Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Doctorado en Ciencias en Ciencias de la Computación

Modelos de inferencia y circuitos lógicos basados en desplazamiento de hebras de ADN

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Doctor en Ciencias

Presenta:

Nelson Emmanuel Ordóñez Guillén

Ensenada, Baja California, México

2019

Tesis defendida por

Nelson Emmanuel Ordóñez Guillén

y aprobada por el siguiente Comité

Dr. Israel Marck Martínez Pérez
Director de tesis

Dr. Carlos Alberto Brizuela Rodríguez

Dr. Hugo Homero Hidalgo Silva

Dr. Alejandro Huerta Saquero

Dr. Iñaki Sainz de Murieta Fuentes

Dr. Ubaldo Ruiz López
Coordinador del Posgrado en Ciencias de la Computación

Dra. Rufina Hernández Martínez
Directora de Estudios de Posgrado

Resumen de la tesis que presenta Nelson Emmanuel Ordóñez Guillén como requisito parcial para la obtención del grado de Doctor en Ciencias en Ciencias de la Computación.

Modelos de inferencia y circuitos lógicos basados en desplazamiento de hebras de ADN

Resumen aprobado por:

Dr. Israel Marck Martínez Pérez
Director de tesis

La alta predictibilidad y especificidad de las interacciones de emparejamiento entre bases nitrogenadas postuladas por Watson y Crick, ha dado paso al nacimiento de la nanotecnología dinámica del ADN. Dentro de esta área, uno de los mecanismos de reacción que ha sido de particular interés es el desplazamiento de hebras. La programabilidad, relativa simpleza y versatilidad de sus primitivas, ha permitido el desarrollo de una variedad de dispositivos entre los que se cuentan: compuertas lógicas, motores, caminadores de ADN y artefactos de cerradura. En la presente investigación estamos interesados en dos aplicaciones: mecanismos de razonamiento lógico y circuitos lógicos. Como objetivo primario se plantea el diseño de dispositivos capaces de generar rutas de inferencia lógica que funcionen catalíticamente. En este contexto, inspirados en uno de los principales enfoques del estado del arte en programación lógica biomolecular, se introduce un esquema de codificación de hechos, reglas y consultas. Con base en esta codificación, se implementan rutas de inferencia, realizando procedimientos de encadenamiento hacia adelante y hacia atrás. Las reacciones que hacen posible estos razonamientos proceden de manera catalítica, aprovechando el mecanismo de desplazamiento de hebras con intercambio de puntos de apoyo. Esta idea se basa en reacciones de repostaje, en donde una molécula desechable se intercambia por una señal útil, preservando así su disponibilidad para reaccionar múltiples veces. Para evaluar el comportamiento cinético de los diseños propuestos, se realizan simulaciones deterministas y estocásticas de los modelos bioquímicos. Los resultados muestran un desempeño superior en el esquema catalítico comparado con su contraparte no catalítica. Por otro lado, como objetivo secundario de este trabajo, se aborda el asunto de la escalabilidad en la implementación de circuitos lógicos con compuertas sube-baja. Para ello, se propone el uso de dos puntos de apoyo en dichas compuertas para distinguir señales en estado alto (“1” lógico) de las señales en estado bajo (“0” lógico), en la lógica de doble carril. Además de la ventaja inmediata de usar la mitad de secuencias con esta modificación, se produce una segmentación en los circuitos que disminuye las reacciones no productivas aproximadamente a la mitad. Como consecuencia, el desempeño cinético del esquema propuesto resulta más eficiente que el original, aumentando además a medida que la complejidad de los circuitos se incrementa.

Palabras clave: Programación lógica, circuitos lógicos, desplazamiento de hebras de ADN, reacciones catalíticas, escalabilidad

Abstract of the thesis presented by Nelson Emmanuel Ordóñez Guillén as a partial requirement to obtain the Doctor of Science degree in Computer Science.

Models for inference and logic circuits based on DNA strand displacement

Abstract approved by:

Dr. Israel Marck Martínez Pérez
Thesis Director

The high predictability and specificity of DNA Watson-Crick base pairing have enabled the advent of dynamic DNA nanotechnology. Within this research area, the DNA strand displacement reaction mechanism has increasingly attracted the interest. The programmability and relative simplicity along with the versatility of its primitives, have allowed the development of a variety of devices, such as logic gates, motors, DNA walkers, and lock mechanisms. In this research, we are interested in two applications: the implementation of logic reasoning mechanisms and logic circuits. As a primary objective, this work tackles the design of devices capable of generating catalytic inference paths. In this context, inspired by the main state-of-the-art work in biomolecular logic programming, we introduce a novel codification scheme for facts, rules, and queries. Based on this encoding, inference paths are implemented by following backward- and forward-chaining procedures. The reactions that make these reasonings possible proceed in a catalytic fashion, harnessing the toehold exchange strand displacement mechanism. The key idea relies on fueling reactions, where disposable molecules are interchanged with useful signals, preserving their availability to react multiple times. To evaluate the kinetic behavior of the proposed designs, deterministic and stochastic simulations of biochemical models were carried out. Results show the enhanced performance of the catalytic scheme against its non-catalytic counterpart. On the other hand, as a secondary objective, this work addresses the issue of scalability in the implementation of see-saw gates. To this end, we propose the use of two toeholds in such gates, to encode high (“1”) signals separately from low (“0”) ones, within dual-rail logic. In addition to the immediate advantage of reducing the required recognition domain sequences by half, this modification produces a virtual segmentation in the circuits, curtailing time-consuming unproductive reactions to roughly half. As a consequence, the kinetic behavior of the modified scheme shows a more efficient performance when compared with the original, and this improvement increases along with the complexity of the circuits.

Keywords: Logic programming, logic circuits, DNA strand displacement, catalytic reactions, scalability

Dedicatoria

A mi madre...

Agradecimientos

A mi padre, de quien tomo como ejemplos la dedicación, tenacidad, deseos de superación que le permitieron salir adelante a pesar de las adversidades. A mi familia, por estar siempre ahí para mí.

