gandy rezaría: "La totalidad del desarrollo y operación del análisis son ahora ejecutables por máquinas"

[7]Es en este trabajo monumental Turing demostró la indecidibilidad del *Entscheidungsproblem* y en consecuencia — a la par de Alonzo Church y Kurt Gödel—, que hay problemas que los mecanismos efectivos no pueden resolver en términos decidibles (responder sí o no a una pregunta hecha sobre

al mecanismo que se tiene hoy como modelo estándar de cómputo, conocido ahora como la Máquina de Turing. Con la ayuda de esta máquina abstracta se puede definir ahora, según Robert Soare (Soare, 1996):

Definición 2.4.1 *Una función f será computable o efectivamente calculable si puede ser computada o calculada mediante un procedimiento mecánico finito.*

Definición 2.4.2 *Una función f es Turing-computable si es definible por una Máquina de Turing según la definición de Turing de 1937.*

Ahora se procederá a definir formalmente a un cómputo. Para ello, se toman las siguientes definiciones (Sipser, 2014):

Definición 2.4.3 (Alfabeto) *Un alfabeto es un conjunto finito de símbolos y se denotará con Σ.*

Definición 2.4.4 (Cadena) *Una cadena w es una secuencia finita de símbolos construida sobre (o a partir de) un alfabeto, cuyos símbolos componentes se escriben uno seguido del otro, usualmente sin entrecomas. Una subcadena w' será cualquier cadena que aparezca de manera consecutiva en la cadena w.*

Por ejemplo, para el alfabeto:

$$\Sigma = \{0, 1\}$$

Se tiene la cadena $\alpha = 001101011$. Para una cadena cualquiera β, se define $l(\beta)$ como la **longitud** de la cadena, i.e. la cantidad de símbolos que componen a la cadena. La cadena vacía, es decir la cadena que no contiene ningún símbolo, se escribe como λ.

El conjunto que contiene todas las cadenas que no sean vacías y construibles a partir de Σ se denota con Σ^+, mientras que el conjunto que contiene todas las cadenas — incluyendo la vacía — construibles a partir de Σ se denota con Σ^*.

Definición 2.4.5 (Lenguaje) *Un lenguaje (formal) $\mathcal{L}$ es un conjunto de cadenas construibles sobre algún alfabeto.*

---

los problemas, éstos llamados **problemas de decisión**)(Sipser, 2014)

Por ejemplo, sea $\mathcal{L} = \{\lambda, 01, 0101, 010101, ...\} = \{(01)^n \mid n \in \mathbb{N} \cup \{0\}\}$.

La siguiente definición de un Autómata de Estados Finitos, que es una computadora simple acotada por memoria, es importante para poder definir formalmente al cómputo.

Definición 2.4.6 (Autómata de estados finitos) *Un autómata de estados finitos se forma por una quíntupla $\langle Q, \Sigma, \delta, q_0, F\rangle$ tal que:*

- *Q es el conjunto finito de estados de la máquina,*
- *Σ es un alfabeto finito,*
- *$\delta : Q \times \Sigma \to Q$ es la función de transición,*
- *$q_0 \in Q$ es el estado inicial,*
- *$F \subseteq Q$ es el conjunto de estados de aceptación.*

Definición 2.4.7 (de un Cómputo) *Sea $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F\rangle$ un autómata de estados finitos, y sea $w = w_1, w_2, ..., w_n$ una cadena donde cada w_i es un miembro del alfabeto Σ. Se dice que $\mathcal{M}$ acepta a w si una secuencia de estados $r_0, r_1, ..., r_n$ en Q existe satisfaciendo las siguientes tres condiciones:*

- *$r_0 = q_0$, es decir, el mecanismo inicia la operación en el estado inicial*
- *$\delta(r_i, w_{i+1}) = r_{i+1}$ para $i = 0, ..., n-1$, es decir, el mecanismo transita estados de acuerdo a las funciones de transición definidas para él,*
- *$r_n \in F$, es decir, el mecanismo acepta la entrada si termina en un estado de aceptación.*

Se dice que $\mathcal{M}$ reconoce al lenguaje $\mathcal{L}$ si ocurre que $\mathcal{L} = \{w | \mathcal{M}$ acepta $w\}$

Ahora se describe a la Máquina de Turing, también según Sipser (Sipser, 2014). Primero, debe verse que la Máquina de Turing tiene tres componentes [8]: un dispositivo de control, una cinta de trabajo y una cabeza de lectura y escritura.

El **dispositivo de control** es uno de los estados de un conjunto finito de estados Q. Cuando la máquina arranca, el dispositivo de control será el estado inicial q_0.

[8] Físicamente realizables si se da por hecho que la cinta de trabajo no será infinita.

La **cinta de trabajo** contiene una secuencia (infinita hacia la derecha) de símbolos de un alfabeto Γ. Casi todos los miembros de la secuencia de la cinta, exceptuando a un número finito de ellos, son el símbolo de espacio en blanco $\sqcup$.

La **cabeza de lectura y escritura** lee una celda (un símbolo en la cinta) y puede o no sobreescribirlo según el programa.

Definición 2.4.8 (de la Máquina de Turing) *Una Máquina de Turing $\mathcal{MT}$ es una séptupla $\langle Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r \rangle$ tal que:*

1. *Q es el conjunto finito de estados del programa,*
2. *Σ es el alfabeto finito de entrada que no contiene el símbolo de espacio en blanco $\sqcup$,*
3. *Γ es el alfabeto finito de la cinta de trabajo que sí contiene el símbolo de espacio en blanco, de forma que $\sqcup \in \Gamma$ y $\Sigma \subseteq \Gamma$,*
4. *δ es la función que describe la transición entre estados, de forma que $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow, —\}$ [9],*
5. *$q_s \in Q$ es el estado inicial,*
6. *$q_r \in Q$ es el estado de rechazo, y además $q_a \neq q_r$,*
7. *$q_a \in Q$ es el estado de aceptación.*

La operación de una $\mathcal{MT}$ puede describirse de la siguiente manera:

1. En cada paso de la ejecución del programa de la $\mathcal{MT}$, ésta se encuentra en algún estado q. (En el momento de inicio de la computación se encontrará específicamente en q_0).
2. La $\mathcal{MT}$ leerá un símbolo s de la cinta de trabajo.
3. Para ejecutar un paso de cómputo, la $\mathcal{MT}$ buscará el valor de una función de transición compatible $\delta(q, s)$ el cual será (q', s', m).

[9] A la definición consultada en (Sipser, 2014) se agrega el símbolo — para facilidad conceptual, siendo que una Máquina de Turing que no contenga tal símbolo podrá simularlo moviéndose primero a la izquierda e inmediatamente a la derecha, quedándose la cabeza de lectura y escritura en el mismo lugar.

4. la $\mathcal{MT}$:

   - transferirá el dispositivo de control desde q a q'.
   - utilizará la cabeza de lectura y escritura para sobre-escribir el símbolo s con el símbolo s' en la cinta de trabajo.
   - moverá la cabeza de lectura y escritura a lo largo de la cinta según el movimiento m, que podría ser a la izquierda ($\leftarrow$), a la derecha ($\rightarrow$) o bien quedarse en el sitio (—).

5. La $\mathcal{MT}$ llega a un estado de paro y termina su computación cuando alcanza el estado de aceptación q_a.

La Máquina de Turing captura con precisión la definición de un mecanismo e induce, como se expuso anteriormente, la piedra angular para definir a lo que es y no es computable (o Turing-computable, en honor a Alan Turing) a partir de manipulaciones simbólicas. Esta naturaleza mecanicista fue abordada por Rosen para describir a su formalismo, razón por la cual Rosen produce una serie de definiciones relevantes para este trabajo en torno a las implicaciones de los mecanismos en la definición de la vida (Rosen, 1991):

Definición 2.4.9 *Un sistema natural N es un mecanismo si y sólamente si* ***todos*** *sus modelos son simulables.*

Ahí mismo — y en *Essays on Life Itself* (Rosen, 2000), Robert Rosen define al término *simulable* como aquello que puede ser expresado mediante un algoritmo (y como algo que refleja los comportamientos, entradas y salidas de un objeto o fenómeno, pero que no tiene las causalidades internas al ser éstas meras aproximaciones ocurriendo en la caja negra que es el programa, producto no de la naturaleza sino de la creatividad del programador). Posteriormente, Rosen intercambia el término *simulable* por *computable, efectivo* y por *evaluable por una Máquina Matemática (de Turing)*. En el capítulo 8 del mismo texto, Rosen produce una demostración — mediante una *reductio ad absurdum* — sobre la imposibilidad de que ciertos modelos de causación sean comprendidos por un mecanismo. En particular, su demostración probó el siguiente teorema:

Teorema 2.4.1 *No puede existir un camino cerrado de causación eficiente en un mecanismo.*

Teorema 2.4.2 *(Contrapositiva) Si un sistema natural N tiene un camino cerrado de causación eficiente, entonces N no puede ser un mecanismo.*

Y considerando la definición que dio Rosen sobre los mecanismos y su relación con la simulabilidad, el mismo Rosen deriva:

Teorema 2.4.3 *Si existe un camino cerrado de causación eficiente para un sistema natural N entonces tal sistema tiene (al menos) un modelo que no es simulable.*

Como se mencionó en la definición 2.2.3 y en el 2.2.3.1, un camino cerrado de implicaciones eficientes debe formar un ciclo jerárquico. A su vez, en el teorema 2.2.4 se rescata el hecho de que un sistema natural N será impredicativo si y sólamente si tiene un ciclo de causación eficiente. Rosen abunda en torno a la impredicatividad: ésta es la imposibilidad de reemplazar estos ciclos jerárquicos con algoritmos finitos (y sintácticos) (Rosen, 1991). Como consecuencia, es posible ver ahora que para el teorema 2.4.3 existe un ciclo jerárquico (el camino cerrado de causación eficiente) por lo que tal sistema será un sistema formal equipado con un ciclo impredicativo de implicaciones. Así, en consecuencia, Rosen prueba que:

Teorema 2.4.4 *Si un ciclo impredicativo de implicaciones existe en un sistema formal, entonces ese sistema formal no será simulable.*

Rosen demuestra en su *opus magnum* el teorema que ha puesto en el ojo de la ciencia de la computación su trabajo y su investigación, mismo que reza (Rosen, 1991):

Teorema 2.4.5 *Un sistema natural N que sea un organismo no tendrá modelos simulables.*

Rosen intenta hacer al lector despojarse de la metáfora de la máquina extendida en la ciencia por el dualismo de Descartes, Newton y el programa de Hilbert. Relevantemente para las Ciencias de la Computación, este teorema es utilizado por Rosen (Rosen, 1991, 2000) para concluir que la vida es no computable en cualquier modalidad: natural o artificial. Así también, Rosen indica que la vida artificial debería poseer los mismos patrones causales que la vida organísmica (o en términos de Rosen, la vida artificial no debería ser una simulación sino un modelo de la vida) y que los intentos por construir artificialmente a la vida se verían obstaculizados por la naturaleza de los mecanismos y sus aproximaciones (Rosen, 2000):

> Sobre este fundamento, podemos ver que la fabricación de algo (e.g., un organismo) es muy diferente de aquello que simula sus comportamientos. [...] La idea fue que, dotando serialmente a la máquina con más y más *simulacra* de la vida entonces cruzaríamos el límite más allá del cual una máquina se convertiría en un organismo. El mismo razonamiento está representado hoy en la inteligencia artificial, y es articulado por la Prueba de Turing. Esta actividad es una forma sofisticada del ajuste de curvas, parecido a la aseveración de que, dado que cierta curva es aproximable por cierto polinomio, entonces tal curva debe ser un polinomio.
> En conclusión, cualquier realización material de los $Sistemas-(M,R)$ debe tener modelos no computables. Por lo tanto no pueden ser mecanismos. (p. 269)

Puede verse entonces que los $Sistemas-(M,R)$ representan un problema para disciplinas que intentan artificializar la vida (o la mente, si uno considera la hipótesis del continuo vida-mente de Michael Kirchhoff y Tom Froese (Kirchhoff y Froese, 2017)). Debido a que es de interés para la Ciencia de la Computación explorar a un modelo de la vida y llevarlo al espacio de lo computable para procurar los progresos en la Vida Artificial e Inteligencia Artificial, las investigaciones computacionales en torno a los $Sistemas-(M,R)$ no han sido raras. Construir una expresión computable del formalismo de Rosen ha sido explorado con anterioridad en el ámbito estrictamente matemático (Goertzel, 2002; Mossio, Longo, y Stewart, 2009) pero en general estos intentos han sido desacreditados por Louie porque, según argumenta, en la reducción se hacen omisiones a componentes de la estructura de causación eficiente (A. H. Louie, 2011).

Para los fines de este trabajo recepcional se revisarán a continuación tres publicaciones en el ámbito de la computación que han promovido una búsqueda (experimental) de reducción de los $Sistemas-(M,R)$ a una simulación en una computadora. Dos de ellas preceden a la que es relevante para este trabajo de tesis y constituyen un buen puente para entender cómo es que diversos investigadores han aproximado al problema.

3 Exploraciones computacionales: tres casos de estudio

3.1. Los *Sistemas-(M,R)* como Máquinas-X (Palmer y cols., 2016)

La primer publicación que se estudiará es aquella producida por Palmer y cols. en la que se trabaja a los $Sistemas - (M, R)$ mediante el uso de Máquinas (i.e. Autómatas) de Estados Finitos y Máquinas-X (Palmer, Williams, y Gatherer, 2016). El contenido de la presente sección se toma de esa misma publicación a menos que se indique lo contrario.
Los autores comienzan la publicación haciendo un recuento sobre la metáfora de la máquina y niveles de especificación encontrados en la Biología de Sistemas, concediendo que Rosen dedicó su vida a cuestionar a los reduccionismos y a los mecanismos en la Biología. Posteriormente se explica brevemente el formalismo de Rosen y los progresos que Aloisius Louie ha producido en torno a la matemática de los $Sistemas - (M, R)$ y una conclusión relevante:

> Los procesos individuales dentro de (M, R) son computables en tiempo finito, pero cuando se ensamblan, la auto-referencia es inevitable y la totalidad de (M, R) deja de ser computable. La irreductibilidad de (M, R) a componentes computables de software asemeja a la irreductibilidad de la vida a sub-procesos mecanísticos. (p. 98)

Los autores hacen mención sobre algunos otros trabajos computacionales realizados sobre el formalismo de los $Sistemas - (M, R)$ (como por ejemplo el realizado

en Álgebras de Procesos (Gatherer y Galpin, 2013), que se revisará posteriormente) y que han sido considerados por Aloisius Louie como meras simulaciones. Esto inspiró a los autores a desarrollar una aproximación computacional mediante el uso de Máquinas-X comunicativas con el fin de circunvenir las restricciones de la autoreferencia. Primero, Palmer y cols. extienden la función de transición[1] exhibida en 2.4.6 agregando:

- $T = \{(T_i)_{i=1,\dots,H}, Q, \Sigma\}$
- $q \in Q$
- $\sigma \in \Sigma$

En donde $T_H(q_{H-1}, \sigma)$ es la función de transición final en una serie de $H -$ *transiciones* de estado al final de las cuales la máquina se encuentra en un estado F equivalente a q_H.

Definición 3.1.1 (De una X-máquina de flujo) *Una Máquina-X será la óctupla/el octeto formado por* $\langle \Sigma, \Gamma, Q, M, q_0, m_0, T, P \rangle$ *donde:*

- Σ *es el alfabeto finito de símbolos de entrada,*
- Γ *es el alfabeto finito de símbolos de salida,*
- Q *es el conjunto finito de estados del programa,*
- M *es el conjunto finito de estados de memoria,*
- $q_0 \in Q$ *y* $m_0 \in M$ *son el estado inicial del programa y el estado inicial de memoria, respectivamente,*
- T *es el tipo de Máquina-X definido como un conjunto de funciones parciales del tipo* $T : M \times \Sigma \to M \times \Gamma$*,*
- P *es la función de transición parcial* $P : Q \times T \to Q$

Donde se puede ver que la definición de la Máquina-X extiende la de un Autómata Finito mediante la adición de una memoria almacenada M y un alfabeto de salida Γ. Para permitir la comunicación entre diferentes Máquinas-X (que servirá a los autores para explorar a los $Sistemas - (M, R)$) se agrega la relación de comunicación R de forma $((C_i^X)_{i=1,\dots,n}, R)$ donde:

[1] Palmer y cols. cambian el símbolo de δ por T para nombrar a la función de transición.

- C_i^X es la $i-$ésima Máquina-X,
- R es la relación de comunicación entre $n-$Máquinas-X, y donde R se puede expresar como una matriz de $(i,j)-$celdas donde cada una define una comunicación entre las $i-$ésima y $j-$ésima Máquinas-X.

Con estas definiciones, los autores realizan una traducción de un $Sistema-(M,R)$ y, considerando a b como el componente catalítico de B y a f' como el componente catalítico de f, especifican:

- Entrada: $\Sigma = \{n, f', \phi\}$, los catalizadores,
- Estados del sistema: $Q = \{A, B, b, f, f', \phi\}$,
- Estado inicial: $q_0 = \{A\}$,
- Estados de aceptación: $F = \{b, f', \phi\}$ productos metabólicos pero jamás substratos,
- Funciones de transición elegidas de manera estocástica:
  - $T_1^B = \{T : A \times f' \to B\}$
  - $T_1^b = \{T : A \times f' \to b\}$
  - $T_2^f = \{T : B \times \phi \to f\}$
  - $T_2^{f'} = \{T : B \times \phi \to f'\}$
  - $T_3^\phi = \{T : f \times b \to \phi\}$

Los autores identifican que una dificultad al usar una Máquina de Estados Finitos sobre un $Sistema-(M,R)$, además de la aleatoreidad de la elección de transiciones, es el hecho de que las funciones de transición $T : Q \times \Sigma \to Q$ implican que las señales de entrada de Σ serán externas al sistema mismo, mientras que en un $Sistema-(M,R)$ estas señales deben ser internas (estados del sistema mismo), causadas por la misma estructura, y por lo tanto auto-referenciales. Con esto, los autores coinciden con Rosen en que estos mecanismos no pueden ser un $Sistema-(M,R)$, ni *vice-versa*. Para los autores, las Máquinas de Estados Finitos pueden describir, en el mejor caso, subsistemas dentro de un $Sistema-(M,R)$.

Inmediatamente después se hace la extensión de la anterior descripción del $Sistema-(M,R)$ hacia una Máquina-X de flujo. Los autores indican que no ven el beneficio

inmediato en hacer esta extensión, aunque el uso de memoria podría ayudar si se permiten $H-$reúsos de cada catalizador:

- Entrada: $\Sigma = \{b_0, ..., b_{H-1}, f'_0, ..., f'_{H-1}, \phi_0, ..., \phi_{H-1}\}$
- Los estados del sistema: $Q = \{A, B, b_0, ..., b_{H-1}, g, f'_0, ..., f'_{H-1}, \phi_0, ..., \phi_{H-1}, \Omega\}$ donde Ω representa el estado donde las $H-$iteraciones han terminado,
- El alfabeto de salida: $\Gamma = \{b_1, ..., b_{H-1}, f'_1, ..., f'_{H-1}, \Omega\}$
- Estados de aceptación: $F = \{\Omega\}$

Mediante la tabla 3.1 se especifican las funciones de transición T (reúso en $H-$ocasiones de un catalizador), donde las filas M definen los estados de memoria desde $n = 0$ hasta H, mientras que las columnas Σ definen las entradas desde $n = 0$ hasta $H - 1$. Cada valor en la tabla define a la salida de la función y al nuevo estado de memoria. Por otro lado, mediante la tabla 3.2, se especifican las funciones parciales de transición P, donde las filas T definen las funciones de transición que ocurren desde $x = 1$ hasta $x = H - 1$, mientras que las columnas Q son los estados del sistema. Cada valor en la tabla define el siguiente estado del sistema, mientras que las celdas vacías representan combinaciones no-válidas..

		Σ		
		b_n	f'_n	ϕ_n
	0	$b_1 + M_1$	$f'_1 + M_1$	$\phi_1 + M_1$
M	n	$b_{n+1} + M_{n+1}$	$f'_{n+1} + M_{n+1}$	$\phi_{n+1} + M_{n+1}$
	...	...	...	...
	H	$\Omega + M_0$	$\Omega + M_0$	$\Omega + M_0$

Tabla 3.1: Las funciones de transición de una Máquina-X de flujo. Adaptada a partir de "Rosen's (M,R) system as an X-machine" por M.L. Palmer, R.A. Williams, & D. Gatherer, 2016, *Journal of Theoretical Biology*, 408, p. 101.

					Q		
		A	B	$b_0, ..., b_{H-1}$	f	$f'_0, ..., f'_{H-1}$	$\phi_0, ..., \phi_{H-1}$
	$b_x M_x$				ϕ		
T	$f'_x M_x$	B/b					
	$\phi_x M_x$		f/f'				

Tabla 3.2: Las funciones de transición parciales de una Máquina-X de flujo. Adaptada a partir de "Rosen's (M,R) system as an X-machine" por M.L. Palmer, R.A. Williams, & D. Gatherer, 2016, *Journal of Theoretical Biology*, 408, p. 101.

A partir de esta construcción como Máquina-X de flujo, los autores realizan la extensión a una Máquina-X comunicante donde cada Máquina-X de flujo será un subsistema. La tabla 3.3 muestra las normas de comunicación para una Máquina-X Comunicante sobre un $Sistema - (M, R)$ donde cada valor en la tabla representa los estados de sistema de la $i-$ésima y $j-$ésima Máquinas-X de flujo post-interacción, mientras que las celdas vacías representan una no-interacción diseñada.

				i			
		A	B	b_x	f	f'_x	ϕ_x
	A					$B/b + f'_{x+1}$	
	B						$f/f' + \phi_{x-1}$
j	b_x				$\phi + b_{x+1}$		
	f			$\phi + b_{x+1}$			
	f'_x	$B/b + f'_{x+1}$					
	ϕ_x		$f/f' + \phi_{x+1}$				

Tabla 3.3: Las R relaciones de comunicación entre las $i-$ésimas y $j-$ésimas Máquinas-X de flujo. Adaptada a partir de "Rosen's (M,R) system as an X-machine" por M.L. Palmer, R.A. Williams, & D. Gatherer, 2016, *Journal of Theoretical Biology*, 408, p. 101.

Ahora ocurre que las implicaciones o consecuencias lógicas, que eran externas en las Máquinas-X de flujo y en las Máquinas de Estado Finito, son ahora internas a la Máquina-X comunicante, aunque son externos a cada subsistema, careciendo de auto-referencia. En esta especificación también se considera que un valor de memoria x puede incrementar arbitrariamente, pero cuando ocurra que $x = H$ entonces el sistema caerá al estado Ω, la condición de paro.

Los autores, finalmente, terminan su experimento haciendo una traducción de la especificación de la Máquina-X Comunicante a un diagrama UML, similar al que se explorará en la tercer sección de este capítulo. De hecho, dos de los autores de la publicación que ahora se revisa participaron en la publicación referente a UML y que será sujeto de estudio en la tercer sección de este capítulo, mientras que la aproximación a las Máquinas-X Comunicantes mediante UML será mencionada en la coda del capítulo siguiente.

En la discusión, los autores indican que la refutación mecanicista que nace de los $Sistemas - (M, R)$ depende de que tal sistema sea una sola máquina, y que tal máquina realice su procesamiento de manera secuencial. A partir de esto, Palmer y cols. indican que el paradigma de las Máquinas-X Comunicantes es capaz (o

mejor dicho, deshacerse de) el obstáculo que impone la auto-referencia nata a los $Sistemas - (M, R)$. No obstante esto, se indica que para una Máquina-X Comunicante con $n = 100$ y $H = 3$ (donde n es el número de Máquinas-X en la relación de comunicación), ocurre que el espacio de estados sería $Q = 10^{26}$ (una máquina de este tipo corriendo en un procesador a 10^10 FLOPS tomaría 3.17×10^8 años en recorrer todas las posibilidades de la máquina), por lo que se presentaría como necesidad el usar procesamiento paralelo (algo que, indican los autores, de hecho va bien con la naturaleza de los sistemas vivos). Así, los autores argumentan que la perspectiva roseneana sobre la "no-mecanicidad"de los $Sistemas - (M, R)$ en realidad emerge de una definición pobre de lo que es una máquina, siendo que las Máquinas-X Comunicantes lidian con el problema de la auto-referencialidad. Antes de cerrar, los autores admiten que tuvieron que hacer ciertas concesiones sobre el diagrama de Rosen al suponer que A, B, F y ϕ no son descripciones generales de metabolismos sino moléculas (mientras que las flechas del diagrama indicarían eventos que tendrían como objeto directo a cada molécula), así como al inducir un componente estocástico en la elección de las transiciones. Los autores concluyen declarando a Rosen vencido, indicando que la Biología de Sistemas podría ser tanto mecanicista como reduccionista.

3.2. Los *Sistemas-(M,R)* en álgebras de procesos (Gatherer & Galpin, 2013)

La segunda publicación que se estudiará, producida por Gatherer & Galpin, enfoca sus esfuerzos en instanciar a un $Sistema - (M, R)$ en Bio-PEPA, un álgebra de procesos utilizada para modelar y analizar redes bioquímicas (Gatherer y Galpin, 2013). Este trabajo es, posiblemente, el menos teórico en términos computacionales pero el que es más cercano a la realización computacional de un $Sistema - (M, R)$. El contenido de la presente sección se toma de esa misma publicación a menos que se indique lo contrario.

Gatherer & Galpin inician describiendo las diferencias conceptuales entre la Biología de Sistemas y la Biología Relacional, repasando de manera somera al formalismo de la Máquina de Turing y de lo que es o no es Turing-computable. Posteriormente hacen una revisión igualmente somera sobre el formalismo de Rosen a partir de los

diagramas de Louie-Kercel (A. H. Louie y Kercel, 2007) y de Goudsmit (Goudsmit, 2007) construidos a partir del original de Rosen, y rescatan la propuesta incomputabilidad de los $Sistemas - (M, R)$ debido a la clausura bajo causas eficientes. Gatherer & Galpin revisan el concepto de modelo y relación de modelado (lograda cuando las estructuras causales del modelo y del objeto en el mundo real son idénticas), esgrimidas por Rosen y por la Biología Relacional, así como el concepto de simulación que se describió en el capítulo anterior. En suma, dicen los autores, los biólogos relacionales ven al fenómeno de la vida como uno que es no-mecanicista, no-reductible, y no-computable.

Con el preámbulo hecho, los autores realizan una visita más profunda al formalismo de Rosen a partir de los diagramas, y deciden suponer que, en el $Sistema - (M, R)$ que han de llevar a Bio-PEPA, A, B, f, y ϕ son entidades individuales, contrario a la interpretación general de Rosen sobre conjuntos.

En la publicación se hace mención a una tabla que relaciona anteriores intentos de simular a los $Sistemas - (M, R)$, incluyendo trabajos realizados sobre Sistemas Autopoiéticos (Varela, Maturana, y Uribe, 1974) y sobre ejemplos completamente consistentes con un $Sistema - (M, R)$ hechos en MatLab (Piedrafita, Montero, Morán, Cárdenas, y Cornish-Bowden, 2010). Es importante destacar que, como los autores mencionan, fue la publicación de Varela y cols. la que dio cabida a la notoriedad de la importancia computacional de los $Sistemas - (M, R)$ una vez que Letelier y cols. demostraron que los sistemas autopoiéticos son una sub-clase de los $Sistemas - (M, R)$ (Letelier, Marı´n, y Mpodozis, 2003). La inspiración para el trabajo de los autores en esta publicación dimana, indican, del trabajo de Varelta y cols. de 1974.