A mi director de tesis, Dr. Israel Marck Martínez Pérez, por la confianza depositada en mí y por la orientación proporcionada a lo largo de la tesis. A los integrantes de mi comité de tesis, Dr. Carlos Alberto Brizuela Rodríguez, Dr. Hugo Homero Hidalgo Silva, Dr. Alejandro Huerta Saquero y Dr. Iñaki Sainz de Murieta Fuentes, por todo el tiempo dedicado y por sus valiosos comentarios, consejos y observaciones que guiaron el desarrollo del presente trabajo.

A los investigadores del departamento de Ciencias de la Computación, por su gran calidad tanto académica como humana, y por todos los conocimientos que adquirí de ellos.

A los amigos que me acompañaron en esta parte de mi vida. No quiero enumerarlos por temor a omitir alguno y confío en que sabrán incluirse. Hubiera sido imposible culminar esta etapa sin compartir tan gratos y amenos momentos que quedarán indeleblemente grabados en mis recuerdos.

Al Centro de Investigación Científica y de Educación Superior de Ensenada por permitirme ser parte de su comunidad, la cual es una de las mejores instituciones del país en cuanto a formación científica. Al personal administrativo por todas las atenciones recibidas.

Por último, al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de doctorado. No. de becario: 262448

Tabla de contenido

Página

Tabla de contenido (continuación)

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de figuras (continuación)

Lista de figuras (continuación)

Lista de figuras (continuación)

Figura Página

Lista de figuras (continuación)

Lista de figuras (continuación)

Lista de figuras (continuación)

Lista de figuras (continuación)

Figura Página

Lista de figuras (continuación)

Lista de tablas

Capítulo 1. Introducción y estado del arte

El término cómputo es universalmente definido como la relación explícita entre un conjunto de entradas con un conjunto de salidas. Al analizar esta definición, nos damos cuenta que en la naturaleza se encuentran infinidad de procesos inherentemente computacionales. Esta noción es la que da origen a un área de investigación llamada cómputo natural (Nunes de Castro, 2007). Particularmente, considerando las interacciones que ocurren a escala molecular, se puede considerar que ciertas reacciones químicas asemejan procesos computacionales, al tomar como entrada moléculas reactantes y generar moléculas producto como salida. Esta interesante idea, inicialmente concebida por el célebre físico Richard Feynman, y planteada en su charla titulada "*There's plenty of room at the bottom*"(Feynman, 1960), dio origen a lo que más adelante sería el campo de la nanotecnología.

Centrándonos en el área de la biología molecular, estas perspectivas visionarias se remontan hasta John Von Neumann, quien interpretó el crecimiento y la reproducción como tareas de procesamiento de información (von Neumann, 1966); o Alan Turing, que estudió las bases químicas de la morfogénesis, vistas con un enfoque programático, en el crecimiento biológico (Turing, 1952). En efecto, los organismos vivos nos constituímos por células que llevan a cabo funciones que involucran el "procesamiento" de moléculas. Las moléculas esenciales para la vida, i.e. ácidos nucleicos y proteínas, interactúan entre sí y con otras, para generar los procesos que nos mantienen vivos. La idea del cómputo biomolecular es entender estos procesos bioquímicos para aplicarlos en el desarrollo de dispositivos capaces de realizar funciones de computadora.

1.1. Cómputo biomolecular: antecedentes y motivación

El cómputo biomolecular (o cómputo con ADN), es un área de investigación multidisciplinaria que se enfoca en el desarrollo de dispositivos, construidos a base de concentraciones de biomoléculas, que emulen algún comportamiento computacional. A partir del avance de Adleman (1994) que demostró la viabilidad experimental en el uso de ADN para resolver problemas clásicos de cómputo, se generó un interés por explorar técnicas de biotecnología como pasos de cómputo. En efecto, el resultado obtenido por Adleman fue tanto prometedor como inspirador: una simple molécula

de ADN podía almacenar una posible solución de un problema combinatorio y se podían diseñar procedimientos de laboratorio con O(n) bio-operaciones para discriminar soluciones inválidas.

Basados en esta idea, surgieron varios modelos cuyos esfuerzos estaban inicialmente dirigidos hacia el aprovechamiento de la alta densidad de información y paralelismo masivo como una nueva alternativa para enfrentar los problemas intratables por medios convencionales de cómputo (Lipton, 1995; Ouyang *et al.*, 1997). Sin embargo, no transcurrió mucho tiempo para que investigaciones posteriores pusieran en evidencia las principales limitantes del nuevo enfoque: complejidad espacial exponencial e imprecisión de algunas operaciones de laboratorio. De esta manera, mientras no se resuelvan estas cuestiones fundamentales, las computadoras moleculares no pueden competir en tiempo y precisión, con las convencionales basadas en silicio.

Fue así que el enfoque y los objetivos del cómputo biomolecular cambiaron, apartándose de la búsqueda de soluciones para problemas clásicos de cómputo y redirigiéndose a explotar la principal ventaja de las computadoras moleculares: la capacidad de procesar moléculas como entrada y generar una salida en base a otras moléculas. Con esta ventaja única, las disciplinas del cómputo biomolecular empezaron a buscar formas de crear dispositivos simples basados en concentraciones de biomoléculas, capaces de reaccionar ante estímulos producidos en ambientes biológicos, y que puedan ser interconectados de manera modular para el ensamblaje de sistemas más complejos (Qian y Winfree, 2009; Siuti *et al.*, 2013, 2014; Bonnet *et al.*, 2013).

Dispositivos de tal naturaleza exhiben la posibilidad de aplicaciones prometedoras en biomedicina y biotecnología, incluyendo el diagnóstico *in vitro* (Du y Dong, 2017; Lai *et al.*, 2014; Jung y Ellington, 2014; Smith *et al.*, 2013) y, en última instancia, la integración *in vivo* de los dispositivos para la entrega de cargas moleculares (Amir *et al.*, 2014; Li *et al.*, 2018). Con tales propósitos, un aspecto clave en el diseño de máquinas biomoleculares consiste en mantener el completo proceso de cómputo funcionando de manera autónoma (Benenson, 2012).