El basamento tecnológico utilizado por Gatherer & Galpin (2013) para construir su experimento, Bio-PEPA, se describe según la página oficial de dicha Álgebra de Procesos (Ciocchetta y Hillston, 2010):

> Bio-PEPA es un lenguaje para el modelado y análisis de *redes bioquímicas*. Está basado en PEPA, un álgebra de procesos originalmente definida para la realización de análisis de sistemas computacionales, y le extiende para poder manejar algunas características de redes bioquímicas, tales como la *estequiometría* y diferentes tipos de *leyes kinéticas*. Una característica principal de Bio-PEPA es la posibilidad de soportar diferentes tipos de

análisis, incluyendo simulaciones estocásticas, análisis basados en ecuaciones diferenciales ordinarias (EDOs) y revisión de modelos en PRISM.

Gatherer & Galpin expresan a una especie molecular[2] en Bio-PEPA de la siguiente manera:

```
S=(a1,k1) op1 S + ... + (an,kn) opn S
```

Donde se definen:

- `a1,...,an` son los nombres de las reacciones,
- `k1,...,kn` son los coeficientes estequiométricos
- `op1,...,opn` son cualquiera de:
  - $<<$ que indica el rol de S como un producto,
  - $>>$ que indica el rol de S como un reactivo/reactante,
  - $+$ que indica el rol de S como un catalizador,
  - $-$ que indica el rol de S como un inhibidor.

Estas especies pueden combinarse en un modelo:

$$\texttt{S1[x1]} < * > \ldots < * > \texttt{Sp[xp]}$$

Donde:

- Hay $p-$especies desde `S1` hasta `Sp`,
- Las concentraciones (o contadores) de moléculas asociados a tales especies van de `x1` a `xp`.

Para llevar a un $Sistema - (M, R)$ a Bio-PEPA, los autores primero identifican las tres reacciones enzimáticas del modelo de Rosen, expresadas de la forma $Sustrato \xrightarrow{Enzima} Producto$ y a partir de ello exponen la construcción de la reacción $B \xrightarrow{p} f$ donde $p = \phi$, la cual transforman en especies moleculares en sus formas admisibles por Bio-PEPA:

[2]Según la IUPAC, una especie molecular o química es un ensamblaje de entidades moleculares idénticas que exploran los mismos niveles de energía en una escala temporal determinada o característica. Tales especies pueden ser átomos, moléculas, radicales o iones. (Gold, 2019)

- `B = (r_l1,1) << B + (r_l2,1) >> B`
- `p = (r_l1,1) << p + (r_l2,1) >> p + (r_l3,1) >> p`
- `f = (r_l3,1) >> f`
- `pB = (r_l1,1) >> pB + (r_l2,1) << pB + (r_l3,1) << pB`

Mismas que corresponden con las reacciones enzimáticas contenidas en el diagrama de Rosen, expresables como:

- $B + p \rightarrow pB$ a velocidad l_1 (el substrato se unen),
- $pB \rightarrow p + B$ a velocidad l_2 (el substrato y la enzima se desacoplan),
- $pB \rightarrow p + f$ a velocidad l_3 (el producto es creado y la enzima es liberada internamente al diagrama/red de reacciones)

Se especifica posteriormente que las velocidades `r_l1, r_l2` y `r_l3` serán definidas, usando la Ley de Masas[3], por:

- `kineticLawOf r_l1: l1 * p * B`, con `l1` razón constante de la Ley de Masas,
- `kinetixLawOf r_l2: l2 * pB`, con `l2` razón constante de la Ley de Masas,
- `kineticLawOf r_l3: l3 * pB` con `l3` razón constante de la Ley de Masas,

De manera análoga se realizó este trabajo para las reacciones enzimáticas restantes (a saber: $f \xrightarrow{b} p$ y $A \xrightarrow{f} B$) representadas en el diagrama del $Sistema - (M, R)$, produciendo el siguiente conjunto de instrucciones que detallan al estado inicial del sistema en Bio-PEPA:

- `A[A_init]` $<*>$
- `fA[fA_init]` $<*>$ `B[B_init]` $<*>$
- `pB[pB_init]` $<*>$ `f[f_init]` $<*>$
- `Bf[Bf_init]` $<*>$ `p[p_init]`

Estado inicial: `B [B_init]` $<*>$ `p[p_init]` $<*>$ `f[0]` $<*>$ `pB[0]`

[3]Según definen Érdi & Tóth, la Ley de Masas indica que para una reacción en equilibrio, la razón entre concentraciones de reactivos y productos será constante (Érdi y Tóth, 1989).

Los autores muestran, entonces, los resultados de su simulación estocástica. En algunos casos el sistema muere poco tiempo después del inicio, y en otros logra estabilidad, lo que los autores asocian con la fragilidad del sistema con respecto a los parámetros de entrada, en particular con respecto al de la velocidad de la reacción.

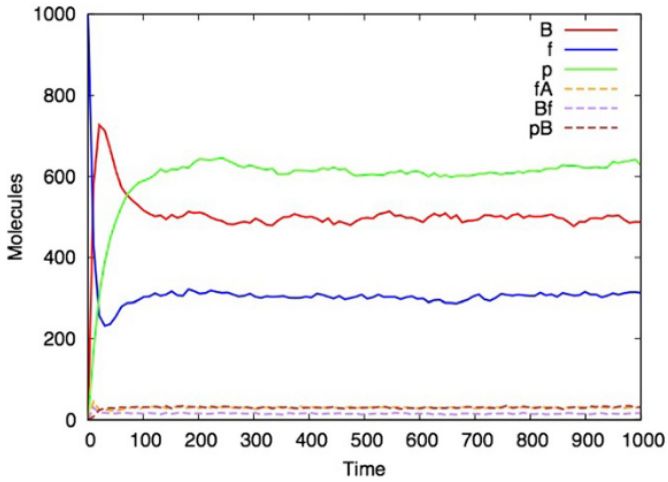


Figura 3.1: Gráfica para el promedio de diez simulaciones estocásticas del $Sistema - (M, R)$ en Bio-PEPA. Reproducido de "Rosen's (M,R) system in process algebra" por D. Gatherer & V. Galpin, 2013, *BMC Systems Biology*, 7:128, p. 5.

En la discusión, los autores indican que, dado que la simulación fue realizada crudamente a partir del diagrama de Rosen (y no a partir de la formulación matemática), podría pensarse que los resultados estables de la simulación prueban que ésta es una realización computacional de un $Sistema - (M, R)$, aunque no admiten esa fuerte aseveración de manera sencilla, en particular a partir de las objeciones de Aloisius Louie a los intentos por llevar a un $Sistema - (M, R)$ a una computadora por ser esto una mera simulación (en los términos estudiados en el capítulo anterior) y no tanto así una realización del modelo. Igualmente los autores exploran la relación estrecha entre los Sistemas Autopoiéticos y los $Sistemas - (M, R)$ (que contienen a los anteriores), en tanto que las estructuras causales de los Sistemas Autopoiéticos computarizados no coinciden con aquellas de los $Sistemas - (M, R)$. Para los autores, la simulación realizada es satisfactoria al comprender todas las relaciones causales productivas que Rosen diseñó. Las críticas que presuponen que pueden hacerse a su trabajo descansan primero en los métodos de establecimiento de condiciones iniciales en la versión determinista de los experimentos, así como en el muestreo aleatorio de la estructura causal completa realizado en la versión estocástica; segundo, el hecho de haber codificado el desarrollo del sistema a lo largo del experimento mediante estados (como, según

indican los autores, ocurre con todas las simulaciones computacionales existentes para un $Sistema-(M,R)$), concepto que Rosen prohíbe en su argumentación en contra de la metáfora de la máquina y por su definición de mecanismos. Gatherer & Galpin se acercan, entonces, al argumento de la computabilidad. Primero destacan que es posible que la noción de computabilidad que Rosen utilizó haya sido inexacta, al requerir Rosen que la definición dependa de una garantía de paro en la computación, cuando la computabilidad de una función (parcial), indican los autores, no requiere paro en una Máquina de Turing que trabaje con entradas no definidas para la función (como lo hacen las Máquinas de Turing Parciales). Los autores exploran, en segundo lugar, el hecho de que es posible transformar conjuntos impredicativos en conjuntos predicativos, razón por la cual argumentan que la impredicatividad — esgrimida por Rosen como garante de la incomputabilidad de su formalismo — podría no ser un obstáculo sino una vía para demostrar la Turing-computabilidad de los $Sistemas-(M,R)$. Inmediatamente después se hace mención sobre un intento, ejecutado por Mossio y cols. por llevar a los $Sistemas-(M,R)$ al Cálculo$-\lambda$ (Mossio y cols., 2009), el cual fue criticado por Cárdenas y cols. debido a que Mossio y cols. no hicieron la distinción adecuada entre B y β (Luz-Cárdenas, Letelier, Gutierrez, Cornish-Bowden, y Soto-Andrade, 2010), misma que Gatherer & Galpin admiten haber omitido igualmente en el trabajo que ahora se revisa. En la coda de su discusión, los autores exhiben caminos para buscar la estructura topológica de los $Sistemas-(M,R)$ en bases de datos biológicos producto de la Biología de Sistemas. Finalmente, Gatherer & Galpin concluyen que las dinámicas «aparentes-de-vida» (sic) de su simulación les permiten concluir que la Biología Relacional es computacionalmente posible.

3.3. Los *Sistemas-(M,R) (Zhang y cols., 2016)* en UML

El último trabajo a revisar, elaborado por Zhang y cols., es uno que se encuentra cercano al corazón de la Programación Orientada a Objetos (P.O.O.), un paradigma de programación que será brevemente descrito en el capítulo subsecuente y que es estudiado en la academia y en la industria, siendo adaptado por diversos lenguajes de programación, incluyendo aquel que se utilizará en el experimento de esta tesis: Python (Zhang y cols., 2016). El contenido de la presente sección se toma de la misma publicación a menos que se indique lo contrario.

Los autores realizan, al igual que aquellos que ya han sido estudiados en las secciones anteriores, una revisión sobre los conceptos que subyacen a la teoría de los $Sistemas - (M, R)$, visitando particularmente la expresión algebraica (de consecuencias lógicas) que se expuso en la elaboración de la definición 2.3.6 sobre un $Sistema - (M, R)$. Además, los autores repasan con precisión la distinción que Rosen hizo entre simulación y modelo, discutiendo que, si es que los $Sistemas - (M, R)$ son el modelo de la vida, entonces la Vida Artificial sería reducida a simulaciones útiles que no obstante no producirían un entendimiento adecuado sobre el fenómeno de la vida. Así mismo, se hace un repaso sobre diversas objeciones al modelo de Rosen, enfocándose a la línea de trabajo que intenta demostrar, de manera programable, pragmática y no tan matemática, que los $Sistemas - (M, R)$ son Turing-computables. En este espacio, los autores enlistan a los trabajos producidos, al menos hasta 2016, enfocados en instanciar computacionalmente a los $Sistemas - (M, R)$, de igual manera que lo hicieron Gatherer & Galpin en el trabajo estudiado en la sección anterior (Gatherer y Galpin, 2013), mismo que retoman indicando que diversos pares señalaron algunos errores los cuales reconocieron en el mismo artículo, como lo fueron el factor estocástico, las dificultades para expresar reacciones metabólicas complejas, y por último: que en algunas instancias Bio-PEPA ejecutó el código del $Sistema - (M, R)$ por un tiempo que excedió, según indican los autores, «nuestra paciencia para observarlo» (sic), por lo que no pudieron determinar si el programa de hecho terminaba o no. No obstante esto, los autores sostienen que la instancia programada en Bio-PEPA tenía cualidades similares a la de la vida. Por último, los autores destacan el problema que, a juicio suyo y de sus pares, fue el más crítico: haber supuesto que B actúa como substrato metabólico para f y como catalizador para ϕ a la vez, algo que no es congruente con la definición de un $Sistema - (M, R)$. Es en este momento en el que los autores mencionan un problema central para el interés de esta tesis: las posibles corrupciones en la estructura de un $Sistema - (M, R)$ a la hora de la codificación, mismo que se retomará en el capítulo 5.

Los autores optan por hacer una traducción *verbatim* de un $Sistema - (M, R)$ al Lenguaje Unificado de Modelado (UML por sus siglas en inglés) utilizado para representar objetos y sus relaciones, y cuya naturaleza permitiría producir una representación completamente gráfica del $Sistema - (M, R)$, la cual sería fácilmente comparable — en términos de su estructura interna — con el diagrama de Rosen. Los autores, no obstante, deciden no producir un programa derivado del diagrama UML que han de derivar para evitar corrupciones al $Sistema - (M, R)$, y es en este espacio en el que

se inserta el experimento de esta tesis, mismo que se presentará en el capítulo 5. Con respecto a UML, en el capítulo siguiente se abordarán los conceptos básicos sobre sus definiciones y convenciones.

Los autores producen cuatro diagramas UML que representan a un $Sistema - (M, R)$, a saber: un diagrama de clase, un diagrama de actividad, dos diagramas de comunicación y siete diagramas de máquina de estado (donde se incluye el diagrama de máquina de estado de la totalidad de un $Sistema - (M, R)$). Los pormenores de estos diagramas, así como la traducción realizada a partir del formalismo de Rosen para llegar a ellos, será explorada con atención en el capítulo siguiente inmediato.

En la discusión del trabajo, Zhang y cols. argumentan que un análisis satisfactorio orientado a objetos, producido en UML, es congruente con la hipótesis de computar de manera exitosa tal análisis debido a la compatibilidad de UML con la Programación Orientada a Objetos. En este tenor, los autores sopesan la posibilidad de que el diagrama original de Rosen haya sido una instancia de un sistema de comunicación orientado a objetos antes de que el término fuese acuñado, lo que induciría la posibilidad de su Turing-computabilidad. En adición, se indica que la traducción *verbatim* del $Sistema - (M, R)$ a UML ayuda a preservar las consecuencias y causas que son internas al sistema y que otras exploraciones computacionales no han logrado capturar. No obstante estos progresos, los autores advierten tres potenciales dificultades observables:

1. El diagrama de comunicación al que llegan puede ser topológicamente manipulado para llegar a otro en extremo parecido al diagrama original de Rosen, pero no es posible aseverar que este diagrama de hecho garantice identidad,

2. Los autores preservan la distinción entre los objetos f y f' que en el diagrama de Rosen en realidad corresponden únicamente con el nodo f,

3. Debido a las reglas de uso de UML, el diagrama total de máquina de estado obligó a los autores a añadir un punto de inicialización y un punto de terminación.

Empero, Zhang y cols. asumen que tales problemas son justificables y admisibles, y que el trabajo realizado representa un paso importante para explorar la computabilidad de los $Sistemas - (M, R)$ debido a que un diagrama UML bien formado es implementable en un lenguaje de programación orientado a objetos.

En el siguiente capítulo se ahondará en los diagramas UML producidos por Zhang y cols., los rudimentos de UML y de Programación Orientada a Objetos requeridos para entender tales diagramas, así como en los procedimientos de traducción utilizados por los autores para producir la versión UML de los $Sistemas - (M, R)$.

4 El sistema-(M,R) como diagrama UML

4.1. Orientación a Objetos

La programación de máquinas computantes es un arte cada día más apreciada y valuada, cuyo desarrollo no es sencillo de relatar: hay bifurcaciones y diversas escuelas que se constituyeron en torno al enorme hallazgo de Turing y del establecimiento de la Ciencia de la Computación. Hacer Historia de la Ciencia en este tenor no es parte del alcance de este trabajo, pero sí lo es el realizar una revisión sobre el paradigma de programación que ha de ser utilizado para llevar a término el experimento nuclear de esta tesis: el paradigma de la Programación Orientada a Objetos.

La orientación a objetos es una de las diversas buenas prácticas de la industria e ingeniería del software, adoptado como un estándar por el *Object Management Group* (OMG, 2017). Hacer algo orientado a objetos es considerar a los contenidos de un sistema como objetos (tanto cuanto los objetos tangibles del mundo analógico) que podrán interactuar con otros objetos y que representan aspectos de interés de manera fácilmente entendible al extender, al mundo digital, a la definición de objeto real (Jacobson, 1992). Un **objeto** es una colección de datos con comportamientos asociados (Phillips, 2015). Los objetos pueden relacionarse entre sí de forma **estática** (a largo plazo y con conocimiento pleno de uno y del otro) o **dinámica**, que en sí representa la relación de comunicación entre objetos (Jacobson, 1992). Una característica central a los objetos es el **encapsulamiento**, que refiere al hecho de que el comportamiento y los datos de un objeto están almacenados exclusivamente en el objeto y son únicamente manipulables mediante las operaciones mismas del objeto, haciendo invisible su estructura interna al exterior. Este es un aspecto que es similar al de **abstracción**), aunque no es igual: mientras que el encapsulamiento esconde los mecanismos internos de un objeto, la abstracción sirve para ocultar información necesaria o irrelevante al usuario, haciendo más sencillo y accesible el entendimiento del

sistema que se está diseñando o programando. (Jacobson, 1992). Objetos que tengan las mismas características generales de tal suerte que una agrupación de estos sea natural (e.g. agrupar manzanas y peras en el grupo de las frutas) representan al molde general, descriptor de estructuras internas para objetos, que se conoce como **clase**, mientras que un objeto creado a partir de una clase se conocerá como **instancia** de una clase en particular (Jacobson, 1992). Así también es posible construir una clase B a partir de una clase A, permitiendo que B posea la estructura y comportamientos de A, a lo cual se le conoce como **herencia** de clases, misma que representa un eslabón en los procesos de **generalización** (poner una clase más arriba en la cadena de herencia) y **especialización** (poner una clase más abajo en la cadena de herencia), lo cual permite el fácil reuso de código y la mejor representación de relaciones en el mundo real (Jacobson, 1992). A su vez, cuando dos instancias se comunican no es necesario que la instancia emisora conozca la clase de la instancia receptora, a lo cual se le conoce como **polimorfismo** (Jacobson, 1992).

Las características de los objetos son representadas por datos que se conocen como **atributos** y que significan las propiedades del objeto en sí (e.g. el color y la especie de una flor), mientras que los comportamientos son acciones que pueden ocurrir sobre una clase de objetos (alterando su estado interno), y a estas acciones se les conoce como **métodos**, mismos que podrían tomar parámetros (argumentos) y devolver valores (Phillips, 2015). Las clases de objetos podrán exponer métodos y atributos para poder interactuar con otros objetos mediante una **interfaz**, que suele ser un componente crucial en la especificación de un sistema orientado a objetos al permitir el buen encapsulamiento de estados internos (Phillips, 2015).

Dentro de la orientación a objetos es posible encontrar tres procesos o etapas necesarias para entender el requerimiento de un sistema (Phillips, 2015):

1. **Análisis o Exploración Orientado a Objetos**, en el cual el analista se encarga de identificar objetos e interacciones entre objetos dentro de un sistema o problema a solucionar, especificando como producto un requerimiento.

2. **Diseño Orientado a Objetos**, en el cual el diseñador traduce el requerimiento a una especificación con objetos bien nombrados, métodos definidos y relaciones de interacción entre objetos a partir de interfaces.

3. **Programación Orientada a Objetos**, en la cual el programador traduce la especificación anteriormente realizada a un programa.

4.2. Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés) fue diseñado para permitir el modelado visual de objetos para el diseño de software, permitiendo el análisis orientado a objetos necesario para guiar a una mejor programación orientada a objetos, aunque su uso puede extenderse a la visualización de procesos entendibles mediante la orientación a objetos, como lo son los negocios (OMG, 2017). En este caso, la extensión de interés es hacia los $Sistemas - (M, R)$.
UML suele ser parte del proceso de Diseño Orientado a Objetos y un buen diseño UML es programable en un lenguaje de programación orientado a objetos. En esta sección se revisarán los rudimentos de UML necesarios para poder analizar los diagramas UML de los $Sistemas - (M, R)$ (Zhang y cols., 2016) a partir de la especificación descrita en el reporte técnico del Object Management Group (OMG, 2017).

4.2.1. Diagrama de Clase

Un diagrama de clases se utiliza para representar clases — según la definición de la Orientación a Objetos — y sus relaciones. Una clase en un diagrama de clases se encuentra estructurada por secciones. La sección superior es la que contendrá al nombre de la clase, escrita en negritas y centrado en su contenedor. La sección siguiente contendrá atributos característicos de la clase, cuyos nombres se escriben utilizando mayúsculas mediales/*camelCase*. La última sección contendrá los comportamientos u operaciones que la clase admite, y cuyas iniciales también serán siempre minúsculas.

La visibilidad de atributos y operaciones, en consonancia con el concepto de encapsulamiento del paradigma de orientación a objetos, será establecido por la notación prefija:

- `+` para indicar que su visibilidad es pública,
- `-` para indicar que su visibilidad es privada,
- `#` para indicar que su visibilidad es protegida,
- `~` para indicar que su visibilidad es de paquete.

Las clases pueden relacionarse de diversas formas, mismas que son expresadas en la tabla 4.1. La figura 4.1 ejemplifica una clase que tiene como atributo público el nombre de su marca y como atributo privado su tiempo máximo de uso continuo.

Sus operaciones públicas son aspirar y dejar de aspirar, mientras que su operación privada será incendiarse. La figura 4.2 ejemplifica relaciones entre clases en donde **HooverConstellation**, además de tener sus propios contenidos, hereda los atributos y comportamientos de **Aspiradora**, mientras que esta última clase se relaciona lógica y estructuralmente con la clase **Color**.

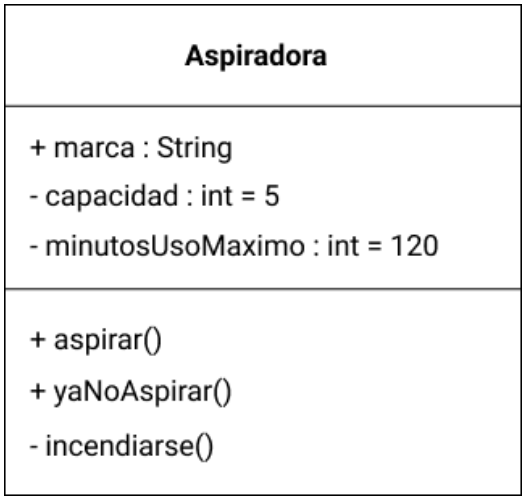


Figura 4.1: Ejemplo de una clase llamada **Aspiradora**.

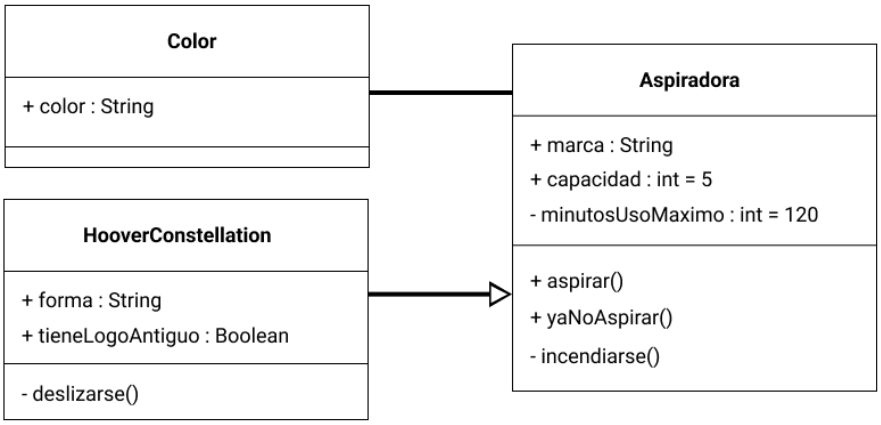


Figura 4.2: Ejemplo de relaciones entre clases.

Notación	Relación	Descripción
	Asociación	Representan relaciones lógicas y estructurales en un sistema que pueden tener restricciones o no. Las asociaciones pueden ser reflexivas también.
	Asociación unidireccional	Representan un flujo de relación unidireccional entre dos clases.
	Asociación bidireccional	Representan un flujo bidireccional de relación entre dos clases.
	Generalización / Herencia	Representan la herencia de una clase hija a partir de atributos y comportamientos de una clase parental. Se dibuja desde la clase hija hacia la clase parental.
	Implementación / Realización	Representan la implementación de un comportamiento definido en una clase. Se dibuja desde la clase implementadora hacia la clase definidora.
	Agregación	Representa la construcción de una clase a partir de una colección de otra clase. Se dibuja desde la clase parental hacia la clase agregada.
	Composición	Similar a la agregación salvo que se establece una dependencia: si la clase parental se destruye, la clase compuesta también.
	Dependencia	Representa dependencia de una clase cliente a partir de una clase proveedora. Si algo cambia en la proveedora, la clase cliente se verá afectada. Se dibuja desde la clase proveedora hacia la clase cliente.

Tabla 4.1: Las diferentes formas de representar relaciones en diragramas de UML. No se denotan cardinalidades debido a que los diagramas UML del $Sistema - (M, R)$ no las utilizan.

4.2.2. Diagrama de Actividad

Un diagrama de actividad representa flujos de acciones. Estas representaciones hacen claras las condiciones para los flujos, ejecuciones con concurrencia, así como las posibilidades de elección dentro del flujo, todo de manera secuencial.

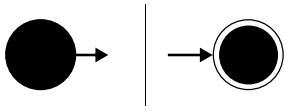
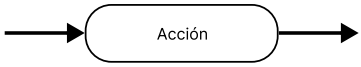
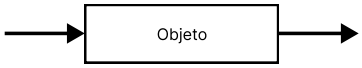
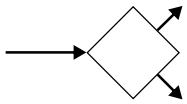
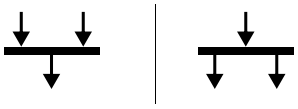
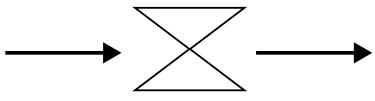
Notación	Elemento	Descripción
	Estado inicial Estado final	Representan, respectivamente, el punto de inicio del flujo diagramado, así como el punto de término del flujo.
	Acción	Representan acciones ejecutadas por o sobre objetos.
	Objeto	Representan objetos relevantes para el flujo.
	Flujo	Representan caminos que permiten transitar de una acción a otra según la direccionalidad de la flecha.
	Decisión	Representan momentos condicionales o de toma de decisiones para poder elegir un camino u otro dentro del diagrama.
	Barra	Representan puntos de partida para una `bifurcación` o puntos de llegada para una `reunión` de acciones concurrentes.
	Evento de tiempo	Representa un momento en el diagrama en el que cierta actividad podría tomar algún tiempo en realizarse.

Tabla 4.2: La notación para elementos comprendidos en un diagrama de actividades.

La tabla 4.2 exhibe la notación para componentes de un diagrama de actividades, mientras que la figura 4.3 ejemplifica un diagrama de actividades sencillo el cual muestra que, a partir de tener hambre y sed, se puede beber agua concurrentemente con elegir comer algo salado o algo dulce. Al final, ambos caminos concurrentes se unirán antes de la acción de saciarse.

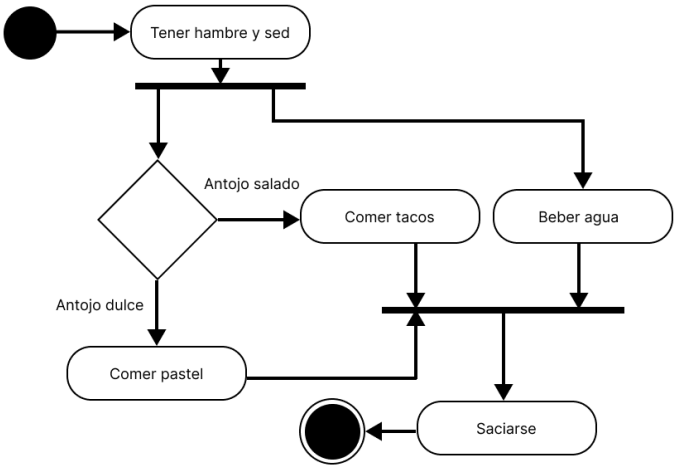


Figura 4.3: Ejemplo de un flujo de actividades.

4.2.3. Diagrama de Comunicación

Los diagramas de comunicación permiten representar interacciones entre objetos del sistema mediante los mensajes que entre ellos se dan. Estos mensajes son ordenados de manera cronológica. Los objetos serán representados dentro de rectángulos. El flujo y direccionalidad de mensajes será representado por líneas con una flecha flotante encima que designará la direccionalidad, y sobre la línea se establecerá el mensaje ejemplar cronológicamente etiquetado, de forma que el primer mensaje será `1 : Mensaje1`, el siguiente será `2 : Mensaje2`, y así sucesivamente. Las convenciones para nombrar a los objetos salen del alcance de este trabajo al no ser utilizadas por Zhang y cols. en su diagrama de comunicación (Zhang y cols., 2016). La figura 4.4 ejemplifica un diagrama de comunicación sencillo para usuarios de un intercambiador de crypto-tokens. En tal diagrama, el primer paso identifica al Tablero del Inversionista enviando el mensaje (1) de revisar los activos a la lista de activos. Ahí, cuando el usuario tiene interés, éste puede comprar (o poner la orden de compra de) un activo, lo cual primero emite un mensaje al objeto OrdenDeCompra (2) que revisará en sí mismo que haya disponibilidad para la compra (3). Finalmente, el objeto

OrdenDeCompra enviará dos mensajes al objeto CarteraDeUsuario: (4) para revisar si hay fondos, y (5) para asignar el activo o token de activo comprado al usuario inversionista.

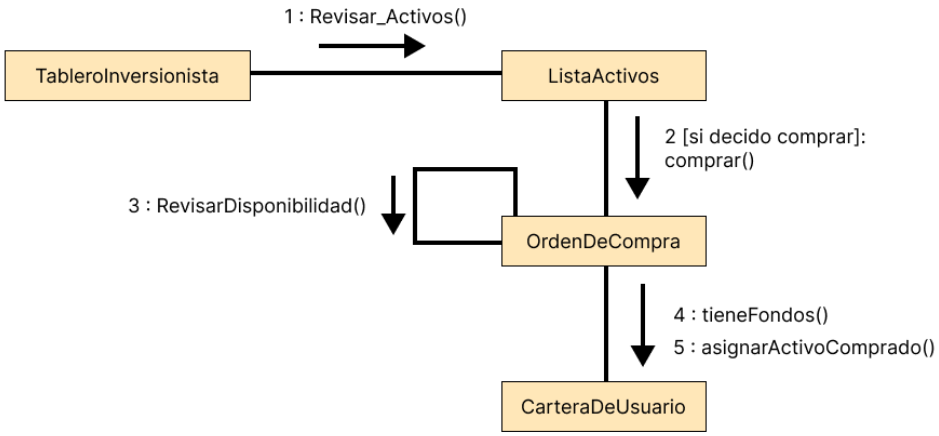


Figura 4.4: Ejemplo de un diagrama de comunicación para un usuario que utiliza un software para invertir en acciones y activos tokenizados.

4.2.4. Diagrama de Máquina de Estados

Los diagramas de máquinas de estado sirven para modelar los comportamientos de un sólo objeto, representando los distintos estados en los que se pueda encontrar. Estos diagramas utilizan la simbología de la tabla 4.2 exceptuando al evento de tiempo. La flecha de flujo — que en este caso indica una transición de un estado i a un estado j — que conecta dos estados, lleva sobre ella texto que indican los disparadores, condiciones o efectos de tal transición, de la forma: `Disparador [Condición]/Efecto`. Para los fines de este trabajo, este nivel de profundidad bastará. La figura 4.5 ejemplifica un diagrama sencillo de máquina de estado en donde el objeto recibe la orden de compra y pasa a un estado de revisión que puede tener dos vertientes: que al revisar la condición sea positiva, o bien negativa. Si es positiva, el objeto pasará al estado de orden aceptada y llegará al estado final mediante el disparador de ir a la cartera del usuario. Si la revisión es negativa, el objeto pasará a un estado de rechazo de orden y luego al estado final mediante un disparador que devolverá al usuario a su tablero y producirá un error.

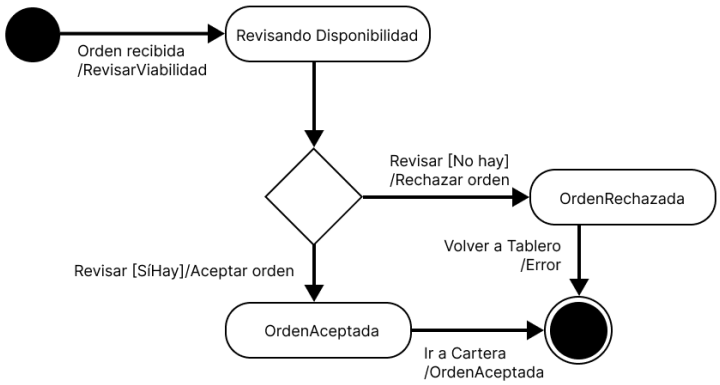


Figura 4.5: Ejemplo de un diagrama de estados para la recepción de una orden de compra dentro del diagrama de comunicación de la figura 4.4.

4.3. El $Sistema-(M,R)$ como diagrama UML: la traducción de Zhang y cols. (2016)

Como se mencionó en el capítulo anterior, Zhang y cols. produjeron cuatro diagramas UML que representan a un $Sistema-(M,R)$ (Zhang y cols., 2016): un diagrama de clase, un diagrama de actividad, dos diagramas de comunicación y siete diagramas de máquina de estado. En esta sección se revisarán los diagramas en cuestión, haciendo énfasis en el significado de cada elemento en ellos en tanto su relación con los $Sistemas-(M,R)$.

4.3.1. El diagrama de clases de un $Sistema-(M,R)$

A partir de las definiciones de clase y objeto que en este capítulo ya se revisaron, Zhang y cols. diseñan el diagrama de clases de un $Sistema-(M,R)$ partiendo de la entidad u objeto más general en el interés del formalismo de Rosen: las **biomoléculas** (Zhang y cols., 2016). A partir de la clase `Biomolecula` nacen dos clases herederas: la clase `Substrato` y la clase `Enzima`, las cuales no tienen atributo alguno pero sí conductas: para la clase `Substrato` se especifica el método `producirOtrasBiomoleculas()`, mientras que la clase `Enzima` tendrá definido el método `catalizarSubstrato(Substrato):Producto` (una interfaz que se volverá una clase concreta en cada subclase de `Enzima`), mismos que son los métodos principales de activación. Cada una de este par de clases es también la clase pa-

rental de tres clases herederas para cada una: para la clase `Substrato` existen las clases herederas `A, B,` y `f`, mientras que para la clase `Enzima` existen las clases herederas `b, f',` y ϕ. Estas clases tampoco tendrán atributos pero sí métodos, a saber: para la clase `A` estarán definidos los métodos `producirB()` y `producirb()`; para la clase `B` estarán definidos los métodos `producirf()` y `producirf'()`; para la clase `f` estará definido el método `producir`ϕ`()`; para la clase `b` estará definido el método `catalizarReplicacion(f):`ϕ; para la clase `f'` estará definido el método `catalizarMetabolismo(A): B / b`; y para la clase ϕ estará definido el método `catalizarReparacion(B):f/f'`. Cada uno de los métodos de producción y catálisis corresponden a un camino de causación en el diagrama de Rosen. Por ejemplo, en el diagrama de Rosen tenemos que f es la causa eficiente que, a partir de la causa material A, produce la causa final B (i.e. $f \vdash B$ o bien $f : A \mapsto B$), lo que corresponde con el diagrama relacional:

$$f \longrightarrow A \longrightarrow B$$

Este mismo diagrama puede entenderse como la representación de una función f que toma como parámetro a A y devuelve un B como resultado. Así, entonces, para la clase **f** se define `catalizarMetabolismo(A):B/b`, donde además aparece b porque los autores están suponiendo a β distinta de B[1] (que en este caso $\beta = b$). Esta misma lógica constructiva se utilizó sobre las otras dos diagramas relacionales contenidos en el $Sistema - (M, R)$, a saber:

$$b \longrightarrow f \longrightarrow \phi$$

Corresponde a la clase **b** con un método definido de la forma:
`catalizarReplicacion(f):`ϕ

$$\phi \longrightarrow B \longrightarrow f/f'$$

Corresponde a la clase ϕ con un método definido de la forma:
`catalizarReparacion(B):f/f'`

[1]Zhang y cols. también mantienen la distinción entre f y f', aunque argumentan que es posible que haya otras formas de diagramar las clases de un $Sistema - (M, R)$, mismas que permitan deshacerse de β y f' (Zhang y cols., 2016). En su publicación, los autores no hacen mención de cómo es que estas otras formas de representación podrían ser construidas.

Por otro lado, para los sustratos `A, B,` y `f` se especifican métodos productivos del valor de retorno esperado dentro de los correspondientes métodos enzimáticos anteriormente especificados para `b, f',` y ϕ

Ninguna clase tiene atributos, respetando la perspectiva de Rosen sobre la vida y los sistemas naturales: no importa la composición material de éstos (atributos), sino cómo se relacionan (comportamientos). Finalmente, todos los comportamientos especificados en el diagrama de clases serán públicos. El diagrama de clases del $Sistema - (M, R)$ diseñado por Zhang y cols. (2016) está adaptado al español en la figura 4.6

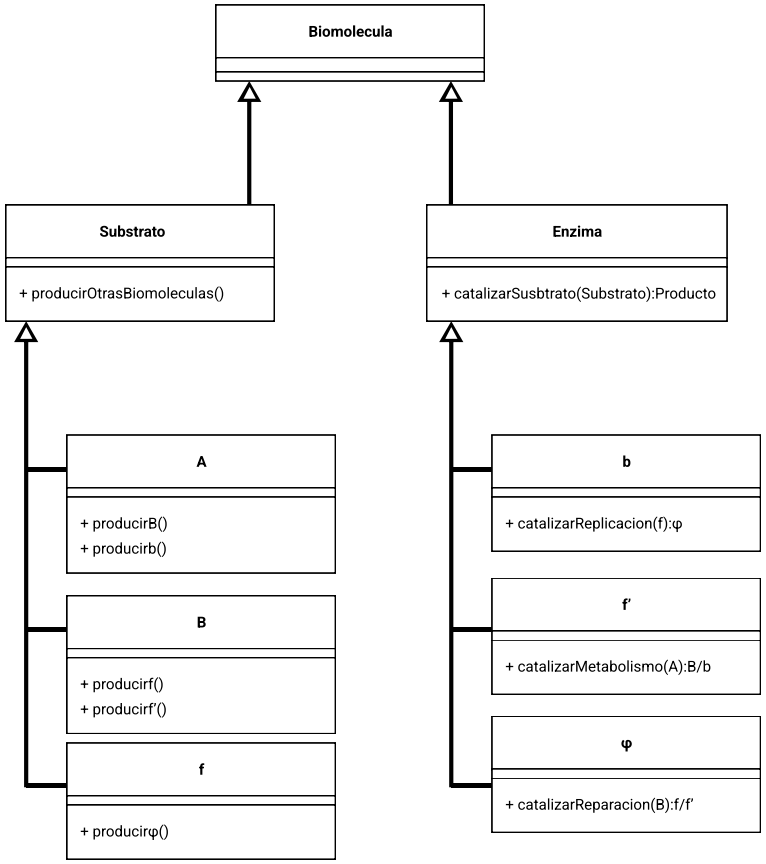


Figura 4.6: El diagrama de clases de un $Sistema - (M, R)$. Adaptado a partir de "Rosen's (M,R) system in Unified Modelling Language" por L. Zhang, R.A. Williams, & D. Gatherer, 2016, *Biosystems*, 139, p. 14 del manuscrito aceptado.

4.3.2. El diagrama de actividades de un $Sistema - (M, R)$

El diagrama de actividades que producen Zhang y cols. a partir del diagrama de clases de un $Sistema - (M, R)$ representa los flujos de biomoléculas dentro de un $Sistema - (M, R)$ a partir de los efectos causados por los métodos descritos en las clases (Zhang y cols., 2016). Estos diagramas representan los fragmentos (o subsistemas) computables de un $Sistema - (M, R)$ que se mencionaron anteriormente,[2]

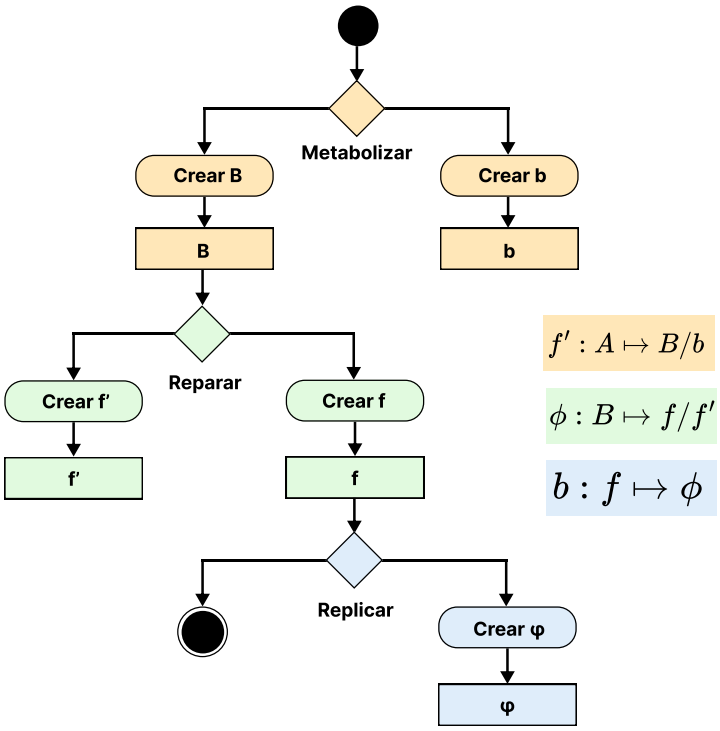


Figura 4.7: El diagrama de actividad de un $Sistema - (M, R)$. Adaptado a partir de "Rosen's (M,R) system in Unified Modelling Language" por L. Zhang, R.A. Williams, & D. Gatherer, 2016, *Biosystems*, 139, p. 15 del manuscrito aceptado.

La figura 4.7 adapta el diagrama de actividades diseñado por Zhang y cols. (2016) aumentado con etiquetas de transformación química correspondientes a aquellas especificadas por Rosen. En él mismo, el orden de causalidad es el siguiente: $f' \vdash B/b, \phi \vdash f/f', b \vdash \phi$ y si se añade la circularidad, podría continuarse de la forma $f' \vdash B/b, \ldots$. Otro orden posible es el «canónico» (i.e. y considerando la distinción entre f y f': $f' \vdash B/b \vdash \phi \vdash f/f' \vdash B/b, \ldots$) pero ambos diagramas recorren los mismos caminos . El diagrama de actividades implica que cuando se crean b, f' y

[2]Zhang y cols. (2016) dicen ahí msimo que, crucialmente, la Biología Relacional rechaza, específicamente, que tales composiciones secuenciales sean representaciones completas de (M, R), pero, conversamente, admiten que son computables. (p. 15)

ϕ, estas enzimas quedan disponibles en el contexto del $Sistema - (M, R)$, mientras que el diagrama deja en claro que el sustrato A tendrá que ser suministrado desde el exterior, como alimento. En este diagrama, los autores indujeron estados iniciales y finales arbitrarios debido a que la actividad dentro de un $Sistema - (M, R)$ es cíclica (e.g. después de crear ϕ es posible catalizar B con ϕ para producir f' y nuevamente catalizar A con f' para producir b, *et sīc in infinitum*).

4.3.3. Los diagramas de comunicación de un $Sistema - (M, R)$

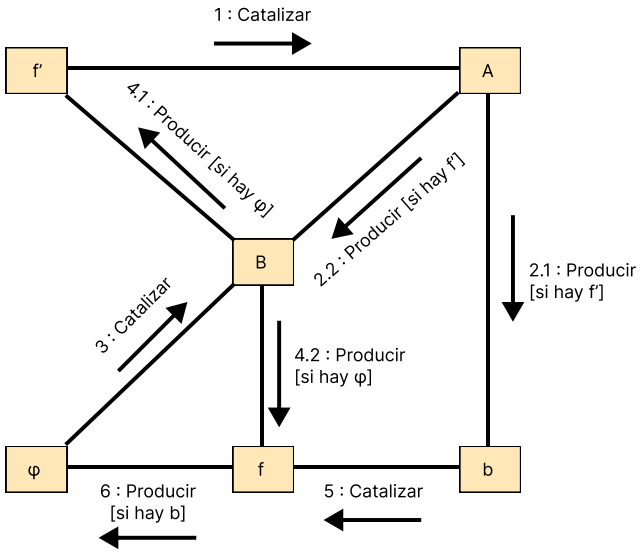


Figura 4.8: El diagrama de comunicación de un $Sistema - (M, R)$ derivado del diagrama de actividades del $Sistema - (M, R)$. Adaptado a partir de "Rosen's (M,R) system in Unified Modelling Language" por L. Zhang, R.A. Williams, & D. Gatherer, 2016, *Biosystems*, 139, p. 17 del manuscrito aceptado.

El diagrama de comunicación diseñado por Zhang y cols. es quizás el más interesante de todos, al ser topológicamente manipulable para ser llevado a una representación extremadamente similar al diagrama original de Robert Rosen, razón por la cual se habla de dos diagramas de comunicación en los productos de los autores (Zhang y cols., 2016). A partir de la exhibición de los efectos de los métodos de las diversas clases, realizada en el diagrama de actividad, los autores se enfocan ahora en la organización de las clases y la forma en la que están conectados (nuevamente, de manera topológica). Los autores argumentan que, debido a que los diagramas de comunicación permiten ciclos (contrario a los diagramas de actividad), esto significa

que en el diagrama que producen existe una composición jerárquica (o un ciclo jerárquico, según la definición 2.2.3). La figura 4.8 adapta el diagrama de comunicación al que Zhang y cols. (2016) llegan a partir del trabajo realizado en el diagrama de actividades, mientras que la figura 4.9 muestra la manipulación topológica realizada por los autores para producir un diagrama prácticamente idéntico al de Rosen, mismo que se incluye en la misma figura como comparativo. Los autores admiten que no tienen garantía de que esta manipulación produzca, de hecho, un diagrama que sea la identidad del diagrama de Rosen. En el diagrama de Rosen se han añadido los pasos de catálisis y producción equivalentes al paso de mensajes que ocurre en el diagrama de comunicación.

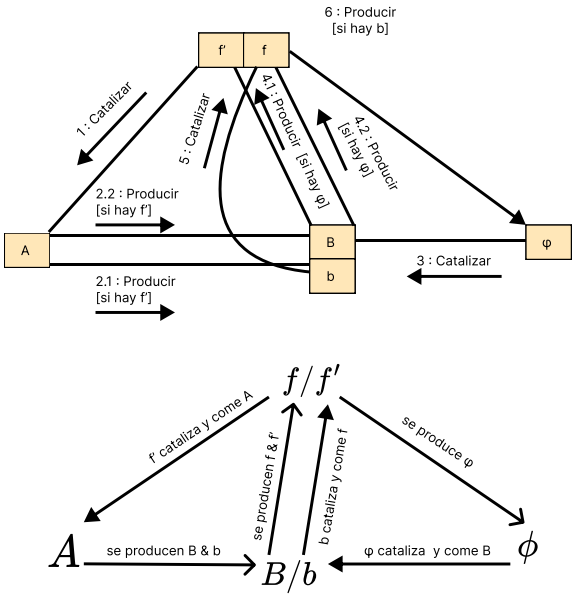


Figura 4.9: El diagrama de comunicación de un $Sistema - (M, R)$ manipulado topológicamente para acercarse al diagrama de Rosen, también contenido en esta figura como comparativo. Adaptado a partir de "Rosen's (M,R) system in Unified Modelling Language" por L. Zhang, R.A. Williams, & D. Gatherer, 2016, *Biosystems*, 139, p. 17 del manuscrito aceptado.

4.3.4. Los diagramas de máquina de estados de un $Sistema - (M, R)$

Finalmente, Zhang y cols. producen los diagramas de máquinas de estado para representar los posibles estados de comportamiento de cada objeto contenido en el ciclo de reacciones bioquímicas que es el $Sistema - (M, R)$ (Zhang y cols., 2016). Los autores indican que para las enzimas f, b y ϕ decidieron establecer 3 reusos máximos antes de que tales objetos sean destruidos debido a que en el diagrama original de

Rosen (y en sí, en la especificación del $Sistema - (M, R)$) no hay una declaración explícita sobre el destino de tales catalizadores (i.e. ¿se usan y se agotan? ¿tienen una cantidad de reusos limitada? ¿son inmortales y se acumulan?). Para esto, los autores indican haber agregado un atributo `memoria` a la clase **Enzima** (mismo que sería heredado por las clases herederas) para poder llevar un conteo sobre el reuso del objeto mismo mediante el uso de un método privado que aumentará el contador cada vez que el método `catalizarSubstrato(Substrato)` se activare. Mientras que los autores argumentan que tal atributo no se ha incluido en el diagrama de clases para mantener la genericidad (un diagrama de clases extendido es agregado a este trabajo en la figura 4.12, aumentado con un atributo privado (`-reusos:int=0`) en cada clase parental **Substrato** y **Enzima** que le permitirá a éstas llevar un conteo de sus reusos. Igualmente se añade el método privado de aumentar el contador de memoria para cada momento de reuso (`-incrementaMemoria()`) y un método privado de autodestrucción para cuando se exceda la cuota de resusos (`-autodestruccion()`). Se añaden anotaciones sobre las clases de objetos producibles por las clases correspondientes a ϕ y f' para mantener la homogeneidad requerida por UML mediante la especificación de un tipo de retorno más general para ambas: `Biomolécula`), es importante destacar que en $Sistema - (M, R)$ original de Rosen no hay mención de una memoria integrada a cada biomolécula, sino que su método de uso y desgaste podría estar dado por las mismas propiedades de fragilidad de las mismas biomoléculas (esta discusión excede el alcance del presente trabajo, pero es un aspecto relevante al considerar que se han añadido elementos a una estructura que es, de otra forma, bastante genérica, con el fin de poder codificar un método de uso y desgaste. Este aspecto se considerará durante las conclusiones). En el caso de los substratos A, B y f, estos no son destruidos sino metabolizados (i.e. transformados) en otros objetos del sistema. Los autores igualmente destacan que es importante recordar que, mientras es cierto que la Biología Relacional admite la representación de componentes de un $Sistema - (M, R)$ mediante estados, ésta rechaza el que un $Sistema - (M, R)$ en su completitud pueda ser representado por estados, siendo estos una característica central de los mecanismos de los que Rosen intentó alejarse. Por esto mismo, la Biología Relacional rechaza que componer un diagrama de máquina de estados general — a partir de los diagramas de máquinas de estados individuales para cada objeto — represente un $Sistema - (M, R)$ al ser ésta una escuela del pensamiento que rechaza el reduccionismo que precisamente busca entender fenómenos grandes a partir de su descomposición en partes. No obstante esto, los autores producen un diagrama de máquina de estados para la totalidad del $Sistema - (M, R)$ dentro de una estruc-

tura circular que, no obstante, requiere estados de inicio y de finalización (puestos de manera arbitraria) debido al requerimiento mismo de un diagrama de máquina de estados en UML. La figura 4.11 adapta los diagramas de máquinas de estado producidos por Zhang y cols. (2016). En él, los substratos A, B y f son inicializados y puestos en espera para ser consumidos, y una vez se encuentran en la presencia de su respectiva enzima, se convierten en otras biomoléculas enzimáticas. Para las enzimas b, f' y ϕ se les inicializa una vez creadas y se les pone en inactividad, a la espera de ser utilizadas. Así, una vez en la presencia de su substrato asociado, estas enzimas registraran en memoria un uso, y en caso de exceder los tres usos, se destruirán.

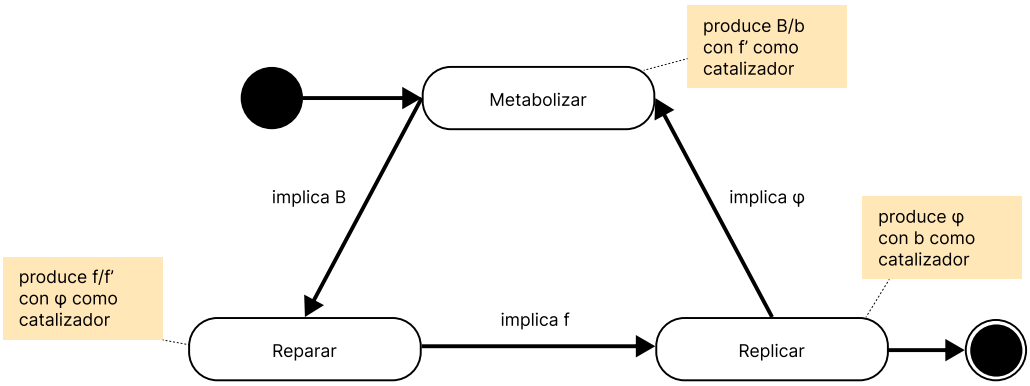


Figura 4.10: El diagrama de máquina de estados para la totalidad del $Sistema - (M, R)$ con anotaciones auxiliares. Adaptado a partir de "Rosen's (M,R) system in Unified Modelling Language" por L. Zhang, R.A. Williams, & D. Gatherer, 2016, *Biosystems*, 139, p. 22 del manuscrito aceptado.

Por otro lado, la figura 4.10 adapta el diagrama de máquinas de estado para la totalidad del $Sistema - (M, R)$ con anotaciones para expresar lo que ocurre en cada estado, por ejemplo: para el estado de `Metabolizar` se requiere que el objeto `A` se encuentre `en espera`, mientras que el objeto `f'` deberá encontrarse en estado `Activo` y destruirse en caso de superar su cuota de reuso. A su vez, el estado `Metabolizar` inicializa objetos de la clase **B** y les inicializa `en espera`, mientras que el objeto `A` es destruido (o, en el caso bioquímico, convertido en otro producto metabólico). Debido a la naturaleza de una máquina de estados, un $Sistema - (M, R)$ modelado en su totalidad con un diagrama de máquina de estados tendría que encontrarse, a lo más, en un estado por cada unidad de tiempo. Los $Sistemas - (M, R)$ no se diseñaron sobre una perspectiva mecanicista, por lo que es posible suponer que un $Sistema - (M, R)$ se encontrará en varios estados en la misma unidad de tiempo. Los autores indican

que la utilidad de UML para modelar a los $Sistemas - (M, R)$ encuentra aquí su límite y que podría llegar a ser necesario utilizar metodologías de manejo concurrente de estados para resolver este problema.