El desplazamiento de hebras de ADN es un mecanismo de reacción que procede de manera autónoma. Al estar relacionado con la construcción de redes de reacciones químicas (CRN’s, por sus siglas en inglés) el modelo de desplazamiento de hebras

proporciona un método universal de cómputo (Cook *et al.*, 2009; Qian *et al.*, 2011a). La utilidad de este mecanismo como un método de procesamiento de información fue primeramente demostrada por Yurke *et al.* (2000). En una reacción de desplazamiento de hebras típica, las hebras de ácidos nucleicos interactúan solamente en virtud de sus secuencias, iniciando con la unión de segmentos cortos complementarios denominados puntos de apoyo (*toeholds*[1]) y sin la necesidad de ningún otro reactivo adicional como las enzimas. De manera general, el paradigma computacional se basa en hebras señal que reaccionan con complejos conocidos de manera genérica como *compuertas*, para liberar nuevas señales de salida que a su vez, repiten el proceso produciendo cascadas de reacciones. Este mecanismo simple ha sido utilizado para la construcción de compuertas lógicas (Seelig *et al.*, 2006; Qian y Winfree, 2009), redes neuronales (Qian *et al.*, 2011b; Cherry y Qian, 2018), mecanismos de cerradura (Andersen *et al.*, 2009), motores (Venkataraman *et al.*, 2007; Omabegho *et al.*, 2009) e inferencia lógica (Rodríguez-Patón *et al.*, 2010; Sainz de Murieta y Rodríguez-Patón, 2012; Rodríguez-Patón *et al.*, 2014). Zhang y Seelig (2011) y, más recientemente, Guo *et al.* (2017) han revisado el amplio rango de dispositivos que se han implementado con este versátil mecanismo.

Una peculiaridad de los sistemas de desplazamiento de hebras es que pueden ser diseñados para permitir la recuperación de las señales de entrada, una vez que ya han reaccionado en sus respectivas compuertas. En otras palabras, una señal de entrada se puede comportar como un agente catalítico, liberando las señales de salida en múltiples compuertas sin agotarse. Esto se logra usando el mismo mecanismo de desplazamiento, solamente con la ayuda de moléculas desechables que serán intercambiadas por las moléculas útiles en una base una por otra. Esta característica ha jugado un rol importante en enfoques donde la amplificación o restauración de señales son necesarias para mantener la abstracción digital (Qian y Winfree, 2011; Qian *et al.*, 2011b; Cherry y Qian, 2018) así como en aplicaciones donde es deseable una discriminación precisa de ácidos nucleicos presentes de manera escasa, como es el caso de los microARNs séricos (Shi *et al.*, 2016; Graybill y Bailey, 2016).

La presente investigación aprovecha esta característica única para implementar, hasta la fecha, el primer esquema catalítico de inferencia lógica. De manera relaciona-

[1] Por simpleza y facilidad de lectura se permitirá el uso de la expresión en inglés a lo largo del presente documento manejándose los términos de manera indistinta

da, se aborda también una idea para introducir una adaptación al esquema de circuitos lógicos de Qian y Winfree (2009) y demostrar sus ventajas.

1.2. Justificación

Al integrar funciones biológicas para reproducir comportamientos computacionales, la motivación en el diseño de dispositivos de cómputo biomolecular se puede dividir en dos aspectos: computacional y biológico.

Por un lado, el objetivo del cómputo no convencional (i.e. no basado en silicio) resulta interesante *per se*. La exploración de nuevos sustratos para desarrollar computadoras que, en un futuro, podrían competir con las tradicionales en ciertas funciones. Un buen ejemplo de ello son los recientes avances que se han obtenido en almacenamiento de datos en ADN (Song y Reif, 2019). En ese sentido, el sistema de inferencia lógica introducido en la presente investigación, significa un pequeño paso en esa dirección, pues podría servir de base para que en un futuro, se implementen bases de datos deductivas a una escala que permita su aplicación en problemas reales.

Por otro lado, las aplicaciones biomédicas y biotecnológicas — actuales y potenciales — que tienen los dispositivos biomoleculares, han sido ampliamente documentadas (Simmel, 2007; Kahan *et al.*, 2008; Lu *et al.*, 2013; Chen *et al.*, 2015; Evans *et al.*, 2016). Hasta ahora, tales aplicaciones han considerado funciones computacionales relativamente simples. Sin embargo, en un futuro podrían abordarse funciones lógicas más complejas, por lo que contar con bloques de construcción escalables y confiables resulta primordial. Así, la mejora que se propone para el estado del arte en circuitos lógicos, podría representar una buena diferencia, en aplicaciones que involucren circuitos complejos.

1.3. Objetivos

1.3.1. Objetivo general

Diseñar dispositivos computacionales que resuelvan esquemas de razonamiento lógico, utilizando mecanismos biológicos existentes en la literatura. Los mecanismos

seleccionados deberán ser autónomos para permitir que los dispositivos propuestos operen de la misma manera.

1.3.2. Objetivos específicos

- Estudiar el estado del arte en inferencia lógica biomolecular identificando los diferentes problemas abordados, así como los mecanismos biológicos empleados, haciendo énfasis en aquellos que permiten realizar cómputo de manera autónoma.
- Estudiar el estado del arte en implementaciones de circuitos lógicos biomoleculares. Identificar nichos de investigación en los esquemas propuestos.
- Diseñar dispositivos de inferencia que funcionen de manera autónoma y que obtengan un desempeño correcto y eficiente.
- Obtener evidencia del comportamiento cinético de los dispositivos propuestos, a través de simulaciones que validen su viabilidad práctica.