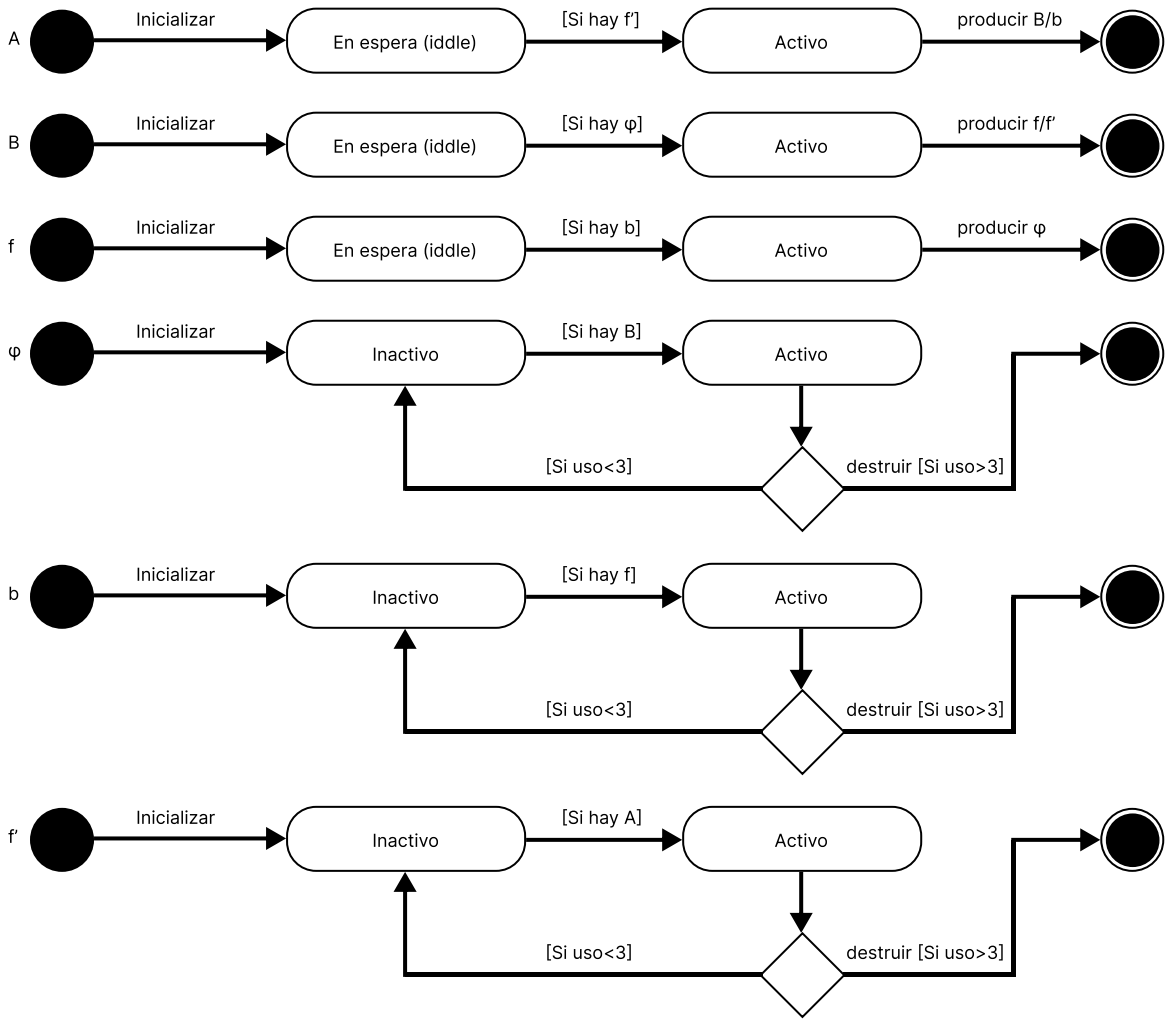


Figura 4.11: Los diagramas de máquinas de estado para cada entidad biomolecular en un $Sistema-(M, R)$. Adaptado a partir de "Rosen's (M,R) system in Unified Modelling Language" por L. Zhang, R.A. Williams, & D. Gatherer, 2016, *Biosystems*, 139, p. 20 del manuscrito aceptado.

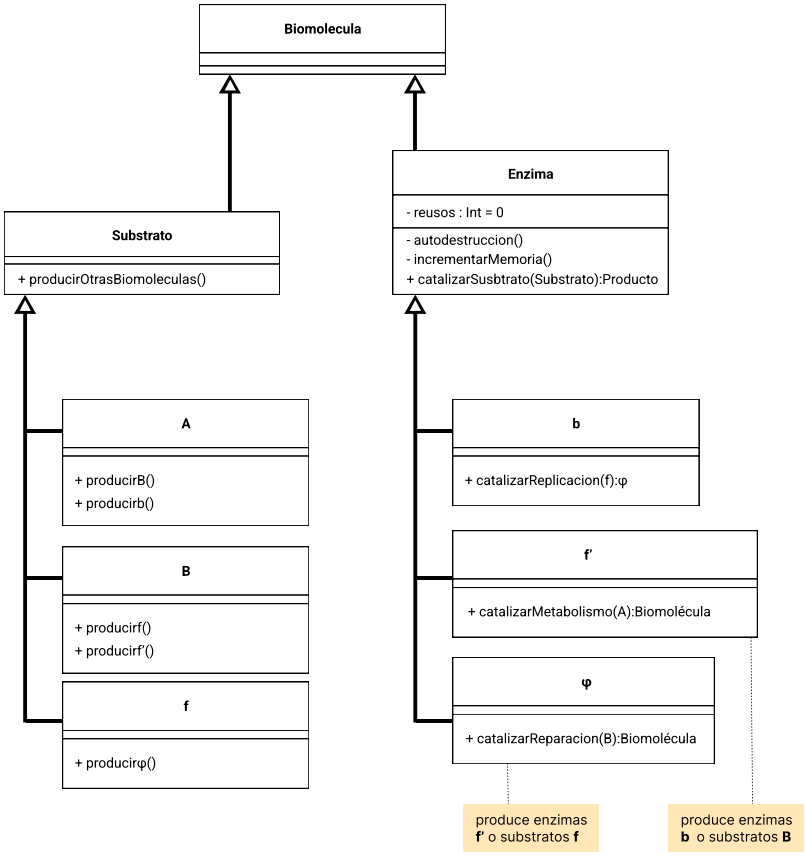


Figura 4.12: El diagrama de clases de un $Sistema - (M, R)$ extendido.

4.4. Las Máquinas-X y los $Sistemas - (M, R)$ en UML

Como se mencionó en el capítulo anterior, el trabajo en torno a las Máquinas-X y los $Sistemas - (M, R)$ (Palmer y cols., 2016) indujo también a explorar a las Máquinas-X Comunicantes en una perspectiva orientada a objetos, utilizando UML. Los autores indican que el diagrama de comunicación de UML es un equivalente, al menos conceptual, a la matriz de relaciones R que en este trabajo se presentó en la tabla 3.3. En dicha publicación, Palmer y cols. producen sólamente bocetos (o, como ellos les llaman, «caricaturas») de una traducción a UML de la especificación de la Máquina-X Comunicante para un $Sistema - (M, R)$, pero sí proponen un buen ejemplo al respecto. Los autores representan a los objetos como Máquinas-X Comunicantes individuales pero relacionadas bajo R, uniendo a objetos cuya comunicación está permitida con una flecha de asociación doble. Además, a cada objeto se le asigna la relativa memoria M, característica de las Máquinas-X de flujo, y de acuerdo a la

tabla 3.1, extendiendo así nuevamente el diagrama de clase de un $Sistema-(M,R)$ (Zhang y cols., 2016). Esta extensión se muestra en la figura 4.13, misma que exhibe la adición de Memoria (M_X) en cada Máquina-X de flujo según se define en 3.1 y según las reglas de comunicación establecidas en la tabla 3.3. Véase que bM_X no tiene con quién comunicarse en este diagrama porque el objeto definido en la matriz para bM_X no se encuentra dibujado . Los autores argumentan que esta adaptación ayuda a resarcir algunos problemas que nacieron de la publicación de Zhang y cols. (2016), en particular aquel que implica el que el diagrama de máquina de estados para la totalidad de un $Sistema-(M,R)$ no sea suficientemente convincente, a pesar de haber modelado a objetos como máquinas de estados de manera satisfactoria. Palmer y cols. indican, pues, que cada diagrama máquina de estado para cada objeto en un $Sistema-(M,R)$ es una representación válida de una Máquina-X individual embebida en un ambiente comunicante, i.e. el $Sistema-(M,R)$ mismo, por lo que la Máquina-X comunicante aditada a este análisis provee la pieza faltante, aunque los autores no producen el análisis UML completo de nueva cuenta, sólo lo visitan (Palmer y cols., 2016). Finalmente, los autores indican que la selección estocástica para producir B/β y para producir f/f' se introduce a su análisis UML para Máquinas-X Comunicantes derivado del análisis publicado por Zhang y cols. (2016), aunque en tal publicación jamás se haya mencionado tal selección. Debido a que se trata del mismo grupo de trabajo, en lo subsecuente se supondrá que en Zhang y cols. (2016) los autores tenían prevista esta selección estocástica, misma que será implementada.

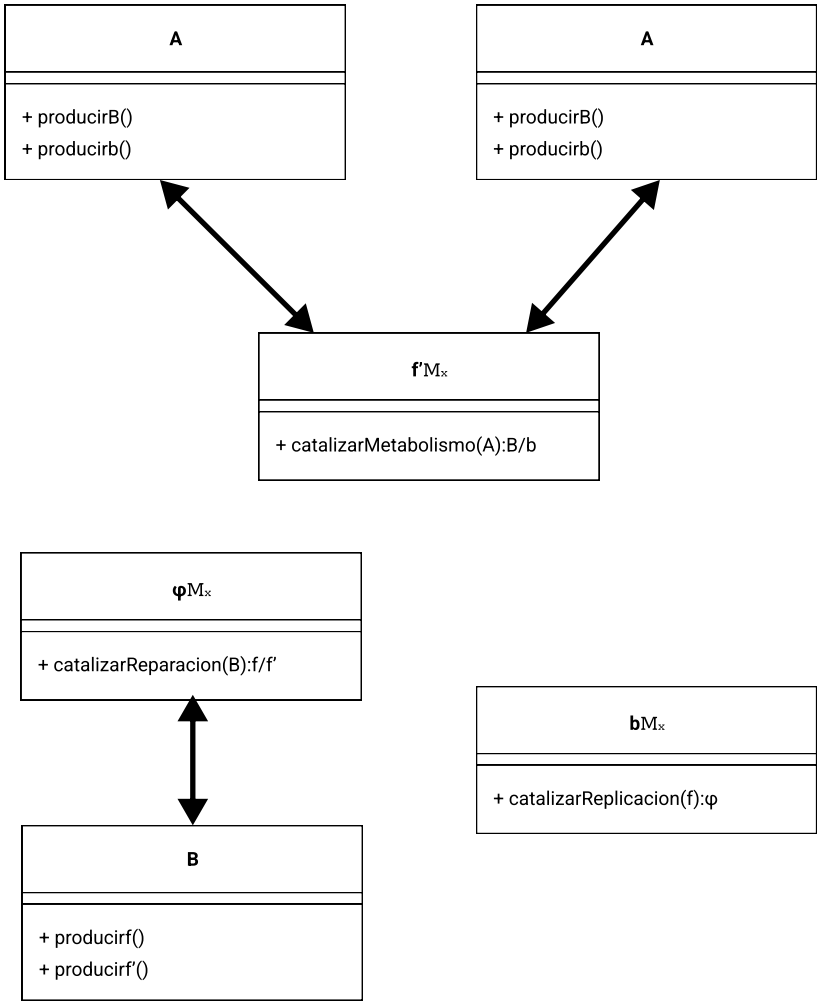


Figura 4.13: El diagrama de clases de un $Sistema - (M, R)$ extendido como Máquina-X Comunicante. Adaptado a partir de "Rosen's (M,R) system as an X-machine" por M.L. Palmer, R.A. Williams, & D. Gatherer, 2016, *Journal of Theoretical Biology*, 408, p. 102.

5 Implementación en Python (Py3K)

El objetivo principal de este trabajo de titulación es el de progresar el entendimiento computacional de los $Sistemas-(M,R)$ a partir de la publicación del trabajo realizado por Zhang y cols. para llevar a los $Sistemas-(M,R)$ a una expresión en UML (Zhang y cols., 2016), en particular mediante la implementación de los diagramas UML en un lenguaje de programación que admita el paradigma orientado a objetos. El presente capítulo, que es el capítulo con contenido experimental, se construye a partir de los objetivos particulares: facilitar el entendimiento sobre los $Sistemas-(M,R)$ y sus antecedentes, así como analizar trabajos de otros autores, realizados en el ámbito computacional, cuyo fin sea explorar a los $Sistemas-(M,R)$ en lo computable más que en lo filosófico o biológico. Con estos elementos básicos y conocimientos adquiridos, es posible ahora proceder a plantear una implementación de los diagramas UML de Zhang y cols. (2016).

5.1. Problema

Zhang y cols. (2016) no producen la implementación de los diagramas UML diseñados para modelar a un $Sistema-(M,R)$ y proponen que esto podría ser tarea de alguien más, a pesar de que este paso podría suponer añadir «corrupciones» al modelo.

5.2. Objetivo de la fase experimental

Producir una implementación de los diagramas UML de Zhang y cols. (2016) —considerando las adecuaciones hechas en el segundo diagrama de clases que se exhibe en la figura 4.12— en un lenguaje de programación que admita el paradigma de orientación a objetos con el fin de poner a prueba la hipótesis.

5.3. Hipótesis

El $Sistema - (M, R)$, modelado en UML por Zhang y cols. (2016) a partir del formalismo de Robert Rosen e implementado en un lenguaje que admita la orientación a objetos, no producirá un modelo fidedigno del $Sistema - (M, R)$ carente de las corrupciones a las que Zhang y cols. hacen referencia (2016), i.e.: se programará el modelo de un objeto de estudio que será similar pero no igual a un $Sistema - (M, R)$ al acoplar características ajenas al modelo de Rosen auxiliares en los fines de programación pero que alejan al programa del formalismo.

5.4. Metodología: implementación en Python

Para realizar el experimento de implementación se eligió a Python, en su versión 3.10.4, el cual es un lenguaje de programación que admite la orientación a objetos (Kuhlman, 2012). Aquí se enumeran, ahora, los pasos metodológicos llevados a término para los fines experimentales de este trabajo de tesis:

1. Se analizaron y se programaron los diagramas UML *verbatim*.

2. Se realizaron las necesarias adecuaciones al código derivado de UML para dar funcionamiento a las relaciones establecidas entre los objetos diseñados en la versión UML de los $Sistemas - (M, R)$ y se produjo el programa final.

3. Se realizaron diversas simulaciones, graficadas con ayuda de `MatPlotLib`, con el fin de observar el comportamiento del $Sistema - (M, R)$ bajo los parámetros establecidos por Zhang y cols. (2016). Crucialmente, se ejecuta además una simulación con parámetros maś cercanos a los que se utilizaron en el artículo de Gatherer & Galpin (2013) donde la simulación de Bio-PEPA mostró propiedades cercanas a la vida, según los autores. Además, se produjo otra simulación con una interpretación alineada con el diagrama original de Rosen.

5.5. Desarrollo

5.5.1. Paso 1: Análisis y Programación

Para poder implementar la versión orientada a objetos de los $Sistemas - (M, R)$ fue primero necesario analizar los diagramas UML producidos por Zhang y cols. (2016).

En primera instancia, se aprovechó el diagrama extendido de clases, mostrado en la figura 4.12, el cual considera atributos y funciones auxiliares para el funcionamiento del sistema, en particular en la clase `Enzima`: `reusos: int = 0`, que es un contador de la vida útil de enzimas, `autodestruccion()` que es una función que establece que la enzima ya no puede ser utilizada, y `incrementarMemoria()` que es una función que apoya con el incremento del contador `reusos`.

En segunda instancia, los diagramas de máquinas de estado mostrados en la figura 4.11 modelan las flechas del diagrama de Rosen. En el caso de los substratos, estas máquinas establecen el comportamiento de cada substrato (A, B, f) cuando están en la presencia de su correspondiente enzima catalizadora $(f', b, phi$, cuyo comportamiento de transformación (e.g. que A se transforme en B/b cuando f' cataliza a A, i.e. $f' : A \mapsto B/b$) se ve impreso en las funciones que cada substrato tiene definidos en sí:

- para la clase **A**, `producirB(), producirb()`;
- para la clase **B**, `producirf(), producirf'()`;
- para la clase **f**, `producirphi()`,
- con `producirOtrasBiomoleculas()` como la función común a todos los substratos que a su vez llama las funciones anteriores cuando el substrato es utilizado.

Estas funciones representan, en realidad, la transformación bioquímica de un substrato N en un producto M después de ser catalizado por una enzima K.
Por otro lado, las máquinas de estado de las enzimas establecen el comportamiento de cada enzima (f', b, ϕ) al ser utilizadas para catalizar a algún substrato. Las máquinas de estado para enzimas definen comportamientos de inactividad (estar disponible para uso) y actividad (estar en uso), mientras que la cola de cada estado de actividad

promoverá el registro de un uso para el contador de la enzima en cuestión. Todo el proceso de activación, registro de uso e inactivación será realizado por las funciones descritas en cada enzima, a saber:

- para la clase **b**, `catalizarReplicacion(f):phi`;
- para la clase **f'**, `catalizarMetabolismo(A):Biomolecula`;
- para la clase **Phi**, `catalizarReparacion(B):Biomolecula`,
- con `catalizarSubstrato(Substrato):Enzima` como la función común a todas las enzimas que a su vez llama las funciones anteriores cuando la enzima es utilizada.

En tercera instancia, el diagrama de actividades mostrado en la figura 4.7 serían utilizados para modelar el proceso iterativo en el que se ejecutará el sistema hasta cumplir ciertas condiciones de quiebre. El diagrama de actividades, que tiene puntos de inicio y término arbitrariamente colocados debido a que éste es, en principio, un sistema cíclico, indica que lo primero que ocurre en cadena descendente es que $f' : A \mapsto B/b$, lo que indica que un objeto de la clase **f'** consumirá (o catalizará) un objeto de la clase **A** para convertirlo en un objeto de la clase **B** o de la clase **b** (esta elección es estocástica en el diseño de Zhang y cols. (2016)). Posteriormente se requiere que $\phi : B \mapsto f/f'$, es decir: se requiere que un objeto de la clase **Phi** consuma un objeto de la clase **B** para producir un objeto de la clase **f** o de la clase **f'**. Finalmente y antes de repetir el ciclo se especifica que deberá ocurrir que $b : f \mapsto \phi$, es decir: un objeto de la clase **b** consumirá un objeto de la clase **f** para producir un objeto de la clase **Phi**. Este ciclo se sostiene en el código programado mediante la inserción de objetos producidos por estas reacciones a varios ambientes de biodisponibilidad representados por listas, cada una para cada tipo de objeto (i.e. 3 de substratos y 3 de enzimas). Dentro del `while`, las instrucciones que se utilizan para producir biomoléculas dependen de las enzimas: de cada lista de enzimas, en el orden establecido en el diagrama, se toma un elemento y se le pide a la enzima que realice la operación pública `catalizarSubstrato(Substrato):Producto`.

Por último, es importante destacar que el diagrama de comunicación de Zhang y cols. (2016) que se muestra en la figura 4.8 (y alternativamente en la figura 4.9) no se alinea a los requisitos establecidos por las normas UML publicadas por el OMG (2017) al no contener etiquetas adecuadas en cada flecha (e.g. la flecha 2.2 contiene

el mensaje desde un objeto `A` hasta un objeto `B` `Producir (si hay f')` que no corresponde con ningún mensaje ejecutable por el objeto `B`). Los autores indican haber utilizado referencias bibliográficas anteriores al más reciente estándar de UML (Zhang y cols., 2016), por lo que es posible suponer que sus diagramas de comunicación son conceptualmente sujetos a la interpretación, de lo cual se deriva que en este punto de implementación se tomó una guía más relacionada a las actividades comprendidas dentro del sistema, auxiliadas por la posibilidad de hacer un diagrama de comunicación cíclico. En suma, el diagrama de comunicación sirvió para comunicar la intuición de los diseñadores y para promover la intuición del programador.

5.5.2. El programa y sus adecuaciones

Aquí se exhiben y describen las funciones y atributos programados en Python para cada clase, junto con las adecuaciones realizadas durante el desarrollo. Para facilidad y claridad en el desarrollo, se optó por llamar **Beta** a la clase **b**, que de cualquier forma hace referencia al objeto β en el diagrama de Rosen. De igual manera se optó por llamar a la clase `f'` como `FPrima`.

Clase Biomolécula

Clase: `Biomolecula(object)`
Atributos: ninguno, según la especificación de herencia de Zhang y cols. (2016)
Métodos: ninguno, según la especificación de herencia de Zhang y cols. (2016).

Substrato

Clase: `Substrato(Biomolecula)`
Atributos: ninguno, según la especificación de herencia de Zhang y cols. (2016)
Métodos:

`producirOtrasBiomoleculas(self)`:
Comportamiento para producir biomoléculas. No tiene ninguna funcionalidad programada pero se hereda a clases hijas, según la especificación de Zhang y cols. (2016).

`__init__`:
Constructor de objeto para la clase Substarto. No tiene ninguna funcionalidad programada.

Enzima

Clase: `Enzima(Biomolecula)`
Atributos: para cada objeto, mediante el constructor, se tienen los siguientes atributos privados:

```
self.__limiteDeVida=reusos,
self.__degradado = False,
self.__cuentaReusos = 0
```

Métodos:

`catalizarSubstrato(self,Substrato)`:
Comportamiento para catálisis. No tiene ninguna funcionalidad programada pero se hereda a clases hijas, según la especificación de Zhang y cols. (2016).

`getStatusDegradacion(self)`:
Función auxiliar añadida para poder consultar el estatus de degradación (desnaturalización) de una enzima. Devuelve el valor de `self.degradado`

`registrarUso(self)`:
Función auxiliar añadida para poder aumentar el registro de uso de una enzima. No devuelve valor alguno.

`__incrementarUsos(self)`:
Función privada equivalente a la función `incrementarMemoria()` que aparece en el diagrama de clases. Aumenta el registro de uso de una enzima incrementado el valor de `self.__reusos`, y cuando éste cruza el límite establecido por `self.__limiteDeVida`, se llama a la función `__autodestruccion()`.

`__autodestruccion(self)`:
Función privada que cambia el valor de `self.__degradado` a `TRUE`, indicando que la enzima ya no podrá ser usada en otros ciclos.

`__init__(self, reusos)`:
Constructor de objeto para la clase Enzima que recibe un parámetro de reusos, el cual establecerá el límite de vida útil de la enzima.

A

Clase: `A(Substrato)`
Atributos: para cada objeto, mediante el constructor, se tiene:

```
self.nombre = ("A")
```

Métodos:

`producirOtrasBiomoleculas(self, option, usos, count_cycles)`:
Función que inicia la producción de otras biomoléculas, particularmente de las enzimas `B` y `Beta`. Recibe una opción que es un `String` que indica si se catalizará para `"B"` o para `"Beta"`, un entero que indica cuántos usos tendrá la enzima `Beta` a producir (valor que no se usa en el caso de producir un substrato `B`), así como un entero que identifica al ciclo actual, utilizado para postfijar un identificador al nombre de las enzimas. La función optará por usar alguna de las funciones añadidas a la especificación original para producir ya sea el substrato o la enzima, y regresará un objeto `B` o un objeto `Beta`.

`producirB(self)`:
Función añadida que regresa un objeto `B`.

`producirBeta(self, usos, count_cycles)`:
Función añadida que recibe el límite de usos y contador de ciclos tomados por `producirOtrasBiomoleculas`. y que regresa un objeto `Beta(usos, count_cycles)` configurado con el parámetro de usos y un identificador dado por el contador de ciclos.

`__init__(self)`:
`Constructor de objeto para la clase A.`

B

Clase: `B(Substrato)`
Atributos: para cada objeto, mediante el constructor, se tiene:

```
self.nombre = ("B")
```

Métodos:

`producirOtrasBiomoleculas(self, option, usos, count_cycles)`:
Función que inicia la producción de otras biomoléculas, particularmente de las enzimas `F` y `FPrima`. Recibe una opción que es un `String` que indica si se catalizará para `"F"` o para `"FPrima"`, un entero que indica cuántos usos tendrá la enzima `FPrima` a producir (valor que no se usa en el caso de producir un substrato `F`), así como un entero que identifica al ciclo actual, utilizado para postfijar un identificador al nombre de las enzimas. La función optará por usar alguna de las funciones añadidas a la especificación original para producir ya sea el substrato o la enzima, y regresará un objeto `F` o un objeto `FPrima`.

`producirF(self)`:
Función añadida que regresa un objeto `F`.

`producirFPrima(self, usos, count_cycles)`:
Función añadida que recibe el límite de usos y contador de ciclos tomados por `producirOtrasBiomoleculas`. y que regresa un objeto `FPrima(usos, count_cycles)` configurado con el parámetro de usos y un identificador dado por el contador de ciclos.

`__init__(self)`:
`Constructor de objeto para la clase B.`

F

Clase: `F(Substrato)`
Atributos: para cada objeto, mediante el constructor, se tiene:

```
self.nombre = ("F")
```

Métodos:

`producirOtrasBiomoleculas(self, usos, count_cycles)`:
Función que inicia la producción de otras biomoléculas, particularmente de la enzima `Phi`. Recibe un entero que indica cuántos usos tendrá la enzima `Phi` a producir así como un entero que identifica al ciclo actual, utilizado para postfijar un identificador al nombre de las enzimas. La función regresará un objeto `Phi`.

`producirPhi(self, usos, count_cycles)`:
Función añadida que recibe el límite de usos y contador de ciclos tomados por

`producirOtrasBiomoleculas`. y que regresa un objeto `Phi(usos, count_cycles)` configurado con el parámetro de usos y un identificador dado por el contador de ciclos.

`__init__(self):`
`Constructor de objeto para la clase F.`

Beta

Clase: `Beta(Enzima)`
Atributos: para cada objeto, mediante el constructor, se tiene:

```
self.nombre = ("Beta" + str(count_cycles))
```

Métodos:

`catalizarSubstrato(self, ambienteEnzimaticoOrigen: [Enzima], ambienteEnzimaticoDestino: [Enzima], ambienteSubstrato: [Substrato], count_cycles)`:
Función que representa el comportamiento de catálisis de una enzima β sobre un substrato F para producir una enzima ϕ. Recibe como parámetros una lista de las enzimas del tipo correspondiente a la enzima que realiza la catálisis, una lista de las enzimas que corresponden al tipo de objeto que será producido si la catálisis es exitosa, una lista con el substrato adecuado para esta catálisis, así como un entero que identifica al ciclo actual, utilizado para postfijar un identificador al nombre de las enzimas. La función obtiene un substrato de la lista `ambienteSubstrato` mediante `pop()`. La función luego realizará la llamada a la función `self.catalizarReplicacion(ComidaF, count_cycles)` para producir la enzima ϕ. Luego, evaluará si la enzima utilizada se ha degradado (desnaturalizado) o no: en caso positivo, no la devolverá a `ambienteEnzimaticoOrigen`, y en caso negativo, la devolverá a esa lista para ser reutilizada después.
Finalmente, la función devolverá `ambienteEnzimaticoOrigen` (que podría o no tener de nuevo a la enzima que se usó para la catálisis), `ambienteEnzimaticoDestino` con la nueva enzima ϕ producida, y a `ambienteSubstrato` sin el substrato que se catalizó.
NOTA: La función tiene código comentado que dio servicio a un experimento en el que se inducía la incapacidad de una enzima para acoplarse a su substrato, en cuyo caso se devuelven las listas `ambienteEnzimaticoOrigen` reagregando la enzima que se iba a usar, `ambienteEnzimaticoDestino` sin cambios, y `ambienteSubstrato` con el substrato que se iba a usar. La elección para decidir si se realiza o no la simulación

de no-acople se realiza mediante la elección de `TRUE` o `FALSE` de manera equiprobable, donde `TRUE` indica que sí se realizará la catálisis y donde `FALSE` indica que no se realizará.

`catalizarReplicacion(self, f, count_cycles)`:
Función que cataliza específicamente al producto ϕ a partir del substrato F. Recibe como parámetros el substrato adecuado y el contador de ciclos. Devuelve un objeto `Phi` producido y configurado por la función `producirOtrasBiomoleculas` del objeto substrato `F`. Además, registra un uso en el contador de usos de la enzima que catalizó.
NOTA: La función tiene código comentado que dio servicio a un experimento en el que se inducía la elección aleatoria de la cantidad de usos para la enzima. Esta elección se realizaría sobre un rango (e.g. de 1 a 5 usos posibles), substituyendo así el máximo de usos, establecido por Zhang y cols. (2016) que limitaba la vida útil de una enzima a 3 reusos.

`__init__(self, reusos, count_cycles)`:
Constructor de objeto para la clase `Beta` que recibe como parámetro al límite de reusos de una enzima, así como al contador de ciclos para postfijarlo al nombre de nuevas enzimas.

FPrima

Clase: `FPrima(Enzima)`
Atributos: para cada objeto, mediante el constructor, se tiene:

```
self.nombre = ("FPrima" + str(count_cycles))
```

Métodos:

`catalizarSubstrato(self, ambienteEnzimaticoOrigen: [Enzima], ambienteEnzimaticoDestino: [Enzima], ambienteSubstrato: [Substrato], ambienteSubstratoNuevo: [Substrato], count_cycles)`:
Función que representa el comportamiento de catálisis de una enzima f' sobre un substrato A para producir B/β. Recibe como parámetros una lista de las enzimas del tipo correspondiente a la enzima que realiza la catálisis, una lista de las enzimas que corresponden al tipo de objeto que será producido si la catálisis es exitosa en la producción de una enzima, una lista con el substrato adecuado para esta catálisis, una lista de las substratos que corresponden al tipo de objeto que será producido

si la catálisis es exitosa en la producción de un substrato, así como un entero que identifica al ciclo actual, utilizado para postfijar un identificador al nombre de las enzimas. La función obtiene un substrato de la lista `ambienteSubstrato` mediante `pop()`. La función luego elegirá de manera aleatoria equiprobable si realiza la llamada a la función `self.catalizarMetabolismoB(ComidaA)` para producir un substrato B, o si realiza la llamada a la función `self.catalizarMetabolismoBeta(ComidaA, count_cycles)` para producir la enzima β. Luego, evaluará si la enzima utilizada se ha degradado (desnaturalizado) o no: en caso positivo, no la devolverá a `ambienteEnzimaticoOrigen`, y en caso negativo, la devolverá a esa lista para ser reutilizada después. Finalmente, la función devolverá `ambienteEnzimaticoOrigen` (que podría o no tener de nuevo a la enzima que se usó para la catálisis), una lista destino llamada `ambienteEnzimaticoDestino` con la nueva enzima producida si es que se catalizó para β, `ambienteSubstrato` sin el substrato que se catalizó, y a `ambienteSubstratoNuevo` con el nuevo substrato producido si es que se catalizó para B.

NOTA: La función tiene código comentado que dio servicio a un experimento en el que se inducía la incapacidad de una enzima para acoplarse a su substrato, en cuyo caso se devuelven las listas `ambienteEnzimaticoOrigen` reagregando la enzima que se iba a usar, `ambienteEnzimaticoDestino` sin cambios, `ambienteSubstrato` con el substrato que se iba a usar, y a `ambienteSubstratoNuevo` sin cambios. La elección para decidir si se realiza o no la simulación de no-acople se realiza mediante la elección de `TRUE` o `FALSE` agregando estos casos a la elección aleatoria que sirve para decidir si se producirá B o β, donde `TRUE` indica que se realizará la catálisis para B, donde `FALSE` indica que se realizará para β, y donde `NONE` indica que no se realizará la catálisis. También hay código comentado para forzar la realización de la catálisis de ambos productos en el mismo uso de la enzima, lo cual también dio servicio a un experimento en el que no se hizo una distinción entre la producción de B y β.

`catalizarMetabolismoB(self, a)`:

Función que cataliza específicamente al producto B a partir del substrato A. Recibe como parámetros el substrato adecuado. Devuelve un objeto `B` producido y configurado por la función `producirOtrasBiomoleculas` del objeto substrato `A`.

`catalizarMetabolismoBeta(self, a, count_cycles)`:

Función que cataliza específicamente al producto β a partir del substrato A. Recibe como parámetros el substrato adecuado y el contador de ciclos. Devuelve un objeto `Beta` producido y configurado por la función `producirOtrasBiomoleculas` del ob-

jeto substrato `A`. Además, registra un uso en el contador de usos de la enzima que catalizó.
NOTA: La función tiene código comentado que dio servicio a un experimento en el que se inducía la elección aleatoria de la cantidad de usos para la enzima. Esta elección se realizaría sobre un rango (e.g. de 1 a 5 usos posibles), substituyendo así el máximo de usos, establecido por Zhang y cols. (2016) que limitaba la vida útil de una enzima a 3 reusos.

`__init__(self, reusos, count_cycles)`:
Constructor de objeto para la clase `FPrima` que recibe como parámetro al límite de reusos de una enzima, así como al contador de ciclos para postfijarlo al nombre de nuevas enzimas.

Phi

Clase: `Phi(Enzima)`
Atributos: para cada objeto, mediante el constructor, se tiene:

```
self.nombre = ("Phi" + str(count_cycles))
```

Métodos:

`catalizarSubstrato(self, ambienteEnzimaticoOrigen: [Enzima], ambienteEnzimaticoDestino: [Enzima], ambienteSubstrato: [Substrato], ambienteSubstratoNuevo: [Substrato], count_cycles)`:
Función que representa el comportamiento de catálisis de una enzima ϕ sobre un substrato B para producir f/f'. Recibe como parámetros una lista de las enzimas del tipo correspondiente a la enzima que realiza la catálisis, una lista de las enzimas que corresponden al tipo de objeto que será producido si la catálisis es exitosa en la producción de una enzima, una lista con el substrato adecuado para esta catálisis, una lista de las substratos que corresponden al tipo de objeto que será producido si la catálisis es exitosa en la producción de un substrato, así como un entero que identifica al ciclo actual, utilizado para postfijar un identificador al nombre de las enzimas. La función obtiene un substrato de la lista `ambienteSubstrato` mediante `pop()`. La función luego elegirá de manera aleatoria equiprobable si realiza la llamada a la función `self.catalizarReparacionF(ComidaB)` para producir un substrato F, o si realiza la llamada a la función `self.catalizarReparacionFPrima(ComidaB, count_cycles)`

para producir la enzima $'$. Luego, evaluará si la enzima utilizada se ha degradado (desnaturalizado) o no: en caso positivo, no la devolverá a `ambienteEnzimaticoOrigen`, y en caso negativo, la devolverá a esa lista para ser reutilizada después. Finalmente, la función devolverá las listas `ambienteEnzimaticoOrigen` (que podría o no tener de nuevo a la enzima que se usó para la catálisis), `ambienteEnzimaticoDestino` con la nueva enzima producida si es que se catalizó para f', `ambienteSubstrato` sin el substrato que se catalizó, y a `ambienteSubstratoNuevo` con el nuevo substrato producido si es que se catalizó para f.

NOTA: La función tiene código comentado que dio servicio a un experimento en el que se inducía la incapacidad de una enzima para acoplarse a su substrato, en cuyo caso se devuelven las listas `ambienteEnzimaticoOrigen` reagregando la enzima que se iba a usar, `ambienteEnzimaticoDestino` sin cambios, `ambienteSubstrato` con el substrato que se iba a usar, y a `ambienteSubstratoNuevo` sin cambios. La elección para decidir si se realiza o no la simulación de no-acople se realiza mediante la elección de `TRUE` o `FALSE` agregando estos casos a la elección aleatoria que sirve para decidir si se producirá f o f', donde `TRUE` indica que se realizará la catálisis para f, donde `FALSE` indica que se realizará para f', y donde `NONE` indica que no se realizará la catálisis. También hay código comentado para forzar la realización de la catálisis de ambos productos en el mismo uso de la enzima, lo cual también dio servicio a un experimento en el que no se hizo una distinción entre la producción de f y f'.

`catalizarReparacionF(self, b)`:

Función que cataliza específicamente al producto f a partir del substrato B. Recibe como parámetros el substrato adecuado. Devuelve un objeto `F` producido y configurado por la función `producirOtrasBiomoleculas` del objeto substrato `B`.

`catalizarReparacionFPrima(self, b, count_cycles)`:

Función que cataliza específicamente al producto f' a partir del substrato B. Recibe como parámetros el substrato adecuado y el contador de ciclos. Devuelve un objeto `FPrima` producido y configurado por la función `producirOtrasBiomoleculas` del objeto substrato `B`. Además, registra un uso en el contador de usos de la enzima que catalizó.

NOTA: La función tiene código comentado que dio servicio a un experimento en el que se inducía la elección aleatoria de la cantidad de usos para la enzima. Esta elección se realizaría sobre un rango (e.g. de 1 a 5 usos posibles), substituyendo así el máximo de usos, establecido por Zhang y cols. (2016) que limitaba la vida útil de una enzima a 3 reusos.

`__init__(self, reusos, count_cycles)`:
Constructor de objeto para la clase `Phi` que recibe como parámetro al límite de reusos de una enzima, así como al contador de ciclos para postfijarlo al nombre de nuevas enzimas.

5.5.3. Paso 3: Simulaciones

Un aspecto fundamental en el trabajo de Zhang y cols. (2016) fue la delegación del trabajo de programación y simulación del modelo UML de los $Sistemas-(M, R)$, por lo que se produjeron una serie de experimentos con diferentes parámetros. El trabajo se realizó en torno a tres tipos de simulación, dependiendo de su configuración: (1) la establecida por Zhang y cols. (2016), (2) una simulación estocástica inspirada en la realizada en Bio-PEPA por Gatherer & Galpin (2013), y una estocástica configurada sin la diferencia entre la producción entre B y β así como entre f y f'.

Experimento 1:

El primer experimento se configuró de la siguiente manera:

- Ambiente inicial de enzimas: `FPrima = 1000, Beta = Phi = 0`
- Ambiente inicial de substratos: `A = 1000, F = B = 0`
- Elección aleatoria en la producción entre `FPrima` y `F` por parte de `Phi`
- Elección aleatoria en la producción entre `Beta` y `B` por parte de `FPrima`
- 10,000 pasos (ciclos)

Debido a que Zhang y cols. (2016) no establecen una población inicial para todas las biomoléculas (tanto las enzimas **Beta/B FPrima/F** internas al sistema así como el alimento **A**), se utilizó la misma que se muestra en los experimentos realizados en Bio-Pepa (Gatherer y Galpin, 2013). La cantidad de ciclos a ejecutar se eligió de manera arbitraria, suficiente para producir gráficas que exhibiesen comportamientos graficables. Se realizaron un total de diez simulaciones, cuyo número también fue elegido de manera arbitraria.

Experimento 2:

El segundo experimento se configuró de la siguiente manera:

- Ambiente inicial de enzimas: Aleatorio entre 0 y 1000 para `FPrima, Beta, Phi`
- Ambiente inicial de substratos: Aleatorio entre 0 y 1000 para `A, B, F`
- Elección aleatoria en la producción entre `FPrima` y `F` por parte de `Phi`
- Elección aleatoria en la producción entre `Beta` y `B` por parte de `FPrima`
- 10,000 pasos (ciclos)

La simulación estocástica está inspirada en la misma realizada en el artículo de Gatherer & Galpin (2013) en el que se buscó establecer diversas trayectorias para el modelo. Se utilizó como cota superior el número sugerido para la población inicial de f' y A del Experimento 1 (`FPrima = A = 1000`). Se eligió el mismo número de ciclos a ejecutar que el que se utilizón en el Experimento 1. Se realizaron un total de diez simulaciones, cuyo número fue elegido de manera arbitraria.

Experimento 3:

En el experimento tres se realizaron algunos sub-experimentos. Para el primero, se estableció la configuración de la siguiente manera:

- Ambiente inicial de enzimas: `FPrima = 1000, Beta = Phi = 0`
- Ambiente inicial de substratos: `A = F = 1000, B = 0`
- Producción simultánea de `FPrima` y `F` por parte de `Phi`
- Producción simultánea de `Beta` y `B` por parte de `FPrima`
- 10,000 pasos (ciclos)

Rosen no hace una distinción entre β y B en tanto entidades, sino entre la funcionalidad de una misma entidad, por lo que es posible interpretar el $Sistema - (M, R)$ de forma en que se produzcan dos objetos de manera simultánea: el objeto `B` y el objeto `Beta` (y así, también, para los objetos `F` y `FPrima`). Por esta causa es que la población inicial de la enzima `FPrima` coincide, en este experimento, con la población inicial de `F`. Se realizó un experimento con parámetros fijos (siendo un experimento

determinista sin necesidad de muestreo).

El segundo sub-experimento introdujo reusos estocásticos para cada enzima:

- Ambiente inicial de enzimas: `FPrima = 1000, Beta = Phi = 0`
- Ambiente inicial de substratos: `A = F = 1000, B = 0`
- Producción simultánea de `FPrima` y `F` por parte de `Phi`
- Producción simultánea de `Beta` y `B` por parte de `FPrima` Reusos aleatorios entre 1 y 3 para cada enzima `FPrima, Beta, Phi`.
- 10,000 pasos (ciclos)

Los reusos aleatorios permiten producir condiciones en las que las enzimas no se acumulan de manera simétrica por lo que así alguna enzima podría desnaturalizarse más rápido que otra. Se realizaron 6 simulaciones de este tipo, cuyo número fue elegido de manera arbitraria.

El tercer sub-experimento introduce un factor de error en el uso de una enzima, así como su desnaturalización.

- Ambiente inicial de enzimas: `FPrima = 1000, Beta = Phi = 0`
- Ambiente inicial de substratos: `A = F = 1000, B = 0`
- Producción simultánea de `FPrima` y `F` por parte de `Phi`
- Producción simultánea de `Beta` y `B` por parte de `FPrima` Reusos aleatorios entre 1 y 3 para cada enzima `FPrima, Beta, Phi`.
- Probabilidad (0.2) de que cualquier enzima se vea imposibilitada a catalizar en algún paso de la simulación, para cada enzima `FPrima, Beta, Phi`.
- Remoción de una enzima cada 7 ciclos, para cada enzima `FPrima, Beta, Phi`.
- 10,000 pasos (ciclos)

En primer lugar, el error de uso se implementa para simular que una enzima y un substrato no pudieron acoplarse al sitio activo por cambios en condiciones contextuales (temperatura, pH, etc.), lo cual permite presentar a estas enzimas como

falibles. Además, se introduce una remoción de enzimas cada siete ciclos permite simular cambios en el contexto que desnaturalizan por completo a las enzimas. Estas aproximaciones se instalan en la simulación para sustituir aspectos de la bioquímica que Zhang y cols. (2016) no consideraron en su modelo UML. Esto será discutido en el análisis de resultados. Se realizaron 6 simulaciones de este tipo, cuyo número fue elegido de manera arbitraria.

Finalmente, el cuarto sub-experimento reintroduce el factor estocástico para las poblaciones iniciales de enzimas y substratos como se realizó en el experimento 2.

- Ambiente inicial de enzimas: Aleatorio entre 0 y 1000 para `Phi`
- Ambiente inicial de substratos: Aleatorio entre 0 y 1000 para `A`
- Ambiente inicial: Aleatorio entre 0 y 1000 para `B, Beta`
- Ambiente inicial: Aleatorio entre 0 y 1000 para `F, FPrima`
- Producción simultánea de `FPrima` y `F` por parte de `Phi`
- Producción simultánea de `Beta` y `B` por parte de `FPrima` Reusos aleatorios entre 1 y 3 para cada enzima `FPrima, Beta, Phi`.
- Probabilidad (0.2) de que cualquier enzima se vea imposibilitada a catalizar en algún paso de la simulación, para cada enzima `FPrima, Beta, Phi`.
- Remoción de una enzima cada 7 ciclos, para cada enzima `FPrima, Beta, Phi`.
- 10,000 pasos (ciclos)

En este último ejercicio se reintroduce diversos estados iniciales para explorar diferentes trayectorias del sistema auxiliadas por las consideraciones de los sub-experimentos anteriores, con la salvedad de que se asigna el mismo número, electo aleatoriamente, a las poblaciones iniciales de `F` y `FPrima`, y de igual manera se asigna un mismo número aleatorio a las poblaciones iniciales de `B` y `Beta`. Se realizaron 6 simulaciones de este tipo, cuyo número fue elegido de manera arbitraria. Auxiliarmente se realizó otra simulación adicional: una en la que, durante algunos pasos del ciclo, se deja sin alimento A al sistema. Esta simulación será discutida en el análisis de resultados.

6 Análisis exploratorio de la implementación

6.1. Resultado de las simulaciones

El **primer experimento**, configurado con los parámetros establecidos por Zhang y cols. (2016), produce comportamientos en los que se observa que la producción de enzimas ϕ no se levanta del mínimo, mientras que la producción de enzimas f' cae después de algunos pasos, usualmente cercanos al ciclo 3000. Esto causa que el sistema deje de funcionar para auto-repararse, produciendo que caiga en un estado de quietud permanente.

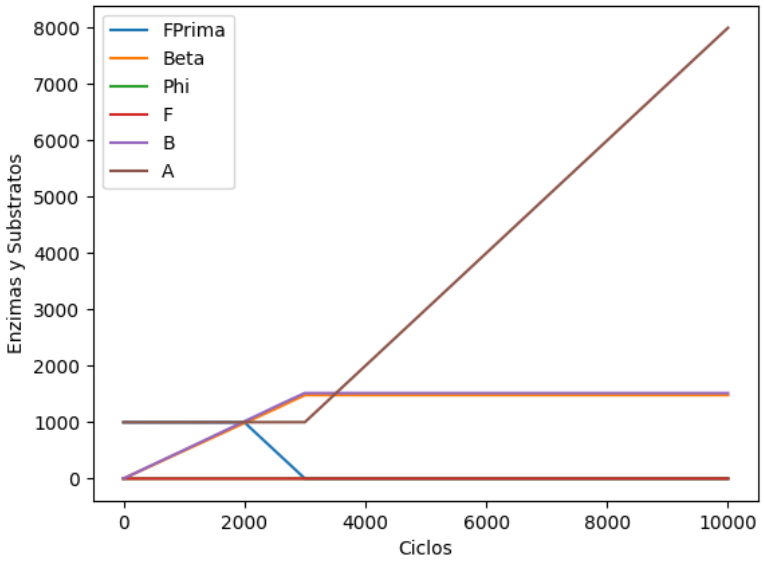


Figura 6.1: Promedio del comportamiento del $Sistema - (M, R)$ en una simulación bajo los parámetros establecidos por Zhang y cols. (2016). Se muestra el comportamiento de las enzimas.

Este comportamiento, que representa la muerte del $Sistema - (M, R)$, fue destacado por Gatherer & Galpin (2013) en algunas simulaciones realizadas en Bio-PEPA, y lo vincularon con la fragilidad del sistema ante diversas condiciones iniciales. En la figura B.1 se muestra el resultado de una de las simulaciones, mientras que en la figura 6.1 se muestra el promedio del comportamiento de las simulaciones.

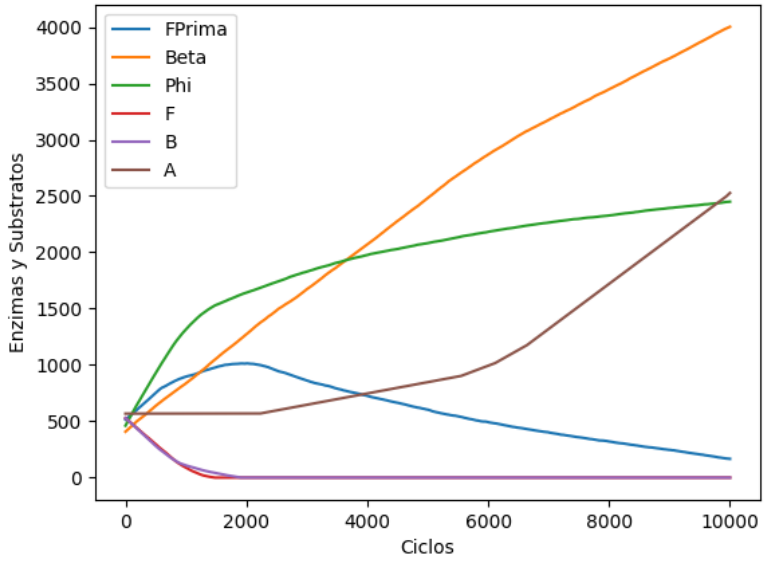


Figura 6.2: Promedio del comportamiento del $Sistema - (M, R)$ en una simulación bajo los parámetros establecidos por Zhang y cols. (2016) agregando la estocasticidad en la elección de las condiciones iniciales. Se muestra el comportamiento de todas las biomoléculas.

El **segundo experimento** comprendió la inyección del factor estocástico en la elección de los estados iniciales de todas las biomoléculas. En este caso es posible observar que el sistema es capaz de mantener cierta estabilidad, aunque la producción de f' cae con el tiempo mientras que la producción de β y ϕ tiende a crecer. En algunas simulaciones la producción de f' cae tempranamente por lo que el sistema cae en un estado de quietud, mientras que en otras simulaciones la producción de f' cae más lentamente. Nuevamente, la sensibilidad del $Sistema - (M, R)$ indicada por Gatherer & Galpin (2013) a las condiciones iniciales podría ser la responsable del comportamiento tendiente a la quietud del sistema, aunque es destacable identificar que entre los pasos 0 y 2000 suele haber una tendencia al crecimiento. En la figura B.2 se muestra el resultado de una de las simulaciones, mientras que en la figura 6.2 se muestra el promedio del comportamiento de las simulaciones.

El **tercer experimento** comprende cuatro tipos de simulaciones. En la **primer tipo de simulación**, se establece que la producción de f deberá coincidir con la de f' mientras que la de B deberá coincidir con la de β. Esta elección respeta la identidad de las biomoléculas, según las describió Rosen en su diagrama. Debido a que todas las condiciones iniciales se fijan, y que no hay factores estocásticos involucrados en otros pasos, la simulación se vuelve determinista. La figura B.3 muestra el comportamiento de esta simulación, en el cual se observa que el sistema crece de manera indeterminada, algo esperable de la vida (y observado en la naturaleza, por ejemplo en el cáncer) si no se imponen restricciones al crecimiento.

En el **segundo tipo de simulación**, se respeta el modo de producción de las enzimas mientras que se agrega la elección aleatoria para los reusos de las enzimas a lo largo del programa, permitiendo aproximar a enzimas que podrían desnaturalizarse más rápido que otras por diversas condiciones impuestas por el ambiente. En la figura 6.3 se muestra el promedio del comportamiento de las 6 simulaciones realizadas con esta configuración. Estas simulaciones mostraron, nuevamente, un $Sistema - (M, R)$ que tiende al crecimiento.

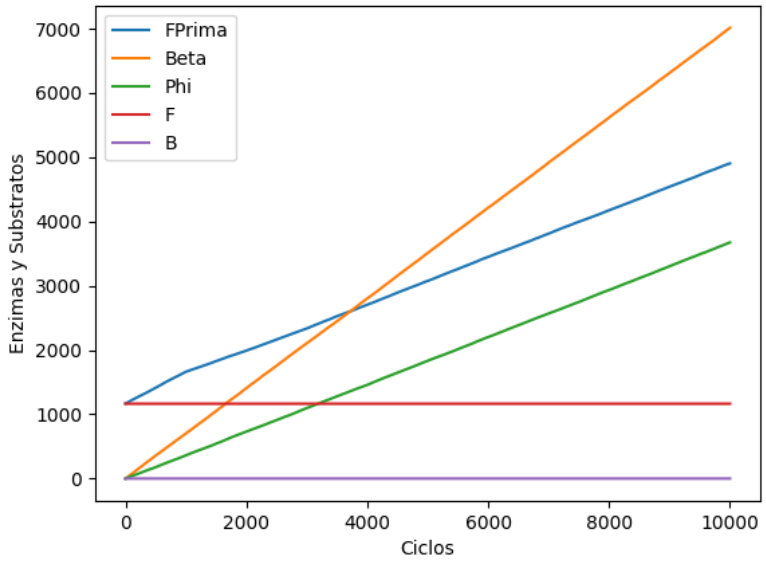


Figura 6.3: Promedio del comportamiento del $Sistema - (M, R)$ en una simulación en la que se agrega la estocasticidad en la elección de reusos. Se muestra el comportamiento de todas las biomoléculas salvo el del alimento A.

En el **tercer tipo de simulación** se introduce el error a la hora de que la enzima quiera catalizar un substrato, así como la desnaturalización de enzimas cada 7 ciclos. Al igual que en el sub-experimento anterior, el objeto de introducir estas condiciones es producir situaciones en las que el contexto de las enzimas no les sea totalmente amigable y les imponga presión. La figura 6.4 muestra el comportamiento de crecimiento del sistema, a pesar de las presiones causadas por el descarte de enzimas.

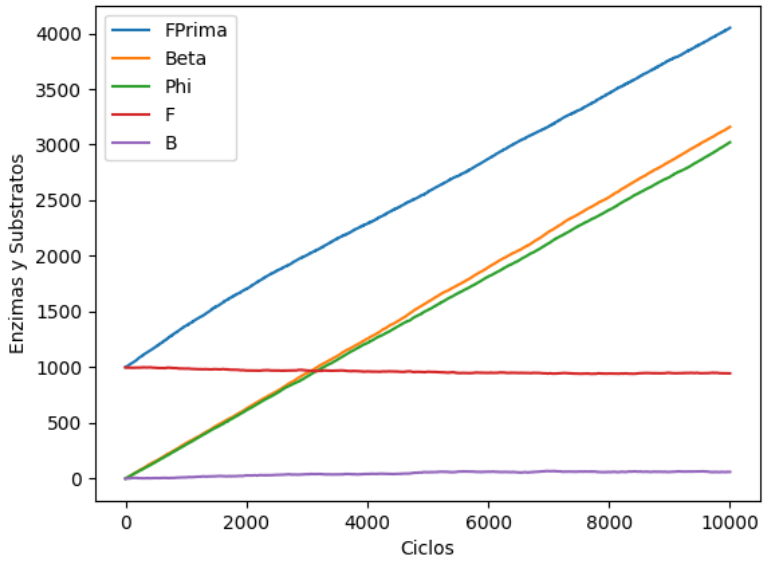


Figura 6.4: Promedio del comportamiento del $Sistema - (M, R)$ en una simulación en la que se inducen presiones a partir de remoción de enzimas y fallas de catálisis. Se muestra el comportamiento de todas las biomoléculas salvo el del alimento A.

En el **cuarto tipo de simulación**, se agrega la estocasticidad de condiciones iniciales utilizada en el experimento 2 para evaluar diversas trayectorias del sistema. La figura 6.5 muestra el comportamiento promedio de las seis simulaciones: crecimiento del sistema a mayor velocidad conforme el alimento incrementa. En general, todas las simulaciones de este tipo mostraron tendencia al crecimiento, a pesar de las presiones diseñadas en y preservadas desde las simulaciones anteriores de este experimento.

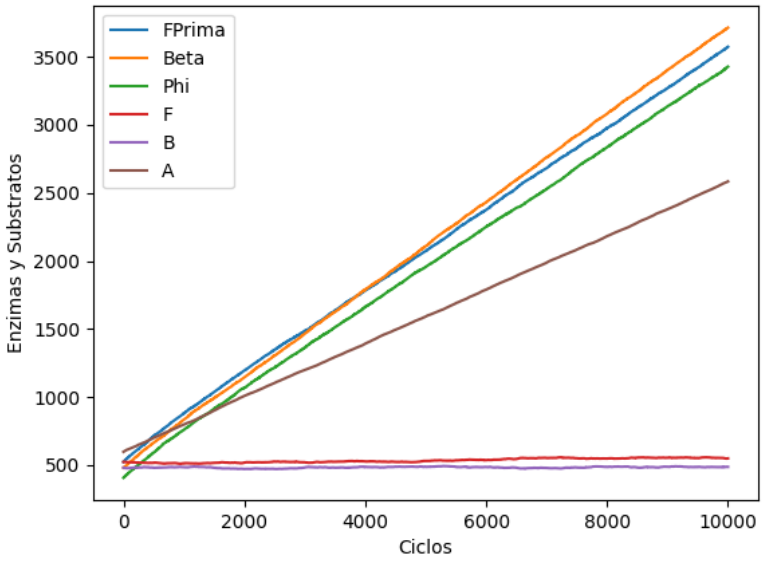


Figura 6.5: Promedio del comportamiento del $Sistema - (M, R)$ en una simulación en la que se agrega la construcción aleatoria de las condiciones iniciales para las biomoléculas. Se muestra el comportamiento de todas las biomoléculas.