1.4. Contribuciones

En vista de los resultados obtenidos a lo largo de la presente investigación, consideramos las siguientes contribuciones:

- Un nuevo esquema de codificación de hechos, reglas y consultas para la implementación de programación lógica en ADN, con funcionamiento catalítico.
- Un procedimiento simple para la búsqueda de secuencias apropiadas para las estructuras de ADN en modelos propuestos y su validación termodinámica.
- Una nueva representación molecular basada en motivo biológico (compuerta subebaja) predominante, hasta la fecha, en el área de circuitos lógicos moleculares. La relevancia de este aporte está, en buena medida, ligada a la relevancia de la compuerta que lo inspiró.

1.5. Trabajo previo

1.5.1. Inferencia lógica biomolecular

El cómputo biomolecular dedicó sus esfuerzos con miras a desarrollar diferentes enfoques de inferencia lógica casi desde sus inicios (con la innovación de Adleman (1994)). Es así como los primeros trabajos se caracterizan por considerar enfoques no autónomos con el objetivo de implementar una computadora molecular masivamente paralela. Más adelante, se empezaron a buscar métodos autónomos que permitieran la eventual aplicación de los diseños en tareas que así lo requieren, como lo es el diagnóstico *in vivo* y el desarrollo de fármacos inteligentes.

1.5.1.1. Modelos no autónomos

Los primeros enfoques de inferencia lógica biomolecular consistieron en protocolos que involucraban pasos de laboratorio (no autónomos), así como propuestas teóricas. El primero de ellos, introdujo un modelo abstracto para codificar las estructuras básicas y el mecanismo de retroceso (*backtracking*) del lenguaje PROLOG (Mihalache, 1997). Los hechos de la forma general $rel(\text{obj}_1, \text{obj}_2, \cdots, \text{obj}_n)$ se codificaban en trechos de ADN con secuencias identificadoras para la relación y los objetos, separadas por secuencias espaciadoras. El método de separación por hibridación se proponía para detectar si un hecho particular estaba en la base de conocimiento. Las reglas con la forma $\text{hecho} \leftarrow \text{hecho}_1, \text{hecho}_2, \cdots, \text{hecho}_m$ también se codificaban en hebras de ADN con las secuencias de cada hecho (secuencias complementarias en las premisas). De esta forma, los sectores premisa en las reglas hibridarían con los correspondientes hechos, en caso de existir, para formar un segmento doble en el cuerpo de la regla. Una enzima hipotética cortaría este segmento liberando la cabeza de la regla como un nuevo hecho en el sistema. El modelo también consideraba la sustitución de variables en una operación similar a los modelos de concatenación, usando secuencias especiales de reconocimiento para enzimas de restricción en las espaciadoras e introduciendo los segmentos correspondientes a objetos dentro del espacio cortado (Figura 1).

Posteriormente, un diseño experimental para una máquina de inferencia que ejecutaba el algoritmo de encadenamiento hacia atrás fue propuesto por Mulawka *et al.* (1998). Basados en la codificación del modelo de etiquetas (Roweis *et al.*, 1998),

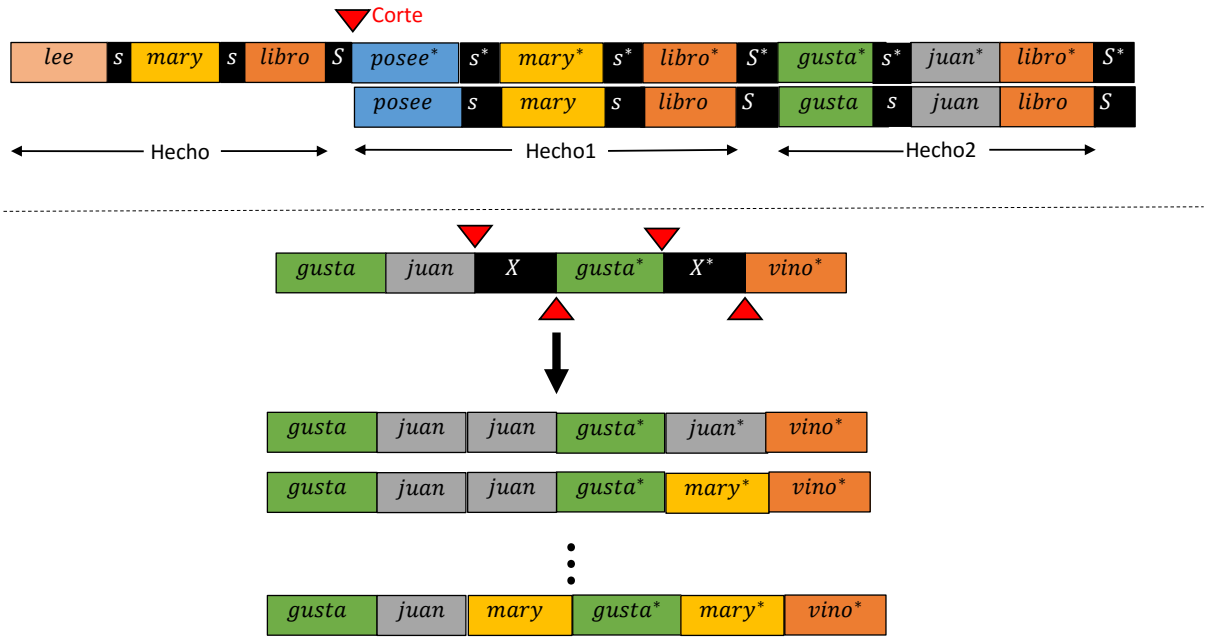

Figura 1. Codificación de Mihalache (1997). Panel superior: codicación de reglas e hibridación con correspondientes hechos en el cuerpo de la regla. Cuando el segmento doble se forma, la enzima corta la cabeza de la regla produciendo el nuevo hecho. Panel inferior: sustitución de variables por objetos. Método similar a los modelos de concatenación. Después del corte con enzimas de restricción las secuencias de los objetos se introducirían produciendo diferentes instancias de la regla.

utilizaron hebras de ADN para representar reglas, hechos e hipótesis. Acorde a esta codificación, hebras de memoria representaban las reglas que tenían la forma $P_1 \wedge P_2 \wedge \cdots \wedge P_n \rightarrow Q$, mientras que hebras etiqueta formaban los hechos y las hipótesis, complementarias a una premisa o conclusión en alguna regla. El encadenamiento hacia atrás se implementaba haciendo coincidir la secuencia del sector conclusión de una regla, con la primera premisa en otra, de manera que hibridaran, uniendo ambas reglas. Una ruta de verdad que probaba cierta hipótesis dada, se producía cuando se formaba una molécula doble completa y con extremos romos, a través de reacciones de hibridación y posterior ligado. Estas moléculas eran detectadas en dos muestras, agregando exonucleasas en una de ellas para digerir aquellas moléculas que no correspondían a rutas de verdad y visualizando los resultados por medio de electroforesis (Figura 2).

Otro protocolo experimental relacionado para el encadenamiento hacia atrás fue desarrollado poco después (Wasiewicz *et al.*, 1999, 2000). La resolución también estaba basada en el encadenamiento de reglas, pero en este caso considerando reglas simples de una sola premisa. Las reglas se codificaban con moléculas que contenían

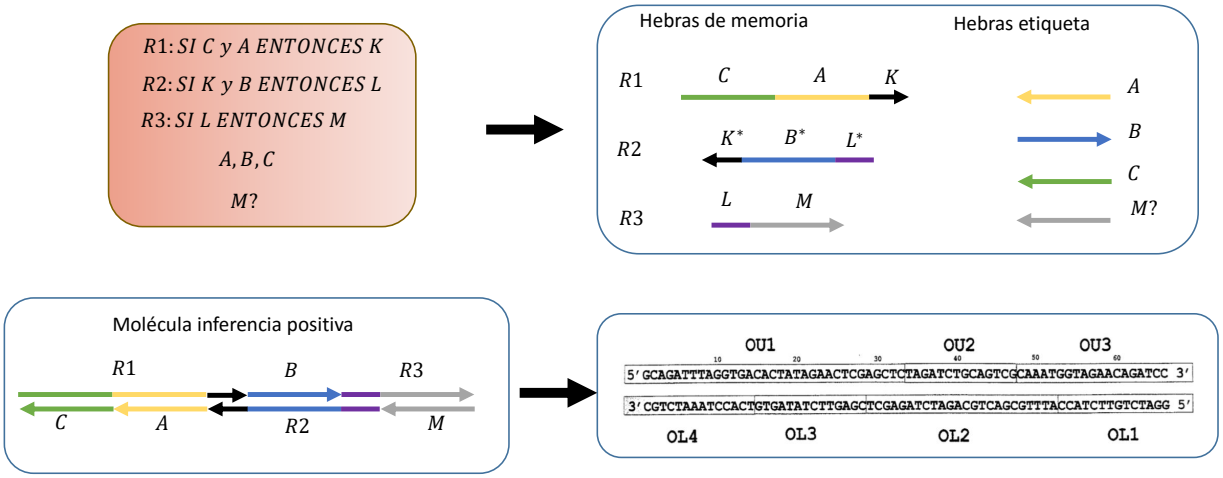

Figura 2. Caso de prueba resuelto por Mulawka *et al.* (1998). Las tres reglas hibridaban con los hechos y la hipótesis para formar una molécula doble completa que se amplificaba y detectaba por electroforesis en gel.

un segmento doble al centro y dos extremos pegajosos con secuencias para codificar el antecedente y consecuente. Una ruta de inferencia positiva se interpretaba cuando las reglas adecuadas — por ejemplo, $A \rightarrow B$ y $B \rightarrow C$ — hibridaban una con otra para generar la regla equivalente $A \rightarrow C$. Entonces, una molécula derivada de un plásmido con la estructura de la regla $C \rightarrow A$ se agregaba. Esta última regla se interpretaba como la consulta ¿C? (si era una consecuencia lógica del programa) dado el hecho A. Así, al hibridar y ligar estas moléculas relevantes, se producía un plásmido que podía ser amplificado y detectado por medios convencionales para indicar la inferencia positiva (Figura 3).

El cómputo de cláusulas de Horn propuesto de forma teórica como un marco de trabajo para el cómputo universal basado en ADN, se exploró en Kobayashi (1999). Este marco estaba basado en la técnica de *Whiplash PCR* de Hagiya *et al.* (1999). Este método bioquímico para la implementación de máquinas de estado en ADN, fue utilizado para derivar consecuencias lógicas de programas de Horn simples. Otro método mejorado para el cómputo de cláusulas de Horn con moléculas de ADN se reportó posteriormente en Uejima *et al.* (2002). Aquí, las derivaciones se realizaban a través del auto ensamblaje de moléculas ramificadas para lógica proposicional. Las variables se codificaban en cada una de las secuencias de extremos pegajosos. La hibridación de estos extremos en cláusulas con literales complementarias producía una molécula completamente hibridada, en caso de tener una inferencia positiva. El modelo fue ex-

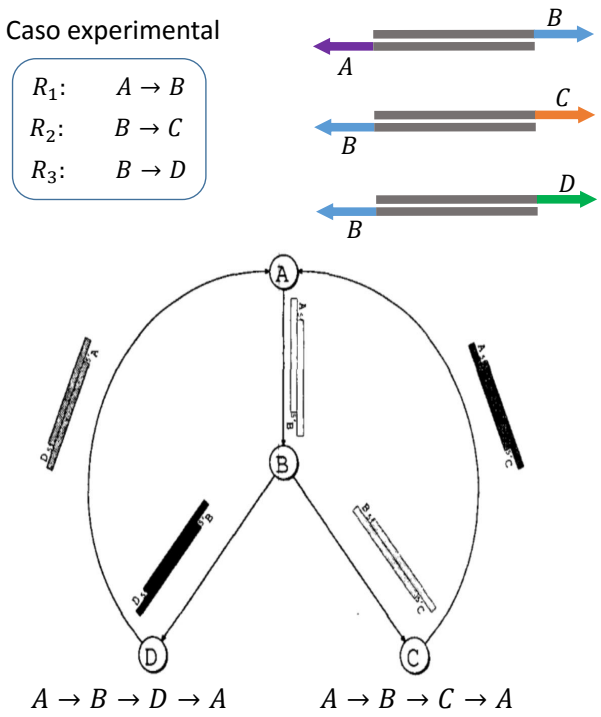

Figura 3. El caso de prueba experimental abordado en Wasiewicz *et al.* (1999). Hibridando reglas simples por sus extremos pegajosos se formaba una ruta de inferencia. Agregando moléculas derivadas de plásmidos denominadas fragmentos básicos de ADN (DBF, por sus siglas en inglés), se podía cerrar una molécula circular que se interpretaba como inferencia positiva. Figura adaptada de Wasiewicz *et al.* (1999).