6.2. Análisis de resultados y de la implementación

En primera instancia, conviene revisar los resultados experimentales a la luz del trabajo realizado por Gatherer & Galpin (2013) sobre Bio-PEPA. Los resultados exhibidos en este trabajo muestran que la implementación del diagrama UML de Zhang y cols. (2016), configurado según los autores y utilizando las condiciones iniciales sugeridas por Gatherer & Galpin (2013) producen comportamientos de un sistema que se caracteriza por el silencio: no pasa mucho tiempo antes de que el sistema deje de funcionar y la comida A se acumule, lo cual es el resultado del experimento 1. En tal experimento es posible observar que en promedio entre el ciclo 2000 y el ciclo 3000 el $Sistema - (M, R)$ es incapaz de sostener su invarianza organizacional, primero por una caída en la producción de la enzima f' y seguido después por una inactividad en la enzima β. Además, la enzima ϕ se produce en cantidades mínimas. Así, la tendencia de el $Sistema - (M, R)$ en el experimento 1 es a la quieto (o muerte). En el experimento 2, en el que se añade el factor estocástico, el crecimiento viene seguido de un colapso en la acumulación de la enzima f'. Es posible apoyarse del argumento de Gatherer & Galpin (2013) para decir que estos comportamientos pueden deberse a la sensibilidad de los $Sistemas - (M, R)$ a las condiciones iniciales, pero es difícil

apuntar hacia esa dirección al considerar que el programa, implementado a partir de los diagramas UML de Zhang y cols. (2016), no considera la granularidad y el rigor bioquímico que sí considera Bio-PEPA.

El conjunto de simulaciones del experimento 3 intentan poner a prueba el programa utilizando una interpretación más cercana al diagrama de Rosen (Rosen, 1991), una donde la producción de β y B, así como de f y f', ocurran a la par. Con ese cambio único es posible ver, en la figura B.3, un comportamiento tendiente al crecimiento conforme el alimento aumenta. Esto coincide con los resultados experimentales reportados por Everardo Robredo (comunicación personal, 18 de abril, 2022) producidos mediante análisis de Ecuaciones Diferenciales en los cuales se muestra que una taza constante de alimento A producirá un sistema de punto fijo, mientras que incrementos en el alimento A producirán un sistema que tiende al crecimiento.
Para entender si el $Sistema - (M, R)$ programado es capaz o no de recobrarse de la falta de alimento y mantener su invarianza organizacional, se produjo una simulación auxiliar que se muestra en la figura B.4. Es posible notar en tal figura que el $Sistema - (M, R)$ se repone de la situación de hambruna, aunque después de un tiempo la producción de substratos B y f decae, a pesar de ser producidos a la par de las enzimas β y f'. Esto indica que la simulación trata sobre un sistema enzimático que es capaz de mantenerse mayormente invariante ante la falta de alimento, y además mantener una tendencia al crecimiento a pesar de que el alimento no aumente en cantidad pero que no exhibe el comportamiento esperado en la producción de los pares enzima-substrato.

Resumiendo, algunos problemas con estos resultados se destacan a continuación:

1. En el experimento 1, que considera los parámetros tal cual son provistos por Zhang y cols. (2016), no existe una instancia en la que el $Sistema - (M, R)$ pueda mantenerse viva, perdiendo usualmente a la enzima f'.

2. En el experimento 2, a pesar de buscar analizar distintas trayectorias, es posible observar una tendencia a la baja en producción de la enzima f', coincidiendo con el comportamiento del experimento 1.

3. En las simulaciones del experimento 3, existen momentos en los que la producción de β y B no coinciden, así como ocurre con la producción de f' y f: en todas las simulaciones y sus promedios es posible ver que, mientras las enzimas se acumulan y crecen, los substratos se producen de manera semi-constante.

Debido a que en el experimento 3 el principal factor de cambio se centra en el modo de producción de los pares de β y B así como de f' y f, es posible conjeturar que el comportamiento indeseable de los experimentos ejecutados con los parámetros de Zhang y cols. (2016) y de Gatherar & Galpin (2013) se deba a la producción separada entre los elementos de dichos pares. Por otro lado, mientras que en el experimento tres la producción de elementos de dichos pares — que ocurre de manera simultánea — permite obtener comportamientos de crecimiento, las acumulaciones entre elementos de los pares no coinciden. Esto se debe, potencialmente, a un error conceptual a la hora de construir el diagrama de clases de este programa: separar en objetos, con atributos diferentes, a los pares. En el diagrama de Rosen se considera que β es una característica de la biomolécula B, y no se considera que existan de manera separada y mucho menos que tengan propiedades diferentes (lo mismo ocurre con la distinción de f y f'). En el diagrama UML de Zhang y cols. (2016), las enzimas son las únicas que tienen un atributo que les permite llevar un registro sobre sus usos para poder ser descartadas después de sobrepasar tales usos. Esta interpretación produce un modelo en el que B no tiene salvo un uso (ser consumida) mientras que β tiene 3 reusos. Así, una B_i se agotará siempre antes que su correspondiente β_i, producida en el $i-$ésimo paso de la simulación.

También es problemático el haber introducido un contador de usos a las enzimas (y pensar ahora, además, en la potencial necesidad de añadirlo a los substratos correspondientes). Las enzimas reales no tienen contadores internos para llevar un registro sobre sus usos, sino que a partir de sus usos van dañándose de manera no-determinista a lo largo de los ciclos catalíticos a los que se someten, ya sea debido a daños colaterales en los sitios donde la enzima acopla al substrato a causa de reacciones químicas ocurridas en la catálisis, o bien por la interferencia, en esos mismos sitios, de residuos reactivos no-catalíticos (Hanson y cols., 2021). Aunque estas dinámicas fueron aproximadas en las simulaciones del experimento 3, es fácil observar que su prolijidad no es suficiente para capturar las complejas dinámicas de la bioquímica. Es posible que la forma en la que se construyeron la clase `Enzima` y sus subclases haya inducido la inyección de un atributo que degenera su dinámica en un experimento, por lo que es posible que el análisis orientado a objetos para la clase `Enzima` no haya sido satisfactorio. Además, agregar el contador induce un el problema de regresión infinita del sensor en la cibernética: el contador modula a la enzima como proceso interno, así que, ¿quién modula al contador? ¿Otro contador? ¿Y a tal contador quién lo modula? ¿Otro contador? ¿Y a tal contador quién lo modula? *Et sīc ad nauseam.*

Por otro lado, a partir del comportamiento de las simulaciones del experimento 3 es

posible argumentar que el análisis orientado a objetos que produjo clases separadas para el par B y β y para el par f y f' tampoco fue satisfactorio, no sólamente en términos experimentales, sino en los conceptos impresos por Rosen en el diagrama del $Sistema - (M, R)$ (Rosen, 1991).

También es posible observar que, mientras la condición inicial referente al alimento A es un morfismo externo (regulado externamente) al diagrama de Rosen, las poblaciones iniciales de β, f', ϕ y sus respectivos b y f son morfismos internos. Las poblaciones iniciales de estos morfismos dadas para un organismo están dadas por la epigenética, la cual ayuda a determinar qué genes se encienden y qué genes no, y por lo tanto qué y cuántas proteínas se producen, entre ellas las enzimas (Dupont, Armant, y Brenner, 2009). Zhang y cols. (2016) no establecen condiciones iniciales para el potencial experimento implementado, por lo cual este trabajo tomó inspiración en la publicación de Gatherer & Galpin (2013) en la cual se utilizan ciertas condiciones iniciales en un ambiente de álgebra de procesos diseñado para el análisis de reacciones bioquímicas. En los experimentos de este trabajo recepcional no fue posible introducir, por ejemplo, el principio de Le Chatelier ni velocidades de reacción, aspectos de la bioquímica que sí fueron considerados en el experimento realizado en Bio-PEPA. No obstante esto, en el experimento realizado en Bio-PEPA las únicas poblaciones inicializadas fueron las de A y las de f, por lo que tampoco es posible apreciar un modelado de la epigenética de algún organismo en particular para realizar la simulación. En el caso de la implementación orientada a objetos, nada sobre el contexto en el cual el $Sistema - (M, R)$ habría de desenvolverse fue dado por Zhang y cols. (2016), lo cual podría contribuir a los comportamientos indeseables obtenidos durante las simulaciones, mismos que potencialmente tendrían que haber sido establecidos en los diagramas de actividad y comunicación.

Finalmente, es adecuado mencionar que un ambiente enzimático no contiene reacciones que ocurran de forma secuencial, sino que varias enzimas podrían estar catalizando a un substrato a la vez (Hammes, 2008). En el caso de el programa derivado de los diagramas UML de Zhang y cols. (2016), esto no ocurre: primero se realiza $f' : A \mapsto B/\beta$, luego se realiza $\phi : B \mapsto f/f'$ y al final se realiza $\beta : f \mapsto \phi$ antes de volver a empezar. Introducir el paralelismo es una de las sugerencias que se plantea en la publicación de Palmer y cols. (2016), aunque no en la de Zhang y cols. (2016). Es posible que UML haya impuesto una restricción semántica al modelo que no está comprendida no sólamente en el diagrama de Rosen, sino en las operaciones

bioquímicas que ocurren en los organismos mismos, por lo cual una representación secuencial de los $Sistemas - (M, R)$ podría ser producto de un análisis orientado a objetos no satisfactorio.

Por lo tanto, se observa que la implementación en un lenguaje orientado a objetos del diagrama UML producido por Zhang y cols. (2016) que pretende demostrar la Turing-computabilidad de los $Sistemas - (M, R)$ altera aspectos estructurales y semánticos de los $Sistemas - (M, R)$ y carece de definiciones suficientes para incorporar al modelo en un espacio experimental adecuado a los requerimientos de la bioquímica y de las dinámicas enzimáticas, por lo cual el objeto de estudio de los experimentos será otro diferente a los $Sistemas - (M, R)$ de Robert Rosen. Las alteraciones realizadas al formalismo de Rosen pueden ser identificadas como **corrupciones**, en los términos utilizados por Zhang y cols. (2016).

7 Conclusiones

7.1. Generales sobre esta tesis

Esta tesis inicia con una revisión de los antecedentes históricos del problema de definir la vida así, partiendo desde la Grecia Antigua y llegando a la Biología Molecular. En ese punto se ancla el formalismo propuesto por Robert Rosen, sus definiciones (las generadas por el mismo Rosen así como aquellas desarrolladas por otros autores) así como los alcances que éste tiene dentro y fuera de la teoría de la computación. Posteriormente se introdujeron los conceptos de la Teoría de la Computación necesarios para entender las consecuencias del trabajo de Rosen, así como tecnicismos que él usó en el momento de establecer la incomputabilidad de los $Sistemas - (M, R)$.
Después, se realizó una revisión de tres diferentes trabajos de investigación desarrollados recientemente que se enfocan en la búsqueda de una prueba experimental sobre la Turing-computabilidad de los sistemas-(M,R). El primer trabajo revisado (Palmer y cols., 2016) versó sobre el uso de Máquinas-X Comunicantes que sugieren una capacidad computacional para capturar la estructura auto-referencial de los $Sistemas - (M, R)$. El segundo (Gatherer y Galpin, 2013) trató sobre una simulación computacional realizada en un Álgebra de Procesos (Bio-PEPA), enfocada en identificar comportamientos similares a los de la vida dentro de un ambiente experimental computacional. El último (Zhang y cols., 2016) discutió las implicaciones de llevar al diagrama de Rosen al Lenguaje Unificado de Modelado (UML) dentro de la búsqueda por falsear la tesis de Rosen sobre la incomputabilidad de los $Sistemas - (M, R)$.
Posteriormente se estudió a detalle el trabajo de Zhang y cols. (2016), estableciendo primero los rudimentos necesarios para entender UML y posteriormente explicitando los diagramas producidos por los autores, así como sus procesos constructivos.
Se implementó, inmediatamente, el programa derivado de los diagramas UML de Zhang y cols. (2016) con el objetivo de someterlo a simulaciones y observar su comportamiento. El programa fue puesto a prueba en tres experimentos: uno en el que se

usaron los parámetros dados por Zhang y cols. (2016) y Gatherer & Galpin (2013), otro en el que se introdujo un factor estocástico a la hora de construir las condiciones iniciales, y otro en el que se acotaron las condiciones experimentales a una interpretación más fiel del formalismo de Rosen.
Con las simulaciones ejecutadas, se analizaron los resultados y se observaron comportamientos que si bien tendían al crecimiento, no produjeron enzimas de la forma esperada, a partir de la interpretación dura del diagrama de Rosen. Los resultados sugieren que hubo errores en los procesos de análisis orientado a objetos que Zhang y cols. (2016) utilizaron para derivar los diagramas UML sobre los cuales se implementó el programa experimental.

El trabajo de Zhang y cols. (2016) es un paso sumamente relevante en la búsqueda de una prueba empírica sobre la Turing-computabilidad de los $Sistemas - (M, R)$ ante los diversos argumentos matemáticos — que Aloisius Louie (2009) establece como demostraciones al respecto de esta cuestión [1] — debido a que continúa abriendo caminos en la práctica de la programación y no únicamente en la discusión matemática. El conjunto de publicaciones revisadas en este trabajo recepcional comparten, todas, lo co-autoría de Derek Gatherer, y a su vez proponen el mismo problema de interpretación: Rosen no dejó bases claras para la universalidad conceptual de los $Sistemas - (M, R)$, por lo que hay aspectos que pueden someterse a la licencia creativa o abductiva del investigador. Uno de estos aspectos es la separación de β y B en entidades discretizadas, así como de f' y f. En este sentido, los autores de las tres publicaciones exploran formas de lidiar con la dualidad de lo que Rosen definió como una sola biomolécula, culminando en un manejo independiente: tanto en la publicación centrada en UML (Zhang y cols., 2016) como en la publicación centrada en Máquinas-X (Palmer y cols., 2016) se establece una distinción a nivel de instancias de objetos entre los duales de enzima-substrato. Esta interpretación no corresponde a lo que Rosen (1991) establece como formalismo a partir de la realidad bioquímica. Aloisius Louie no produce una explicación formal suficiente para entender de manera puntual la dualidad en términos matemáticos, pero sostiene (2009) que tal dualidad existe. Más, todavía, la dualidad ha sido extendida recientemente por Everardo Robredo (comunicación personal, 18 de abril, 2022) para la entidad ϕ, misma que debería ser acompañada por su dual ϕ' que no es otra cosa sino una cara diferente de la misma moneda. A partir de esto, es posible considerar que una distinción a nivel de objetos

[1] Luz-Cárdenas y cols. (2010) ya también habrían propuesto instrumentos matemáticos para sostener el argumento de la incomputabilidad de los $Sistemas - (M, R)$

no es adecuada: es posible repensar entonces los diagramas UML para dotar a una misma biomolécula, e.g. β, de funciones diferentes dependiendo del rol que tengan en el juego metabólico.

Por otro lado, en la publicación de Gatherer & Galpin (2013) los autores mencionan que los experimentos deterministas son, en su opinión, los más cercanos a las implicaciones contenidas en el diagrama de Rosen y construidas a partir de la definición de un diagrama relacional. Mientras que las implicaciones contenidas al interior de los $Sistemas - (M, R)$ sí determinan, según el diagrama de Rosen, consecuencias y causalidad, el ambiente en el que se situaría la realización física de un $Sistema - (M, R)$, dígase un organismo unicelular, no es determinista: todos los factores ambientales a los que un organismo pueda someterse, directa o indirectamente, pueden afectar el tiempo de vida útil de una enzima, así como la optimalidad de su velocidad de reacción (Hanson y cols., 2021). Esto mismo es retomado por Zhang y cols. (2016) en el momento en el que citan el trabajo realizado sobre Bio-PEPA (Gatherer y Galpin, 2013): los autores se preocupan sobre las implicaciones de haber ejecutado la simulación de manera determinista utilizando un algoritmo de regulación que tiene sus propias implicaciones causales internas.

En general, Aloisius Louie (2005) demuestra que, en términos *roseneanos*, aproximaciones como la de Palmer y cols. mediante Máquinas-X (2016) son útiles, pero no pasan de ser simulaciones. El mismo pensamiento de Louie podría extenderse a esta implementación del diagrama UML de Zhang y cols. (2016). En sí, la biología relacional niega a la computabilidad como una función de los seres vivos, pero no niega su simulabilidad, aunque ésta última se establece como una aproximación y no como una realización. Por otro lado, la relación de modelado que establece Rosen entre un sistema natural y un sistema formal podría utilizarse para criticar la decisión de Zhang y cols. (2016) de agregar atributos a las biomoléculas con el fin de que sus *simulacra* sean similares en conducta a las enzimas: Rosen indica que esto es una aproximación y no una construcción (Rosen, 1991).

Crucialmente, Palmer y cols. (2016) coinciden con Rosen en un aspecto fundamental de los $Sistemas - (M, R)$ que se desprende del talante anti-reduccionista de Rosen: *input* y estado no son separables (en el contexto de las Máquinas-X). Más generalmente, Rosen (1991, 2000) indica que los $Sistemas, -(M, R)$ son un modelo en el que hardware y software son la misma cosa: el programa está implícito en la organización

de las causas materiales mediante las causas eficientes. En este sentido, un organismo que sea la realización de un $Sistema - (M, R)$ reparará y reemplazará partes de hardware y software a la vez, so la clausura bajo la causa eficiente, con el único fin de mantener su invarianza organizacional, poder continuar anticipándose a su ambiente y buscar evitar así su condición de paro: la muerte causada por una degeneración organizacional. Así pues Rosen indica (1991) que su modelo no contiene, realmente, estados. Por tanto, el producir un modelo en UML de los $Sistemas-(M, R)$ introduce la necesidad de manejar estados, tal cual se establece en la implementación, contraviniendo la naturaleza de los $Sistemas-(M, R)$. Es posible decir que UML en realidad modela el modelo (sic) de Rosen, pero no logra producir una instancia del mismo una vez se programa. En Breiner y cols. (2021) se realiza una crítica a los alcances de modelado de UML y se argumenta que la Teoría de las Categorías podría ser una herramienta poderosa para un modelado menos ambiguo de sistemas de información. Robert Rosen utilizó la Teoría de las Categorías para producir el formalismo de los $Sistemas - (M, R)$, mientras que su alumno, Aloisius Louie, extiende el formalismo de Rosen también sobre la Teoría de las Categorías. En esta línea de pensamiento, es posible sugerir que UML no es, tal vez, el lenguaje de modelado adecuado para los $Sistemas - (M, R)$ los cuales nacieron en la Teoría de las Categorías. Explorar más a profundidad las limitantes de UML en el argumento pro-computacional de los $Sistemas-(M, R)$ es una línea de investigación potencial para el futuro en el área de las Ciencias de la Computación sugerida por los resultados de este trabajo recepcional.

7.2. Conclusiones

7.2.1. Las corrupciones de los Sistemas-(M,R) en el fondo de un programa

Como se ha indicado a lo largo de este trabajo, el caso de los $Sistemas - (M, R)$ es de particular interés para la Ciencia de la Computación debido a las implicaciones que tiene sobre áreas de desarrollo que son abordadas con fervor hoy-día tanto por la academia como por la industria. Zhang y cols. (2016) mencionan que, de ser cierto que los $Sistemas - (M, R)$ son el modelo para el organismo mínimo viable, entonces debido a la supuesta incomputabilidad de los $Sistemas - (M, R)$ la Vida Artificial sería también incomputable, proponiendo un obstáculo a los progresos actuales en dicha materia. Con el afán de encontrar una prueba empírica para falsear la tesis de

Rosen, los autores que se revisaron en este texto se encuentran con problemas que tienen que ver con la forma de interpretación del formalismo de Rosen a las cuales llaman **corrupciones**. Zhang y cols. (2016) deciden no programar los diagramas UML que produjeron debido a que, opinan ellos, tal programa podría inducir corrupciones al modelo. Estas corrupciones, se conjetura en estas conclusiones, hacen fallar a las implementaciones debido a que éstas últimas no pueden capturar un aspecto fundamental de los $Sistemas-(M,R)$: la clausura bajo causas eficientes. Esto puede abordarse de la siguiente manera: como se dijo antes, un $Sistema-(M,R)$ no es un cuerpo (*hardware*) con una programatura (*software*) asociada, sino que es en sí la realización material de la especificación (i.e. es *hardware* y *software* a la vez). Es por ello que una célula es capaz de cambiar sus propias partes (su propia especificación) en la búsqueda de evitar su condición de paro, mientras que las Máquinas de Turing (y sus derivados) no [2]. En el caso de las implementaciones de los $Sistemas-(M,R)$, el programador estaría tentado a suponer que el programa captura la clausura eficiente de manera completa, pero sería posible conjeturar que tal cosa no ocurre debido a que la clausura eficiente está definida para organismos (que son *hardware* y *software* a la vez). Cuando se piensa en un programa, éste no está clausurado bajo sus causas eficientes, aunque el programador argumente que sí, puesto que siempre son necesarias herramientas para su funcionamiento (morfismos externos), y si se implementare a un $Sistema-(M,R)$ en un lenguaje de programación (como se hizo en este trabajo), se necesitarán las bibliotecas auxiliares requeridas para trabajar en él. El lenguaje de programación mismo requerirá su compilador o intérprete, mientras que estos requerirán servicios del sistema operativo. Éste último dependerá de las instrucciones ensambladoras de bajo nivel, y éstas dependerán de compuertas lógicas en la unidad de procesamiento para dejar pasar o bloquear electrones. Cuando se programa en una computadora clásica una clausura eficiente, puede decirse que no se le está construyendo en verdad, sino que sólo se le está simulando en los términos de Rosen porque no hay forma de que el $Sistema-(M,R)$ programado pueda cambiar o reemplazar las partes dañadas en el basamento físico de la simulación misma (e.g. no podrá sustituir una compuerta lógica que haya dejado de funcionar), por lo que ni podrá ver su propia especificación ni estará clausurado bajo causas eficientes, sólo bajo una **simulación** de causas eficientes. Así, como es el caso de los sistemas autopoiéticos (Kampis, 1991), un $Sistema-(M,R)$ no podría ser describible utilizando espacios de estados finitos ni máquinas de estados finitos (lo cual puede ser

[2] Parecería interesante destacar también la similitud entre una *célula sin programa* y las *mentes básicas sin contenido* estudiadas por Hutto & Myin (2013)

intuitivamente derivado de la contención de los sistemas autopoiéticos dentro de los $Sistemas-(M,R)$ demostrada por Letelier y cols. (2003)). Es interesante indicar que el hecho de que un $Sistema-(M,R)$ implementado no tenga una verdadera clausura bajo causas eficientes no implica que el sistema no sea anticipatorio, pero para Rosen la anticipación no es causa suficiente para la vida, mientras que la clausura bajo causas eficientes sí (A. H. Louie, 2019).

7.2.2. La hipótesis del continuo Vida-Mente

Michael Kirchhoff y Tom Froese (2017) hacen un repaso sobre la conexión entre la vida y la mente desde el entendimiento del Principio de Energía Libre (FEP, por sus siglas en inglés), un formalismo que indica que los sistemas vivos (y los no-vivos) buscan mantenerse en sus estados esperados frente a estados inesperados, o que buscan estar listos para un estado de no-equilibrio (Kirchhoff y Froese, 2017). La forma cognitivista del FEP requeriría la presencia de contenido semántico en los organismos (integración de la información) lo cual es una demanda alta de mecanismos de procesamiento para formas de vida simple, por lo que la forma cognitivista del FEP establecería una distinción dura entre organismos con mentes (semánticas) y organismos sin mentes. Por otro lado, una versión más laxa del FEP propondría que la mente es un aspecto prácticamente ubicuo en el universo. Kirchhoff y Froese deciden acotar a la versión no-cognitivista mediante el uso de conceptos de normatividad e individualidad pertenecientes a la perspectiva enactivista de la Ciencia Cognitiva. Relevantemente y por un lado, los autores indican que la Teoría de la Autopoiesis estaría en el basamento del gradiente continuo entre vida y mente (Bitbol & Luisi (2004) estudiaron las condiciones de necesidad y suficiencia de la autopoiesis para la cognición, y concluyeron que, mientras la autopoiesis es condición necesaria para la cognición, no es condición suficiente). Por otro lado, la propuesta de Rosen forma parte de una familia de varios marcos de trabajo desarrollados para explicar formalmente a la vida (familia que incluye a las teorías de los Conjuntos Autocatalíticos y los Hiperciclos, por ejemplo) pero que comparte especialmente la noción de la causalidad circular en el metabolismo celular con la Teoría Autopoiética, razón por la cual la autopoiesis ha sido explorada mediante el uso de la teoría de los $Sistemas-(M,R)$ como fundamento matemático debido a la mayor generalidad del formalismo de Rosen (Letelier y cols., 2003). Por esto, puede ser adecuado realizar investigación en torno al uso de los $Sistemas-(M,R)$ en el estudio de la Ciencia Cognitiva que supone a la autopoiesis en su marco de trabajo. Interesantemente, Kirchhoff y Froese (2017) hacen referencia

a la definición de una *cognición* que produjo Maturana (1978) y que Maturana & Varela (1980) vuelven a describir como las conductas de un organismo relevantes para su auto-mantenimiento (en el entrecruce de interacciones entre el organismo y su entorno, considerando condiciones de cambio). Esta definición sobre una *cognición* (en contraste a una *computación*) es apropiada para el estudio de los $Sistemas-(M,R)$ por Everardo Robredo (comunicación personal, 2015) debido a la similitud entre la definición de cognición de Maturana y las dinámicas de auto-reparación celular descritas por $Sistemas-(M,R)$. Como se mencionó antes, para Rosen un $Sistema-(M,R)$ es una célula que está intentando, todo el tiempo, evitar su condición de paro mediante la reposición o reemplazo de partes o componentes que se van desgastando, se pierden o se agotan. Este concepto parece relacionable con aquel que subyace al FEP (i.e. un sistema busca estar preparado para estados inesperados minimizando su energía libre) así como a la definición de la cognición de Maturana. Es en este punto en el que es posible abordar el estudio de los $Sistemas-(M,R)$ dentro de la Filosofía de la Ciencia Cognitiva como un interés para la discusión sobre (a) el continuo vida-mente, así como sobre (b) las condiciones de necesidad y suficiencia de la minimalidad organísmica para la mentalidad, y finalmente (c) realizar un estudio sobre los alcances de mecanismos computantes para el estudio de la mente, considerando el contexto en el que se sitúa este trabajo. Es posible, inclusive, esgrimir la hipótesis de que tanto los aspectos anticipatorios como de clausura metabólica de los $Sistemas-(M,R)$ son *heredables* a lo largo del gradiente del continuo vida-mente.