tendido para abordar una clase restringida de programas de Horn en lógica de primer orden mediante el uso de auto ensamblaje de cadenas-mosaico propuestos por Winfree *et al.* (2001), requiriendo una cantidad constante de operaciones de laboratorio.

Lee *et al.* (2003) demostraron una prueba de concepto para el clásico problema de prueba de teoremas dentro de la lógica proposicional. Ellos utilizaron moléculas de ADN lineales (hebras), estructuras de horquilla (*hairpins*) y estructuras ramificadas para las cláusulas de una, dos y más de dos literales, respectivamente. Dado que la codificación incluía secuencias complementarias para una literal y su complemento, los pasos de resolución se realizaban por medio de reacciones de hibridación. Implementando el método de resolución por refutación, el primer paso consistía en añadir el complemento de la cláusula objetivo al tubo que contenía las moléculas del programa, buscando por medio de pasos de resolución obtener la cláusula nula: una molécula de ADN completamente hibridada y ligada. La detección por medio de amplificación y electroforesis en gel indicaba si la cláusula objetivo era una consecuencia lógica del

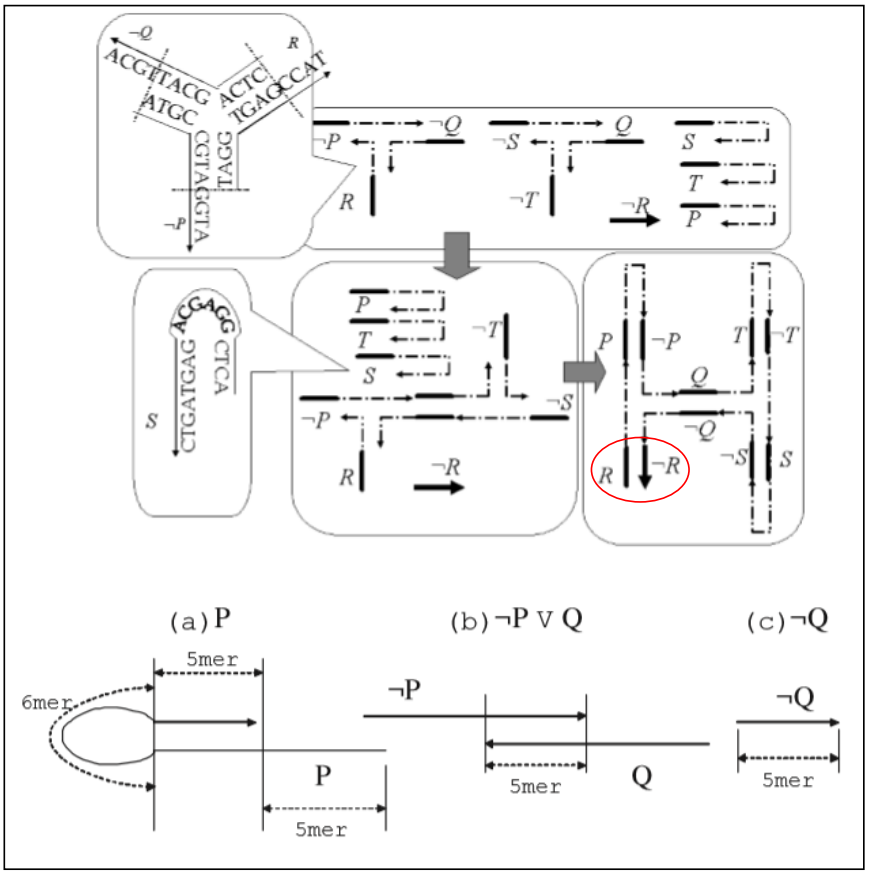

Figura 4. El esquema de codificación con moléculas lineales, horquillas y moléculas ramificadas para aplicar resolución por refutación de Lee *et al.* (2003). Arriba, el diagrama de un ejemplo que incluye varias cláusulas. A través de la hibridación de extremos pegajosos se busca la formación de la cláusula vacía, es decir, una molécula cerrada que empieza y termina con la cláusula objetivo y su negación (encerrado en el círculo rojo). Abajo, el caso simple que se implementó en laboratorio. Las cláusulas corresponden a un programa simple que aplica una regla *modus ponens*. Figura adaptada de Lee *et al.* (2003).

programa (Figura 4).

Más recientemente, un enfoque automatizado de este esquema para demostración de teoremas usando plataformas de microfluídos fue reportado (Lee *et al.*, 2012). El proceso de resolución por refutación fue implementado controlando el flujo de información entre canales y cámaras a través de válvulas neumáticas. En la cámara principal (en la que se realizaba el cómputo), todas las moléculas de ADN que codificaban las diferentes cláusulas se mezclaban con enzimas de ligado. Posteriormente, un paso de desplazamiento de hebras de ADN seguido de digestión con enzimas se realizaba en otra cámara para detectar los resultados computacionales a través de fluorescencia (Figura 5).

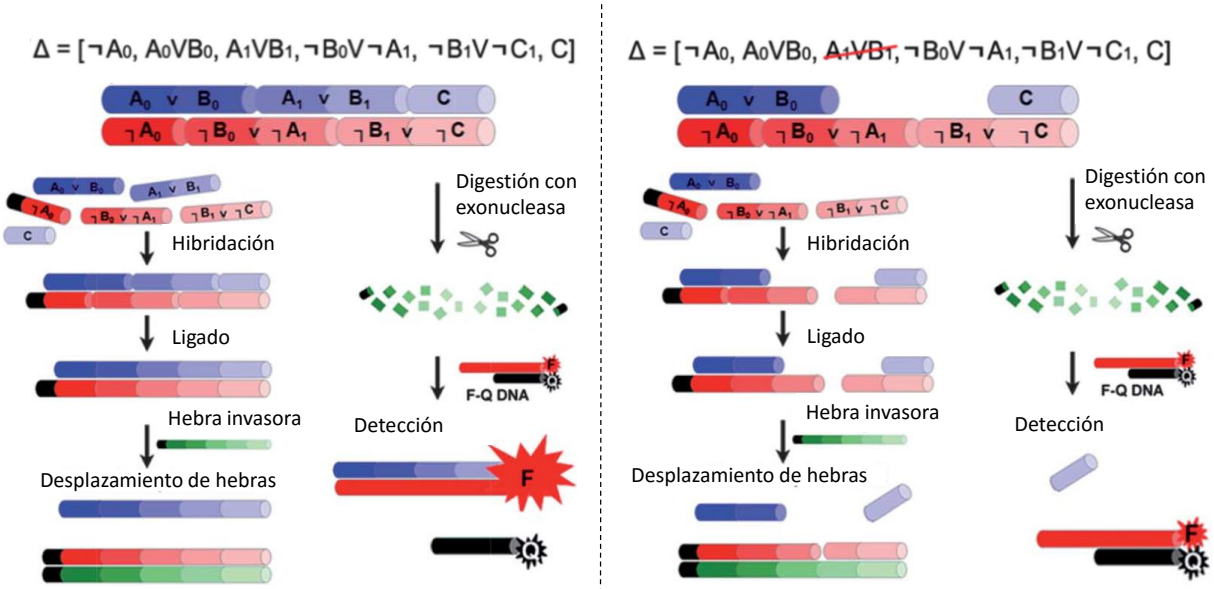

Figura 5. La prueba de concepto de prueba para resolución por refutación implementado de forma automatizada en el trabajo de Lee *et al.* (2012). El paso principal del cómputo se basaba en la hibridación de cláusulas con literales complementarias para formar una molécula doble (en caso de inferencia positiva). Para el desplazamiento de hebras se incluía un punto de apoyo en la cláusula objetivo $\neg A_0$ (marcada en negro). La hebra liberada en el desplazamiento reaccionaba con la molécula sonda para producir fluorescencia (solamente en caso de estar completa). Figura adaptada de Lee *et al.* (2012).

1.5.1.2. Modelos autónomos

La inferencia biomolecular autónoma fue primeramente abordada de manera teórica por Yan *et al.* (2007) con un modelo que consideraba la hibridación de extremos pegajosos y el corte con enzimas de restricción para realizar inferencia hacia adelante. El núcleo operativo de la máquina de inferencia se basaba en una clase IIB de enzimas de restricción (llamada PpiI). Todos los hechos y reglas de entrada de la base de conocimientos se codificaban como moléculas dobles de ADN con extremos pegajosos. Dentro del diseño, los hechos contenían la porción izquierda de la secuencia de reconocimiento de PpiI, mientras que las reglas mantenían la porción derecha. De esta forma, cuando un hecho se unía a su correspondiente regla, el complejo resultante juntaba ambas porciones permitiendo que la enzima actuara sobre él. El producto cortado, que consistía en un nuevo hecho, podía entonces hibridar en una siguiente etapa, con otra regla coincidente (Figura 6).

A la fecha, la única implementación de laboratorio de inferencia autónoma es la de Ran *et al.* (2009), quienes presentaron un modelo para la resolución de programas lógicos simples, adaptando un autómata de dos estados y dos símbolos (Benenson

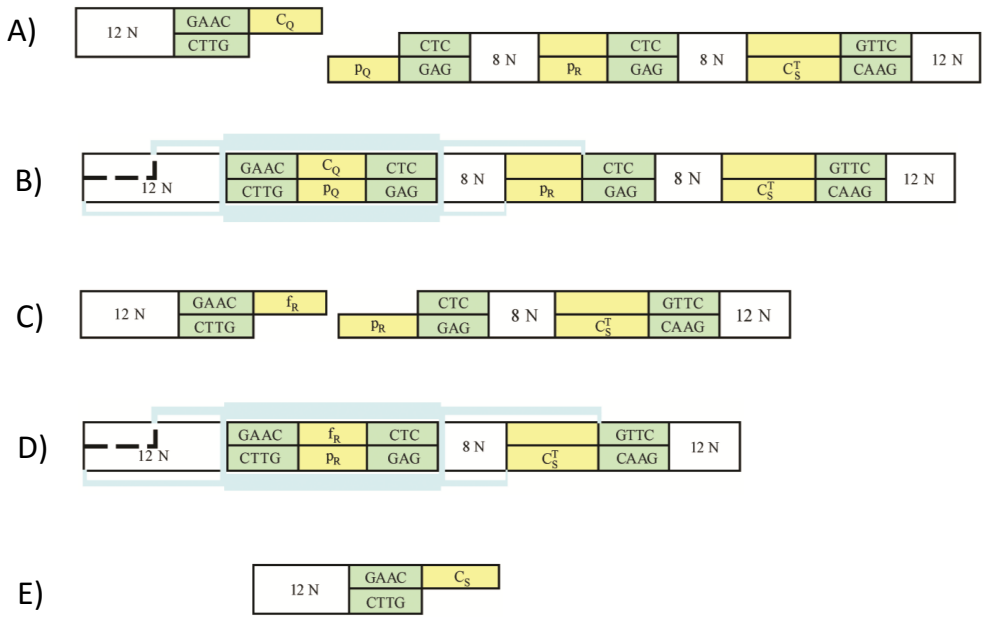

Figura 6. El esquema de inferencia hacia adelante con enzimas de Yan *et al.* (2007). A) La proposición Q hibrida con su extremo pegajoso a la regla $Q \wedge R \rightarrow S$. B) Al unirse los extremos, se forma el sitio de reconocimiento para la enzima PPiI. C) El corte abre otro extremo pegajoso para la siguiente proposición R. D) Se repite el proceso de corte. E) Al final, se libera la nueva proposición S. Figura adaptada de Yan *et al.* (2007).