7.3. El estado del arte y trabajo futuro

Robert Rosen consideró el desarrollo de su formalismo dentro de un entendimiento muy propio sobre la complejidad, una disciplina que hoy permanece sin una definición única. Para Rosen, los sistemas simples son aquellos cuyos modelos son todos computables, mientras que un sistema complejo será un sistema para el cual exista, al menos, un modelo incomputable (Rosen, 2006). El estudio de los Sistemas Complejos en la Biología ha resurgido recientemente y en esta disciplina se ha desarrollado un mayor entendimiento de el formalismo de Rosen (Baianu, 2006). Diversos intentos por explorar al modelo se han ejecutado, como la producción de Cornish-Bowden & Luz-Cárdenas (2007) que trabaja sobre una red simple de reacciones. Otros investigadores han probado la aplicabilidad de los $Sistemas-(M,R)$ a escalas mayores, explorando la complejidad de la Tierra mediante el formalismo de Rosen (Rubin y Crucifix, 2021), y recientemente Kineman (2012, 2019) ha extendido el concepto de

causalidad explotado por Rosen para considerar las cuatro causas de manera total en un marco llamado *Teoría-R*, lo que ha permitido proponer que tal teoría extendida predice satisfactoriamente la emergencia de los eukarya, los archaea, y de las bacterias, así como también ha permitido describir las condiciones necesarias y suficientes para decidir si un sistema natural es sustentable o no. Rosen fue un prolífico escritor y, en sí mismo, un divulgador, por lo que estos progresos en el estudio holístico de las implicaciones y alcances de los $Sistemas - (M, R)$ han ocurrido, en parte, gracias a que Rosen supo hacer accesibles los conceptos con los que construyó su formalismo a pesar de que éste último fuese tan abstracto y poco alcanzable por la mayor parte de los biólogos de su tiempo. Es posible que el rechazo a la incomputabilidad de los $Sistemas - (M, R)$ provenga también de la falta de comprensión total del formalismo por parte de quienes se dedican exclusivamente a lo computable, aunque eso no es, en suma, culpa únicamente de este último actor: Rosen, como Marx, dejó un trabajo inconcluso, y sus alumnos no han logrado apuntalarlo con la fuerza que la comunidad científica requiere en la era de la tecnociencia.

A pesar de los esfuerzos de Louie, no existe hasta hoy una forma canónica matemática para $(\mathcal{M}, \mathcal{R})$, y no obstante es posible distinguir coincidencias relevantes entre el trabajo de Rosen y otros modelos de metabolismo circular. A la par de la búsqueda de la forma canónica de el formalismo de Rosen, es conveniente plantearse preguntas sobre si la relación entre los Conjuntos Autocatalíticos, Sistemas Autopoiéticos y $Sistemas - (M, R)$ reside en alguna jerarquía, como la contención que Letelier y cols. (2003) demostraron y que se mencionó anteriormente, y si esta relación permitiría el uso de los $Sistemas - (M, R)$ para un estudio más general de espacios en los que otros modelos de metabolismo circular se han visto criticados por sufrir de limitantes relevantes (que bien podrían pensarse en los términos de su nivel de generalidad), como es el caso de la autopoiesis en la Ciencia Cognitiva (Froese y Stewart, 2010). Recientemente, Everardo Robredo (comunicación personal, 18 de octubre, 2021) ha encontrado buenos resultados utilizando el Teorema de Representación de Riesz–Fréchet en espacios de Hilbert, así como acoplando el concepto del tunelaje cuántico — que más y más químicos comienzan a admitir como necesario para entender la reactividad química (Schreiner, 2020), algo que es de máximo momento para el estudio del metabolismo celular — en el desarrollo de trabajo matemático y teórico enfocado en forjar un *framework* matemático adecuado para los $Sistemas - (M, R)$: la forma canónica. Es debido mencionar que, aunque el trabajo matemático que requiere la formalización de los $Sistemas - (M, R)$ es grande, lo es también el trabajo de conferirle un significado comunicable a otras disciplinas y para la misma interdisciplina.

Transportar al estudio del modelo de Rosen a la discusión filosófica no sólamente proveerá de mayor significado a cualesquiera basamentos matemáticos que puedan ser descubiertos conforme el tiempo avance, sino que también servirá como puente para que más disciplinas puedan preguntarse aspectos sobre sí mismas a partir de $(\mathcal{M}, \mathcal{R})$, definiendo a sus propios objetos de estudio en los términos causales que Rosen aprovechó. Además, los estudios transdisciplinarios pueden verse beneficiados del mayor tratamiento filosófico de $(\mathcal{M}, \mathcal{R})$, pudiendo éste ser incluido en discusiones como las que se dan ahora con Santo Tomás o San Agustín en búsqueda de actualizaciones.

En su definición de complejidad, Rosen concluye que el hecho de mirar a la Física como el caso general de estudio es una herencia de la metáfora de la máquina, y que cuando se parte de la Física, infinitas reducciones son necesarias para entender objetos en el mundo que, al final, nunca se entienden (Rosen, 1991). Así, y como ha quedado ya expuesto, para Rosen lo relevante en el estudio de los organismos no era el aspecto material, sino la organización. Es posible, a la par de la investigación matemática, computacional y molecular de los $Sistemas - (M, R)$, perseguir el análisis sobre la perspectiva *roseneana* de la vida en su forma general y en particular desde su propuesta de entender a la Biología como el caso general, para continuar y alentar la discusión sobre las ideas de Rosen quien lamentablemente murió a temprana edad, dejando una serie de notas y bocetos que su hija todavía resguarda y que han de ser publicados en el futuro. Continuar el trabajo de Rosen será dar aire nuevamente a un programa de investigación principalmente filosófico: aquel que busca definir a la vida y, potencialmente, sus orígenes, empresa en la cual Rosen depositó toda su vida.

«La belleza de una cosa viva no está en los átomos que le componen, sino en la forma en la que esos átomos están organizados».

Carl Edward Sagan

A El Sistema-(M,R) en Python

A.1. Clase Biomolecula

```
class Biomolecula(object):
    """"""
```

A.2. Clase Substrato

```
import Biomolecula

class Substrato(Biomolecula):

    def __init__(self):
        pass

    def producirOtrasBiomoleculas(self):
        pass
```

A.3. Clase Enzima

```
from Biomolecula import *

class Enzima(Biomolecula):
    count = 0

    def __init__(self, reusos):
        self.limiteDeVida = reusos
        self.degradado = False

```

```
10      def catalizarSubstrato(self, Comida):
11          pass
12
13      def getStatusDegradacion(self):
14          return self.degradado
15
16      def __autodestruccion(self):
17          print("Autodestruccion")
18          self.degradado = True
19
20      def registrarUso(self):
21          self.__incrementarUsos()
22
23      def __incrementarUsos(self):
24          print("Desgastando...")
25          self.limiteDeVida = self.limiteDeVida-1
26          if self.limiteDeVida <= 0:
27              self.__autodestruccion()
```

A.4. Clase A

```
1  from Substrato import *
2
3  class A(Substrato):
4
5      def __init__(self): #Constructor del objeto
6          super().__init__()
7          self.nombre = ("A")
8
9      def producirOtrasBiomoleculas(self, option, usos, count_cycles):
10         """
11         Funcion que elegira que biomolecula se produce (o en que se
       transforma el Substrato) a partir de una opcion
12         :param option:
13         :param usos:
14         :return:
15         """
16         super().producirOtrasBiomoleculas()
17         if option == "B":
18             return self.producirB()
19         elif option == "Beta":
```

```
            return self.producirBeta(usos, count_cycles)

    def producirB(self):
        """
        Funcion que simplemente regresa un objeto substrato B
        :return:
        """
        from B import B
        producto = B()
        return producto

    def producirBeta(self, usos, count_cycles):
        """
        Funcion que devuelve un objeto enzimatico Beta con usos dados por el parametro usos
        :param usos:
        :return:
        """
        from Beta import Beta
        producto = Beta(usos, count_cycles)
        return producto
```

A.5. Clase B

```
from Substrato import *

class B(Substrato):

    def __init__(self): #Constructor del objeto
        super().__init__()
        self.nombre = ("B")

    def producirOtrasBiomoleculas(self, option, usos, count_cycles):
        """
        Funcion que elegira que biomolecula se produce (o en que se transforma el Substrato) a partir de una opcion
        :param option:
        :param usos:
        :return:
        """
        super().producirOtrasBiomoleculas() #Delegado de
```

```
    responsabilidad a la clase parental
        if option == "F":
            return self.producirF()
        elif option == "FPrima":
            return self.producirFPrima(usos, count_cycles)

    def producirF(self):
        """
        Funcion que simplemente regresa un objeto substrato F
        :return:
        """
        from F import F
        producto = F()
        return producto

    def producirFPrima(self, usos, count_cycles):
        """
        Funcion que devuelve un objeto enzimatico FPrima con usos
    dados por el parametro usos
        :param usos:
        :return:
        """
        from FPrima import FPrima
        producto = FPrima(usos, count_cycles)
        return producto
```

A.6. Clase F

```
from Substrato import *

class F(Substrato):

    def __init__(self): #Constructor del objeto
        super().__init__()
        self.nombre = ("F")

    def producirOtrasBiomoleculas(self, usos, count_cycles):
        """
        Funcion que solicitara al substrato producir/transformarse
    en un unico tipo de biomolecula
        :param usos:
```

```
        :return:
        """
        super().producirOtrasBiomoleculas()
        return self.producirPhi(usos, count_cycles)

    def producirPhi(self,usos, count_cycles):
        """
        Funcion que devuelve un objeto enzimatico Phi con usos dados por el parametro usos
        :param usos:
        :return:
        """
        from Phi import Phi
        producto = Phi(usos, count_cycles)
        return producto
```

A.7. Clase Beta

```
from Enzima import *
import Substrato
import random as rand

class Beta(Enzima):

    def __init__(self, reusos, count_cycles): #Constructor del objeto
        super().__init__(reusos)
        self.nombre = ("Beta " + str(count_cycles))

    def catalizarSubstrato(self, ambienteEnzimaticoOrigen: [Enzima], ambienteEnzimaticoDestino: [Enzima], ambienteSubstrato: [Substrato], count_cycles):
        """
        Funcion del catalizador Beta. Devuelve el ambiente celular completo que se ingreso.
        :param ambienteEnzimaticoOrigen:
        :param ambienteEnzimaticoDestino:
        :param ambienteSubstrato:
        :return:
        """
        super().catalizarSubstrato("B") #Delegado de responsabilidad
```

```
    a la clase parental
20          elector = rand.choices((True, False), [0.5, 0.5]) #Elegimos
    aleatoriamente True, False o None.
21          ComidaF = ambienteSubstrato.pop() #Obtenemos un substrato-
    comida F para trabajar, el cual no volvera al ambiente
22          eleccionAleatoria = elector.pop() #Obtenemos el valor de
    nuestra eleccion aleatoria
23          #if eleccionAleatoria == False: #Si la eleccion aleatoria es
     False simulamos que la enzima no pudo hacer su trabajo
24           #   print("No me pude acoplar...")
25            #  return (ambienteEnzimaticoOrigen,
    ambienteEnzimaticoDestino, ambienteSubstrato)  #Regresamos lo que
     fue ingresado, tal cual.
26          if True: #Caso para garantizar que siempre se realice
    catalisis para FPrima
27          #elif eleccionAleatoria == True: #Si la eleccion es True,
    catalizamos para Phi
28              print("Beta: F -> Phi")
29              producto = self.catalizarReplicacion(ComidaF,
    count_cycles)
30              ambienteEnzimaticoDestino.append(producto) #Agregamos el
     resultado de la catalisis al ambiente destino
31          if self.getStatusDegradacion() == True: #Si la enzima no
    esta biodisponible
32              print("Esta Beta se ha degradado y no sera devuelta al
    ambiente.")
33          elif self.getStatusDegradacion() == False:
34              print("Devolviendo")
35              ambienteEnzimaticoOrigen.append(self) #Si llegmos hasta
    aca, regresamos la enzima Beta al ambiente
36          return (ambienteEnzimaticoOrigen,ambienteEnzimaticoDestino,
    ambienteSubstrato) #Devolvemos los ambientes alterados
37
38      def catalizarReplicacion(self, f, count_cycles):
39          """
40          Funcion de catalisis especifica para el producto Phi y con
    substrato F
41          :param f:
42          :return:
43          """
44          #usos = rand.randint(1, 5) #Construimos los usos que tendra
    la enzima de manera aleatoria
45          usos = 1 #Construimos los usos segun Zhang et al. (2016)
```

```
46          produccion = f.producirOtrasBiomoleculas(usos,count_cycles)
    #Asignamos a produccion lo que produzca el substrato B al ser
    catalizado con la opcion FPrima y un parametro de usos.
47          print("Registrando uso")
48          self.registrarUso() #Aumentamos el conteo de usos de la
    enzima Phi por uno.
49          return produccion #Regresamos el producto
```

A.8. Clase FPrima

```
1  from Enzima import *
2  import Substrato
3  import random as rand
4
5  class FPrima(Enzima):
6
7      def __init__(self, reusos, count_cycles): #Constructor del
    objeto
8          super().__init__(reusos)
9          self.nombre = ("FPrima " + str(count_cycles))
10
11     def catalizarSubstrato(self, ambienteEnzimaticoOrigen: [Enzima],
        ambienteEnzimaticoDestino: [Enzima], ambienteSubstrato: [
    Substrato], ambienteSubstratoNuevo: [Substrato], count_cycles):
12         """
13         Funcion del catalizador FPrima. Devuelve el ambiente celular
        completo que se ingreso.
14         :param ambienteEnzimaticoOrigen:
15         :param ambienteEnzimaticoDestino:
16         :param ambienteSubstrato:
17         :param ambienteSubstratoNuevo:
18         :return:
19         """
20         super().catalizarSubstrato("A") #Delegado de responsabilidad
        a la clase parental
21         elector = rand.choices((True, False, None), [0.4, 0.4, 0.2])
        #Elegimos aleatoriamente True, False o None.
22         ComidaA = ambienteSubstrato.pop() #Obtenemos un substrato-
    comida B para trabajar, el cual no volvera al ambiente
23         eleccionAleatoria = elector.pop() #Obtenemos el valor de
    nuestra eleccion aleatoria
```

```
24          #if eleccionAleatoria == None: #Si la eleccion aleatoria es
    None simulamos que la enzima no pudo hacer su trabajo
25           #   print("No me pude acoplar...")
26            #  return (ambienteEnzimaticoOrigen,
    ambienteEnzimaticoDestino, ambienteSubstrato,
    ambienteSubstratoNuevo) #Regresamos lo que fue ingresado, tal
    cual.
27          if True: #Caso para garantizar que siempre se realice
    catalisis para B
28          #elif eleccionAleatoria == True: #Si la eleccion es True,
    catalizamos para B
29              print("FPrima: A -> B")
30              producto = self.catalizarMetabolismoB(ComidaA)
31              ambienteSubstratoNuevo.append(producto)  #Agregamos el
    resultado de la catalisis al ambiente destino
32          if True: #Caso para garantizar que siempre se realice
    catalisis para Beta
33          #elif eleccionAleatoria == False: #Si la eleccion es False,
    catalizamos para Beta
34              print("FPrima: A -> Beta")
35              producto = self.catalizarMetabolismoBeta(ComidaA,
    count_cycles)
36              ambienteEnzimaticoDestino.append(producto) #Agregamos el
     resultado de la catalisis al ambiente destino
37          if self.getStatusDegradacion() == True: #Si la enzima no
    esta biodisponible
38              print("Esta FPrima se ha degradado y no sera devuelta al
     ambiente.")
39          elif self.getStatusDegradacion() == False:
40              print("Devolviendo")
41              ambienteEnzimaticoOrigen.append(self) #Si llegmos hasta
    aca, regresamos la enzima FPrima al ambiente
42          return (ambienteEnzimaticoOrigen,ambienteEnzimaticoDestino,
    ambienteSubstrato, ambienteSubstratoNuevo) #Devolvemos los
    ambientes alterados
43
44      def catalizarMetabolismoB(self, a):
45          """
46          Funcion de catalisis especifica para el producto B y con
    substrato A
47          :param a:
48          :return:
49          """
```

```
50          produccion = a.producirOtrasBiomoleculas("B", 3, 0) #
    Asignamos a produccion lo que produzca el substrato A al ser
    catalizado con la opcion B y un parametro dummy.
51          self.registrarUso() #Aumentamos el conteo de usos de la
    enzima FPrima por uno
52          return produccion  #Regresamos el producto
53
54      def catalizarMetabolismoBeta(self, a, count_cycles):
55          """
56          Funcion de catalisis especifica para el producto Beta y con
    substrato A
57          :param a:
58          :return:
59          """
60          #usos = rand.randint(1, 5) #Construimos los usos que tendra
    la enzima de manera aleatoria
61          usos = 1 #Construimos los usos segun Zhang et al. (2016)
62          produccion = a.producirOtrasBiomoleculas("Beta", usos,
    count_cycles) #Asignamos a produccion lo que produzca el
    substrato A al ser catalizado con la opcion Beta y un parametro
    de usos.
63          print("Registrando uso")
64          self.registrarUso() #Aumentamos el conteo de usos de la
    enzima FPrima por uno
65          return produccion #Regresamos el producto
```

A.9. Clase Phi

```
1  from Enzima import *
2  import Substrato
3  import random as rand
4
5  class Phi(Enzima):
6
7      def __init__(self, reusos, count_cycles): #Constructor del
    objeto
8          super().__init__(reusos)
9          self.nombre = ("Phi " + str(count_cycles))
10
11     def catalizarSubstrato(self, ambienteEnzimaticoOrigen: [Enzima],
       ambienteEnzimaticoDestino: [Enzima], ambienteSubstrato: [
```

```
Substrato], ambienteSubstratoNuevo: [Substrato], count_cycles):
        """
        Funcion del catalizador Phi. Devuelve el ambiente celular completo que se ingreso.
        :param ambienteEnzimaticoOrigen:
        :param ambienteEnzimaticoDestino:
        :param ambienteSubstrato:
        :param ambienteSubstratoNuevo:
        :return:
        """
        super().catalizarSubstrato("B") #Delegado de responsabilidad a la clase parental
        elector = rand.choices((True, False, None), [0.4, 0.4, 0.2]) #Elegimos aleatoriamente True, False o None.
        ComidaB = ambienteSubstrato.pop() #Obtenemos un substrato-comida B para trabajar, el cual no volvera al ambiente
        eleccionAleatoria = elector.pop() #Obtenemos el valor de nuestra eleccion aleatoria
        #if eleccionAleatoria == None: #Si la eleccion aleatoria es None simulamos que la enzima no pudo hacer su trabajo
         #   print("No me pude acoplar...")
          #  return (ambienteEnzimaticoOrigen, ambienteEnzimaticoDestino, ambienteSubstrato, ambienteSubstratoNuevo)  #Regresamos lo que fue ingresado, tal cual.
        if True: #Caso para garantizar que siempre se realice catalisis para F
        #elif eleccionAleatoria == True: #Si la eleccion es True, catalizamos para F
            print("Phi: B -> F")
            producto = self.catalizarReparacionF(ComidaB)
            ambienteSubstratoNuevo.append(producto) #Agregamos el resultado de la catalisis al ambiente destino
        if True: #Caso para garantizar que siempre se realice catalisis para FPrima
        #elif eleccionAleatoria == False: #Si la eleccion es False, catalizamos para FPrima
            print("Phi: B -> FPrima")
            producto = self.catalizarReparacionFPrima(ComidaB, count_cycles)
            ambienteEnzimaticoDestino.append(producto) #Agregamos el resultado de la catalisis al ambiente destino
        if self.getStatusDegradacion() == True: #Si la enzima no
```

```
    esta biodisponible
38              print("Esta Phi se ha degradado y no sera devuelta al
    ambiente.")
39          elif self.getStatusDegradacion() == False:
40              print("Devolviendo")
41              ambienteEnzimaticoOrigen.append(self) #Si llegmos hasta
    aca, regresamos la enzima Phi al ambiente
42          return (ambienteEnzimaticoOrigen,ambienteEnzimaticoDestino,
    ambienteSubstrato, ambienteSubstratoNuevo) #Devolvemos los
    ambientes alterados
43
44      def catalizarReparacionF(self, b):
45          """
46          Funcion de catalisis especifica para el producto F y con
    substrato B
47          :param b:
48          :return:
49          """
50          produccion = b.producirOtrasBiomoleculas("F", 3, 0)  #
    Asignamos a produccion lo que produzca el substrato B al ser
    catalizado con la opcion F y un parametro dummy.
51          self.registrarUso() #Aumentamos el conteo de usos de la
    enzima Phi por uno.
52          return produccion  #Regresamos el producto
53
54      def catalizarReparacionFPrima(self, b, count_cycles):
55          """
56          Funcion de catalisis especifica para el producto FPrima y
    con substrato B
57          :param b:
58          :return:
59          """
60          #usos = rand.randint(1, 5) #Construimos los usos que tendra
    la enzima de manera aleatoria
61          usos = 1 #Construimos los usos segun Zhang et al. (2016)
62          produccion = b.producirOtrasBiomoleculas("FPrima", usos,
    count_cycles) #Asignamos a produccion lo que produzca el
    substrato B al ser catalizado con la opcion FPrima y un parametro
     de usos.
63          print("Registrando uso")
64          self.registrarUso() #Aumentamos el conteo de usos de la
    enzima Phi por uno.
65          return produccion #Regresamos el producto
```

A.10. MAIN para conectar clases y correr simulaciones

Es importante notar que éste MAIN corresponde a una de las muchas variaciones que éste pudo haber presentado para acoplarlo a las distintas condiciones experimentales que fueron de interés para este trabajo recepcional.

```
from A import A
from B import B
from F import F
from FPrima import FPrima
from Beta import Beta
from Phi import Phi

import numpy as np
import matplotlib.pyplot as plt
import random as rand
import os

"""
@Autor: Enrique Francisco Soto Astorga
@Versi[U+FFFD]n: 1.0
@Descripci[U+FFFD]n: Main para construir un ambiente celular y simular un
    Sistema-(M,R) sg[U+FFFD]n el diagrama UML de Zhang et al. (2016).
"""

def main():

    def count(start=0, step=1): #Funci[U+FFFD]n para aumentar un conteo,
    utilizada en la graficaci[U+FFFD]n
        n = start
        while True:
            yield n
            n += step

    plt.style.use('ggplot') #El estilo de gr[U+FFFD]fica
    x_vals = [] #Almac[U+FFFD]n de valores para eje X en la gr[U+FFFD]fica
    index = count() #Contador para aumentar valores en la gr[U+FFFD]fica

    def celula(): #Funci[U+FFFD]n que construye un ambiente celular y pone
    a trabajar al Sistema-(M,R)
        print("Construyendo ambiente celular...")
```

```
33        #Ambientes para cada tipo de biomol[U+FFFD]cula
34        Beta_environ = []
35        Phi_environ = []
36        FPrima_environ = []
37        A_environ = []
38        B_environ = []
39        F_environ = []
40
41        #Construcci[U+FFFD]n de cada tipo de ambiente enzim[U+FFFD]tico
42        for i in range(3):
43            #usos = rand.randint(1, 5) #Construimos los usos que
    tend[U+FFFD]la enzima de manera aleatoria
44            FPrima_environ.append(FPrima(3, 0))
45        for i in range(3):
46            #usos = rand.randint(1, 5) #Construimos los usos que
    tend[U+FFFD]la enzima de manera aleatoria
47            Beta_environ.append(Beta(3, 0))
48        for i in range(3):
49            #usos = rand.randint(1, 5) #Construimos los usos que
    tend[U+FFFD]la enzima de manera aleatoria
50            Phi_environ.append(Phi(3, 0))
51        #Construcci[U+FFFD]n de cada tipo de ambiente de substratos
52        for i in range(3):
53            F_environ.append(F())
54        for i in range(3):
55            B_environ.append(B())
56        for i in range(1):
57            A_environ.append(A())
58
59        count_cycles = 0 #Contador de ciclos para algunos
    experimentos
60        count_veces_realimento = 0
61        #Contadores de longitudes de cada ambiente, utilizados como
    output para graficar el comportamiento del Sistema-(M,R)
62        count_FPrima = []
63        count_Beta = []
64        count_Phi = []
65        count_A = []
66        count_B = []
67        count_F = []
68        #Contadores de vida[U+FFFD]til de cada enzima cuando no est[U+FFFD]n en
     uso
69        FPrima_counter = 0
```

```
70        Beta_counter = 0
71        Phi_counter = 0
72
73
74        while True: #Inicio de la simulaci[U+FFFD]n
75            if count_cycles > 3000:
76                break
77            import Enzima
78            print("Actual")
79            print("|A| = "+str(len(A_environ))+", |B| = "+str(len(
    B_environ))+", |F| ="+str(len(F_environ)))
80            print("|FPrima| = " + str(len(FPrima_environ)) + ", |
    Beta| = " + str(len(Beta_environ)) + ", |Phi| =" + str(len(
    Phi_environ)))
81
82            FPrima_printer = []
83            Phi_printer = []
84            Beta_printer = []
85            for i in range(len(FPrima_environ)):
86                FPrima_printer.append(FPrima_environ[i].nombre)
87            for i in range(len(Beta_environ)):
88                Beta_printer.append(Beta_environ[i].nombre)
89            for i in range(len(Phi_environ)):
90                Phi_printer.append(Phi_environ[i].nombre)
91            print("FPrima list:")
92            print(FPrima_printer)
93            print("Beta list:")
94            print(Beta_printer)
95            print("Phi list:")
96            print(Phi_printer)
97
98            if count_cycles<11000: #Si llevamos menos de n-ciclos,
    alimentar con A al Sistema-(M,R)
99                A_environ.append(A())
100           if len(FPrima_environ)<100 and count_veces_realimento<3:
101               count_veces_realimento = count_veces_realimento+1
102               while len(A_environ)<100:
103                   A_environ.append(A())
104           count_cycles = count_cycles+1 #Aumentar los cliclos por
    uno
105           count_FPrima.append(len(FPrima_environ)) #Contar los
    FPrima
106           count_Beta.append(len(Beta_environ)) #Contar los Beta
```

```
107             count_Phi.append(len(Phi_environ)) #Contar los Phi
108             count_A.append(len(A_environ)) #Contar los A
109             count_B.append(len(B_environ)) #Contar los B
110             count_F.append(len(F_environ)) #Contar los F
111             x_vals.append(next(index))
112             print("Ciclo:")
113             print(count_cycles)
114             if len(FPrima_environ)>0: #Si todav[U+FFFD] hay enzimas
    FPrima
115                 if len(A_environ)>0: #Y hay comida para FPrima:
116                     #FPrima: A -> B/Beta
117                     enzima1 = FPrima_environ.pop(0) #Sacamos la
    FPrima y la usamos
118                     FPrima_environ, Beta_environ, A_environ,
    B_environ = enzima1.catalizarSubstrato(FPrima_environ,
    Beta_environ, A_environ, B_environ, count_cycles)
119             if len(Phi_environ)>0: #Si todav[U+FFFD] hay enzimas Phi
120                 if len(B_environ)>0: #Y hay comida para Phi:
121                     #Phi: B -> F/FPrima
122                     enzima2 = Phi_environ.pop(0) #Sacamos la Phi y
    la usamos
123                     Phi_environ, FPrima_environ, B_environ,
    F_environ = enzima2.catalizarSubstrato(Phi_environ,
    FPrima_environ, B_environ, F_environ, count_cycles)
124             if len(Beta_environ)>0: #Si todav[U+FFFD] hay enzimas Beta
125                 if len(F_environ)>0 : #Y hay comida para Beta:
126                     #Beta: F -> Phi
127                     enzima3 = Beta_environ.pop(0) #Sacamos la Beta y
     la usamos
128                     Beta_environ, Phi_environ, F_environ = enzima3.
    catalizarSubstrato(Beta_environ, Phi_environ, F_environ,
    count_cycles)
129             #if F_environ ==  [] and B_environ == [] and A_environ
    == []: #Si ya no hay comida, detenemos la simulaci[U+FFFD]n
130              #    break
131             if len(FPrima_environ) == 0 and len(Beta_environ) == 0
    and len(Phi_environ) == 0: #Si ya no hay enzimas, detenemos la
    simulaci[U+FFFD]n.
132                 break
133             if len(FPrima_environ) > 0: #Si hay FPrimas
134                 if FPrima_counter < 3: #Y el contador de vida[U+FFFD]til
    no se ha rebasado
```

```
135                     FPrima_counter = FPrima_counter+1 #Aumentar el
    contador
136                 if FPrima_counter == 3: #Si el contador es [U+FFFD] por
    rebasarse
137                     FPrima_counter = 0 #Se reinicia
138                     FPrima_environ.pop() #Sacamos una FPrima
139             if len(Beta_environ) > 0: #Si hay Betas
140                 if Beta_counter < 3: #Y el contador de vida [U+FFFD] tiene
     se ha rebasado
141                     Beta_counter = Beta_counter+1 #Aumentar el
    contador
142                 if Beta_counter == 3: #Si el contador es [U+FFFD] por
    rebasarse
143                     Beta_counter = 0 #Se reinicia
144                     Beta_environ.pop() #Sacamos una Beta
145             if len(Phi_environ) > 0: #Si hay Phis
146                 if Phi_counter < 3: #Y el contador de vida [U+FFFD] tiene
    se ha rebasado
147                     Phi_counter = Phi_counter+1 #Aumentar el
    contador
148                 if Phi_counter == 3: #Si el contador es [U+FFFD] por
    rebasarse
149                     Phi_counter = 0 #Se reinicia
150                     Phi_environ.pop() #Y sacamos una Phi
151             input("CONT")
152             os.system('clear') #Limpiamos la pantalla por comodidad
153 
154         print("La vida ha cesado...")
155         return count_FPrima, count_Phi, count_Beta, count_A, count_B
    , count_F #Regresamos las listas de conteo de ambientes
156 
157     def graficar(): #Funci[U+FFFD] para graficar los resultados de la
    simulaci[U+FFFD]n usando MatPlotLib
158         fprimavals = np.array(l1)
159         phivals = np.array(l2)
160         betavals = np.array(l3)
161         avals = np.array(la)
162         bvals = np.array(lb)
163         fvals = np.array(lf)
164         x = np.array(x_vals)
165         np.array(fprimavals)
166         np.array(phivals)
167         np.array(betavals)
```

```
168         np.array(avals)
169         np.array(bvals)
170         np.array(fvals)
171         print(len(fprimavals))
172         print(len(betavals))
173         print(len(phivals))
174         print(len(x))
175         plt.xlabel("Ciclos")
176         plt.ylabel("Enzimas y Substratos")
177         plt.plot(x, l1, label='fPrima')
178         plt.plot(x, phivals, label='Phi')
179         plt.plot(x, betavals, label='Beta')
180         #plt.plot(x, avals, label='A')
181         plt.plot(x, bvals, label='B')
182         plt.plot(x, fvals, label='F')
183         plt.legend(loc='upper left')
184         plt.show()
185 
186     l1, l2, l3, la, lb, lf = celula() #Asignamos los resultados de
    una simulaci[U+FFFD]n de un Sistema-(M,R) a las listas que ser[U+FFFD]
    graficadas.
187     graficar()
188 
189 if __name__ == "__main__":
190     main()
```

B Algunos resultados de las simulaciones

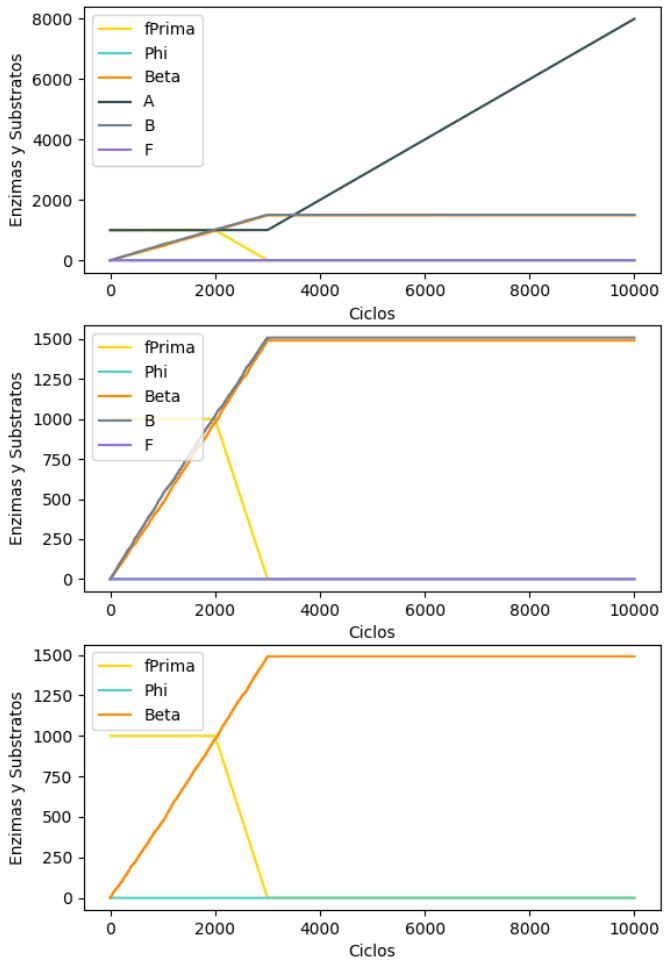


Figura B.1: Comportamiento del $Sistema - (M, R)$ en una simulación bajo los parámetros establecidos por Zhang y cols. (2016). Se muestran las gráficas que dibujan: el comportamiento de todas las biomoléculas, el comportamiento de todas las biomoléculas salvo A, y el comportamiento de las enzimas.

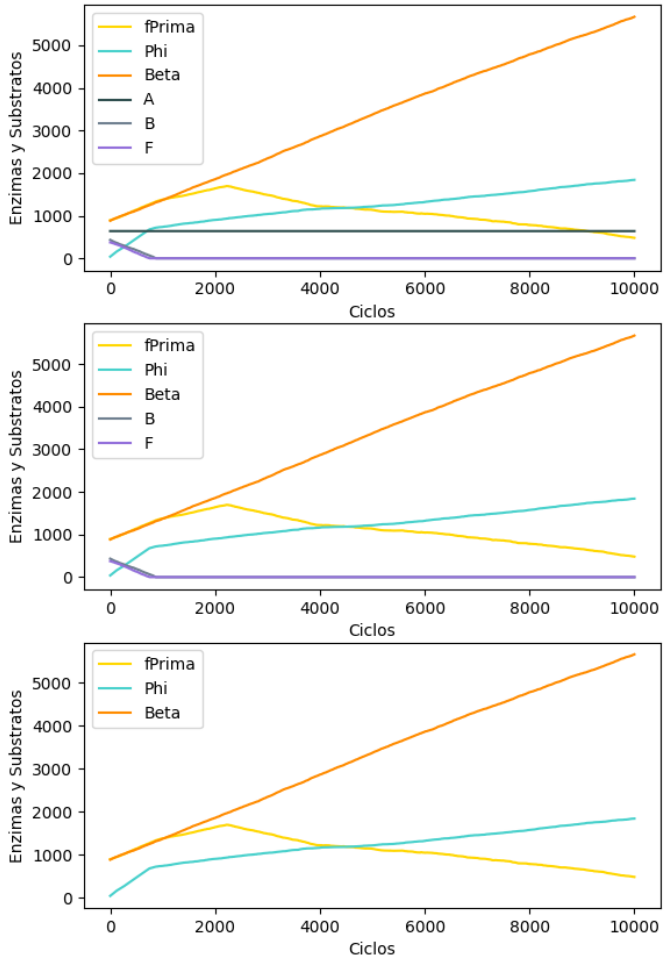


Figura B.2: Comportamiento del $Sistema - (M, R)$ en una simulación bajo los parámetros establecidos por Zhang y cols. (2016) agregando la estocasticidad en la elección de las condiciones iniciales. Se muestran las gráficas que dibujan: el comportamiento de todas las biomoléculas, el comportamiento de todas las biomoléculas salvo A, y el comportamiento de las enzimas.

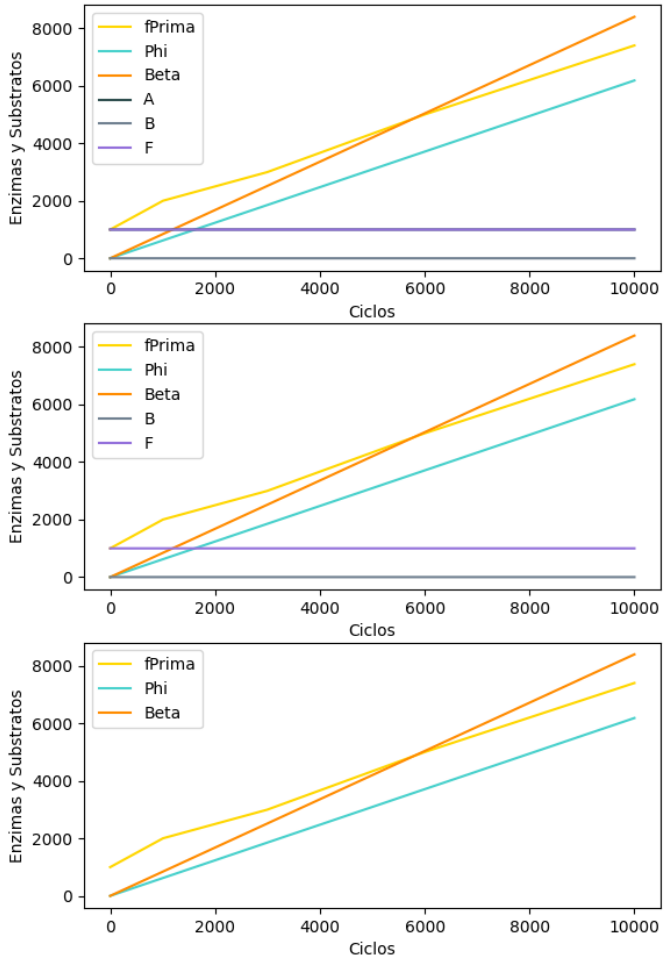


Figura B.3: Promedio del comportamiento del $Sistema - (M, R)$ en una simulación bajo los parámetros establecidos por Zhang y cols. (2016) modificando el requerimiento sobre la separación de la producción de las enzimas y substratos. Se muestra el comportamiento de todas las biomoléculas.

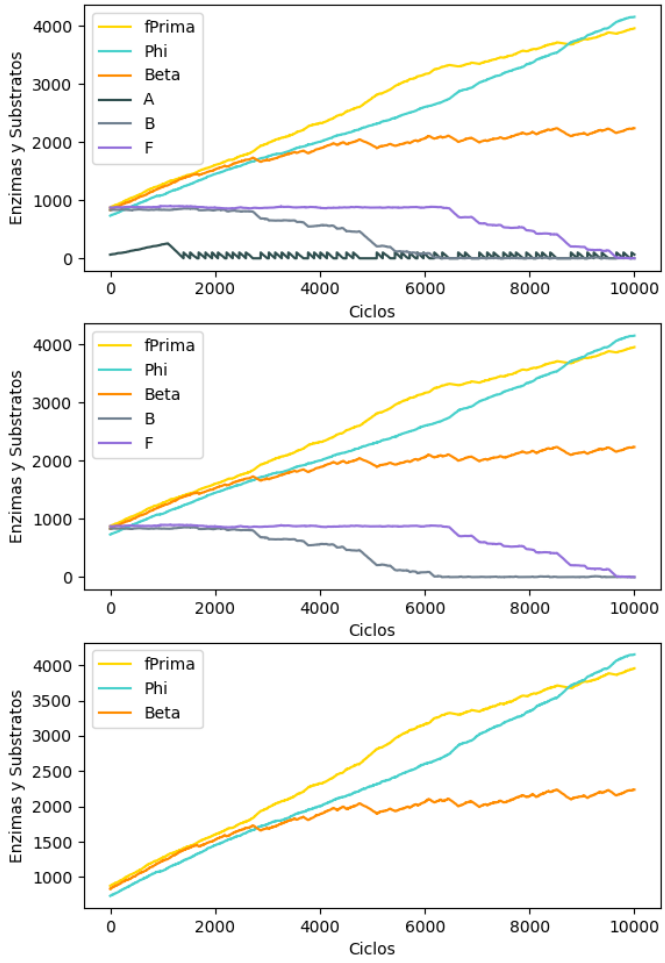


Figura B.4: Comportamiento del $Sistema - (M, R)$ en una simulación en la que se deja de alimentar aleatoriamente al sistema entre el ciclo 0 y el ciclo 2000. Se muestran las gráficas que dibujan: el comportamiento de todas las biomoléculas, el comportamiento de todas las biomoléculas salvo A, y el comportamiento de las enzimas.

Bibliografía

Referencias

Agustín de Hipona, S. (s.f.). *Del génesis contra los maniqueos. Traducido por Lope Cilleruelo.* Descargado de http://www.augustinus.it/spagnolo/genesi_dcm/index2.htm

Baianu, I. C. (2006, marzo). Robert rosen's work and complex systems biology. *Axiomathes*, *16*(1-2), 25–34. Descargado 2022-06-02, de http://link.springer.com/10.1007/s10516-005-4204-z doi: 10.1007/s10516-005-4204-z

Bedau, M., y Cleland, C. (Eds.). (2010). *The nature of life: classical and contemporary perspectives from philosophy and science* (1st ed.). Cambridge ; New York: Cambridge University Press.

Bitbol, M., y Luisi, P. L. (2004, noviembre). Autopoiesis with or without cognition: defining life at its edge. *Journal of The Royal Society Interface*, *1*(1), 99–107. Descargado 2022-06-02, de https://royalsocietypublishing.org/doi/10.1098/rsif.2004.0012 doi: 10.1098/rsif.2004.0012

Breiner, S., Padi, S., Subrahmanian, E., y Sriram, R. D. (2021, mayo). *Deconstructing uml, part 1: modeling classes with categories* (Inf. Téc.). National Institute of Standards and Technology. Descargado 2022-06-02, de https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8358.pdf doi: 10.6028/NIST.IR.8358

Cavendish, M., y Lilley, K. (1994). *The blazing world and other writings* (Reprint ed.). London ; New York: Penguin Classics.

Ciocchetta, F., y Hillston, J. (2010, mayo). *Bio-PEPA.* Descargado de https://homepages.inf.ed.ac.uk/jeh/Bio-PEPA/biopepa.html

Cornish-Bowden, A., y Cárdenas, M. (2007, octubre). Organizational invariance in (M,r)-systems. *Chemistry & Biodiversity*, *4*(10), 2396–2406. Descargado 2022-06-02, de https://onlinelibrary.wiley.com/doi/10.1002/cbdv.200790195 doi: 10.1002/cbdv.200790195

Dupont, C., Armant, D., y Brenner, C. (2009, septiembre). Epigenetics: definition, mechanisms and clinical perspective. *Seminars in Reproductive Medicine*, *27*(05), 351–357. Descargado 2022-06-02, de http://www.thieme-connect.de/DOI/DOI?10.1055/s-0029-1237423 doi: 10.1055/s-0029-1237423

Froese, T., y Stewart, J. (2010). Life After Ashby: Ultrastability and the Autopoietic Foundations of Biological Autonomy. *Cybernetics and Human Knowing*, *17*, 7–49.

Gandy, R. (1988). *The confluence of ideas in 1936. In R. Gandy, A half-century survey on the universal turing machine* (pp. 55–111 ed.). Oxford University Press.

Garey, M. R., y Johnson, D. S. (2009). *Computers and intractability: a guide to the theory of NP-completeness* (27. print ed.). New York [u.a]: Freeman.

Gatherer, D., y Galpin, V. (2013). Rosen's (M,R) system in process algebra. *BMC Systems Biology*, *7*(1), 128. Descargado 2022-06-02, de http://bmcsystbiol.biomedcentral.com/articles/10.1186/1752-0509-7-128 doi: 10.1186/1752-0509-7-128

Goertzel, B. (2002). *Creating internet intelligence: wild computing, distributed digital consciousness, and the emerging global brain.* Springer New York, NY. (OCLC: 1159113636)

Gold, V. (Ed.). (2019). *The iupac compendium of chemical terminology: the gold book* (4.ª ed.). Research Triangle Park, NC: International Union of Pure and Applied Chemistry (IUPAC). Descargado 2022-06-02, de https://goldbook.iupac.org/ doi: 10.1351/goldbook

Goudsmit, A. (2007, octubre). Some reflections on Rosen's conceptions of semantics and finality. *Chemistry & Biodiversity*, *4*(10), 2427–2435. Descargado 2022-06-02, de https://onlinelibrary.wiley.com/doi/10.1002/cbdv.200790198 doi: 10.1002/cbdv.200790198

Goy, I. (2015). The Antinomy of Teleological Judgment. *Studi Kantiani*, *XXVIIII*, 65–88.

Hall, M. B. (1994). *The scientific renaissance 1450-1630.* New York: Dover.

Hammes, G. G. (2008, agosto). How do enzymes really work? *Journal of Biological Chemistry*, *283*(33), 22337–22346. Descargado 2022-06-02, de https://linkinghub.elsevier.com/retrieve/pii/S0021925820747654 doi: 10.1074/jbc.X800005200

Hanson, A. D., McCarty, D. R., Henry, C. S., Xian, X., Joshi, J., Patterson, J. A., ... Millar, A. H. (2021, marzo). The number of catalytic cycles in

an enzyme's lifetime and why it matters to metabolic engineering. *Proceedings of the National Academy of Sciences*, *118*(13), e2023348118. Descargado 2022-06-02, de https://pnas.org/doi/full/10.1073/pnas.2023348118 doi: 10.1073/pnas.2023348118

Hutto, D. D., y Myin, E. (2013). *Radicalizing enactivism: basic minds without content.* Cambridge, Mass: MIT Press.

Jacobson, I. (1992). *Object-oriented software engineering: a use case driven approach* (Revised Fourth Printing ed. ed.). Addison-Wesley.

Kampis, G. (1991). *Self-modifying systems in biology and cognitive science: a new framework for dynamics, information, and complexity* (1st ed ed.) (n.º v. 6). Oxford ; New York: Pergamon Press.

Kineman, J. J. (2012, septiembre). R-theory: a synthesis of robert rosen's relational complexity: a synthesis of robert rosen's relational complexity. *Systems Research and Behavioral Science*, *29*(5), 527–538. Descargado 2022-06-04, de https://onlinelibrary.wiley.com/doi/10.1002/sres.2156 doi: 10.1002/sres.2156

Kineman, J. J. (2019). Four kinds of anticipatory (M-r) life and a definition of sustainability. En R. Poli (Ed.), *Handbook of Anticipation* (pp. 1147–1193). Cham: Springer International Publishing. Descargado 2022-06-02, de http://link.springer.com/10.1007/978-3-319-91554-8_53 doi: 10.1007/978-3-319-91554-8_53

Kirchhoff, M. D., y Froese, T. (2017, abril). Where there is life there is mind: in support of a strong life-mind continuity thesis. *Entropy*, *19*(4), 169. Descargado 2022-06-02, de http://www.mdpi.com/1099-4300/19/4/169 doi: 10.3390/e19040169

Kuhlman, D. (2012). *A python book: beginning python, advanced python, and python exercises.* Archivado el 23 de junio de 2013. Descargado de https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html

Letelier, J. C., Marı´n, G., y Mpodozis, J. (2003, mayo). Autopoietic and (M,R) systems. *Journal of Theoretical Biology*, *222*(2), 261–272. Descargado 2022-06-02, de https://linkinghub.elsevier.com/retrieve/pii/S0022519303000341 doi: 10.1016/S0022-5193(03)00034-1

Letelier, J.-C., Soto-Andrade, J., Guíñez Abarzúa, F., Cornish-Bowden, A., y Luz Cárdenas, M. (2006, febrero). Organizational invariance and metabolic closure: Analysis in terms of systems. *Journal of Theoretical Biology*, *238*(4), 949–961. Descargado 2022-06-02, de https://linkinghub.elsevier.com/

retrieve/pii/S0022519305003073 doi: 10.1016/j.jtbi.2005.07.007

Louie, A. H. (2005, diciembre). Any material realization of the (M,r)-systems must have noncomputable models. *Journal of Integrative Neuroscience*, *04*(04), 423–436. Descargado 2022-06-04, de http://www.worldscientific.com/doi/abs/10.1142/S0219635205000926 doi: 10.1142/S0219635205000926

Louie, A. H. (2011, septiembre). Essays on more than life itself. *Axiomathes*, *21*(3), 473–489. Descargado 2022-06-03, de http://link.springer.com/10.1007/s10516-011-9153-0 doi: 10.1007/s10516-011-9153-0

Louie, A. H. (2013). *The reflection of life: functional entailment and imminence in relational biology* (n.º v. 29). New York, NY: Springer.

Louie, A. H. (2019). Relational biology. En R. Poli (Ed.), *Handbook of Anticipation* (pp. 191–218). Cham: Springer International Publishing. Descargado 2022-06-02, de http://link.springer.com/10.1007/978-3-319-91554-8_17 doi: 10.1007/978-3-319-91554-8_17

Louie, A. H., y Kercel, S. W. (2007, julio). Topology and life redux: robert rosen's relational diagrams of living systems. *Axiomathes*, *17*(2), 109–136. Descargado 2022-06-02, de http://link.springer.com/10.1007/s10516-007-9014-z doi: 10.1007/s10516-007-9014-z

Louie, A. H.-Y. (2009). *More than life itself: a synthetic continuation in relational biology* (n.º Vol. 1). Frankfurt Heusenstamm Paris: Ontos-Verl. [u.a.].

Luz-Cárdenas, M., Letelier, J.-C., Gutierrez, C., Cornish-Bowden, A., y Soto-Andrade, J. (2010, marzo). Closure to efficient causation, computability and artificial life. *Journal of Theoretical Biology*, *263*(1), 79–92. Descargado 2022-06-02, de https://linkinghub.elsevier.com/retrieve/pii/S0022519309005360 doi: 10.1016/j.jtbi.2009.11.010

Madigan, M. T., Martinko, J. M., y Parker, J. (2002). *Brock biology of microorganisms* (10th ed ed.). Upper Saddle River, NJ: Prentice Hall/Pearson Education.

Magner, L. N. (2002). *A history of the Life Sciences* (3rd ed., rev. and expanded ed.). New York: M. Dekker.

Malaterre, C., y Chartier, J.-F. (2021, mayo). Beyond categorical definitions of life: a data-driven approach to assessing lifeness. *Synthese*, *198*(5), 4543–4572. Descargado 2022-06-02, de https://link.springer.com/10.1007/s11229-019-02356-w doi: 10.1007/s11229-019-02356-w

Maturana, H. R. (1978). Cognition. En *Hejl, Peter M., Wolfram K. Köck, and Gerhard Roth (Eds.) Wahrnehmung und Kommunikation* (pp. pp. 29–49). Frankfurt: Peter Lang. Descargado de http://www.enolagaia.com/

M78bCog.html

Maturana, H. R., y Varela, F. J. (1980). *Autopoiesis and cognition: the realization of the living* (Vol. 42). Dordrecht: Springer Netherlands. Descargado 2022-06-02, de http://link.springer.com/10.1007/978-94-009-8947-4 doi: 10.1007/978-94-009-8947-4

Mossio, M., Longo, G., y Stewart, J. (2009, abril). A computable expression of closure to efficient causation in terms λ-calculus. *Journal of Theoretical Biology*, *257*(3), 489–498. Descargado 2022-06-02, de https://linkinghub.elsevier.com/retrieve/pii/S0022519308006498 doi: 10.1016/j.jtbi.2008.12.012

OMG. (2017). *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5.1 Object Management Group.* (Technical report, Object Management Group). Descargado de https://www.omg.org/spec/UML/2.5.1/PDF

Palmer, M. L., Williams, R. A., y Gatherer, D. (2016, noviembre). Rosen's (M,r) system as an X-machine. *Journal of Theoretical Biology*, *408*, 97–104. Descargado 2022-06-02, de https://linkinghub.elsevier.com/retrieve/pii/S0022519316302417 doi: 10.1016/j.jtbi.2016.08.007

Phillips, D. (2015). *Python 3 Object-oriented Programming: Building robust and maintainable software with object oriented design patterns in Python* (2nd edition ed.). Packt Publishing.

Piedrafita, G., Montero, F., Morán, F., Cárdenas, M. L., y Cornish-Bowden, A. (2010, agosto). A simple self-maintaining metabolic system: robustness, autocatalysis, bistability. *PLoS Computational Biology*, *6*(8), e1000872. Descargado 2022-06-02, de https://dx.plos.org/10.1371/journal.pcbi.1000872 doi: 10.1371/journal.pcbi.1000872

Rashevsky, N. (1954, diciembre). Topology and life: In search of general mathematical principles in biology and sociology. *The Bulletin of Mathematical Biophysics*, *16*(4), 317–348. Descargado 2022-06-02, de http://link.springer.com/10.1007/BF02484495 doi: 10.1007/BF02484495

Rosen, R. (1972). *Some relational cell models: The metabolism-repair systems. In R. Rosen (Ed.), Foundations of mathematical biology* (Vols. 2, pp. 217–253). New York: Academic.

Rosen, R. (1991). *Life itself: a comprehensive inquiry into the nature, origin, and fabrication of life.* New York: Columbia University Press.

Rosen, R. (1993, junio). Drawing the boundary between subject and object: Comments on the mind-brain problem. *Theoretical Medicine*, *14*(2), 89–100. Descargado 2022-06-02, de http://link.springer.com/10.1007/BF00997269 doi:

10.1007/BF00997269

Rosen, R. (2000). *Essays on life itself*. New York: Columbia University Press.

Rosen, R. (2006, marzo). Autobiographical reminiscences of robert rosen. *Axiomathes*, *16*(1-2), 1–23. Descargado 2022-06-04, de http://link.springer.com/10.1007/s10516-006-0001-6 doi: 10.1007/s10516-006-0001-6

Rubin, S., y Crucifix, M. (2021, julio). Earth's complexity is non-computable: the limits of scaling laws, nonlinearity and chaos. *Entropy*, *23*(7), 915. Descargado 2022-06-02, de https://www.mdpi.com/1099-4300/23/7/915 doi: 10.3390/e23070915

Ryle, G. (2021). *The concept of Mind (Senior series) by Gilbert Ryle*. Barnes & Noble Books. (original-date: 1905)

Schreiner, P. R. (2020, noviembre). Quantum mechanical tunneling is essential to understanding chemical reactivity. *Trends in Chemistry*, *2*(11), 980–989. Descargado 2022-06-02, de https://linkinghub.elsevier.com/retrieve/pii/S258959742030215X doi: 10.1016/j.trechm.2020.08.006

Schrödinger, E. (2021). *What is life? the physical aspect of the living cell*. Cambridge ; New York: The Folio Society. (original-date: 1944)

Shmailov, M. M. (2016). *Intellectual pursuits of Nicolas Rashevsky: the queer duck of biology*. New York, NY: Springer Berlin Heidelberg.

Sipser, M. (2014). *Introduction to the theory of computation* (Third edition, international edition ed.). Australia Brazil Japan Korea Mexiko Singapore Spain United Kingdom United States: Cengage Learning.

Soare, R. I. (1996, septiembre). Computability and recursion. *Bulletin of Symbolic Logic*, *2*(3), 284–321. Descargado 2022-06-02, de https://www.cambridge.org/core/product/identifier/S1079898600007836/type/journal_article doi: 10.2307/420992

Tomás de Aquino, S. (2001). *Suma de Teología I. Traducido por José Martorell Capó*. Biblioteca de Autores Cristianos.

Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, *s2-42*(1), 230–265. Descargado 2022-06-02, de http://doi.wiley.com/10.1112/plms/s2-42.1.230 doi: 10.1112/plms/s2-42.1.230

Varela, F., Maturana, H., y Uribe, R. (1974, mayo). Autopoiesis: The organization of living systems, its characterization and a model. *Biosystems*, *5*(4), 187–196. Descargado 2022-06-02, de https://linkinghub.elsevier.com/retrieve/pii/0303264774900318 doi: 10.1016/0303-2647(74)90031-8

Zhang, L., Williams, R. A., y Gatherer, D. (2016, enero). Rosen's (M,r) system in unified modelling language. *Biosystems*, *139*, 29–36. Descargado 2022-06-02, de `https://linkinghub.elsevier.com/retrieve/pii/S0303264715002051` doi: 10.1016/j.biosystems.2015.12.006

Érdi, P., y Tóth, J. (1989). *Mathematical models of chemical reactions: theory and applications of deterministic and stochastic models*. Manchester: Manchester University Press.