et al., 2001). Su esquema de codificación incluía representación para las proposiciones, implicaciones y consultas en moléculas cortas de ADN. Específicamente, una proposición p se codificaba con una molécula doble de ADN con un extremo pegajoso de 4 nt, cuya secuencia identifica a la proposición. En el extremo opuesto del extremo pegajoso tenía agregados un fluoróforo y su respectivo supresor para detección de resultados. Correspondientemente, una consulta $¿p?$ se representa con una molécula parcialmente doble con una secuencia de extremo pegajoso complementaria a p. Dentro del segmento doble también se incluye un sitio de reconocimiento de la enzima FokI. Así, cuando los extremos pegajosos se unen, la endonucleasa puede entrar en acción, cortando la proposición p de una forma que no quedan suficientes pares de bases para mantener unidas ambas hebras, por lo que la separación de la molécula emisora de la supresora produce una respuesta positiva a la consulta.

Por otro lado, la molécula implicación $q \rightarrow p$ se representaba con una molécula horquilla de ADN, con un extremo pegajoso con la secuencia de la proposición del consecuente p. La horquilla estaba dispuesta con la secuencia de una de las hebras de la proposición del antecedente q. La reacción que liberaba esta hebra, se iniciaba con la

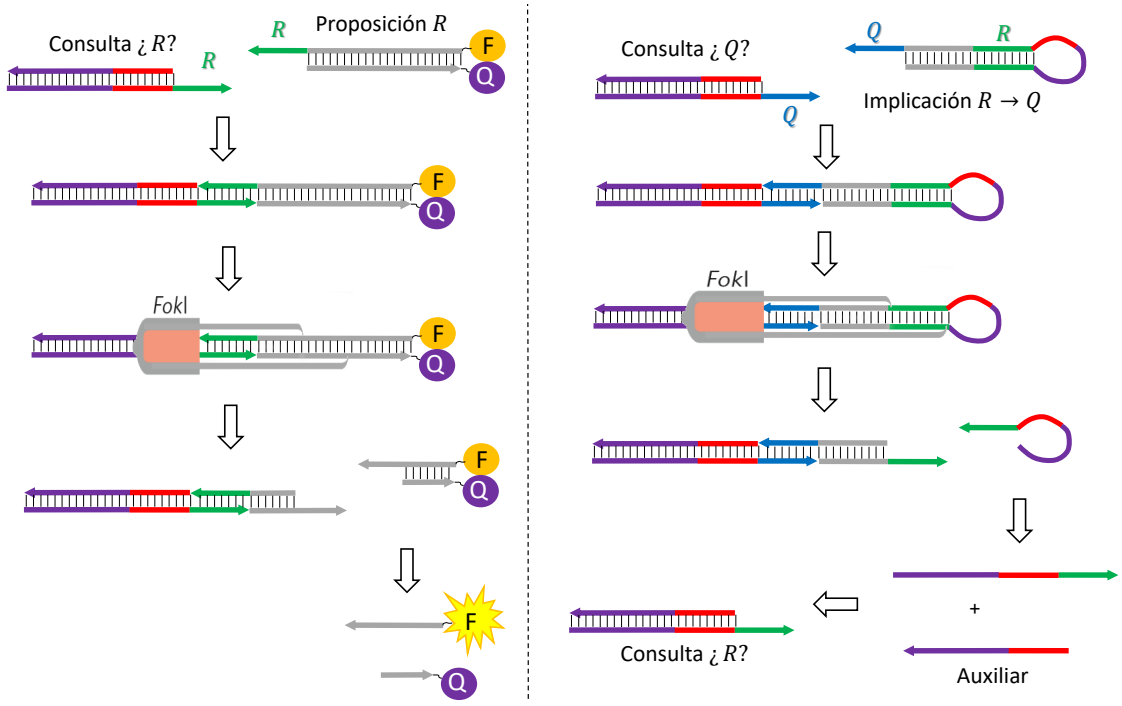

Figura 7. Resolución de programas lógicos simples de Ran *et al.* (2009). A la izquierda, una consulta se responde con su respectivo hecho. Ambas moléculas comparten extremos pegajosos complementarios que al unirse, permiten que la enzima FokI se una al complejo y corte la molécula en los puntos indicados. El segmento doble que contenía el fluoróforo y el supresor eventualmente se separaba, emitiendo fluorescencia. A la derecha, el mecanismo de retroceso (*backtracking*) donde una consulta de entrada genera una nueva consulta a partir de una implicación. La unión de extremos pegajosos y corte con FokI procede de manera similar. El corte libera la horquilla de la molécula que así, puede hibridar con la molécula auxiliar para formar una nueva consulta.

union de extremos pegajosos complementarios entre la consulta *¿p*? y la implicación. Una vez más, esta unión habilitaba a la enzima para producir el correspondiente corte, liberando la hebra en la horquilla. Finalmente, esta hebra hibridaba con una auxiliar (que era universal para todas las implicaciones) para formar una nueva consulta *¿q*? (ver Figura 7).

Otro modelo basado en corte con enzimas fue introducido más tarde en Rogowski y Sosík (2014). Ellos diseñaron un esquema de codificación proposicional usando el concepto de empalmes de ADN (*DNA splicing*), un paradigma biomolecular en el que se concatenan segmentos dobles de ADN, resultantes después de cortarlos con enzimas de restricción específicas. Moléculas especialmente diseñadas con sitios de reconocimiento para la enzima BseXI y con extremos pegajosos de 4 nt representaban proposiciones. Extremos pegajosos complementarios se usaban para proposiciones negadas. El cómputo consistía en secuencias de reacciones de hibridación — catali-

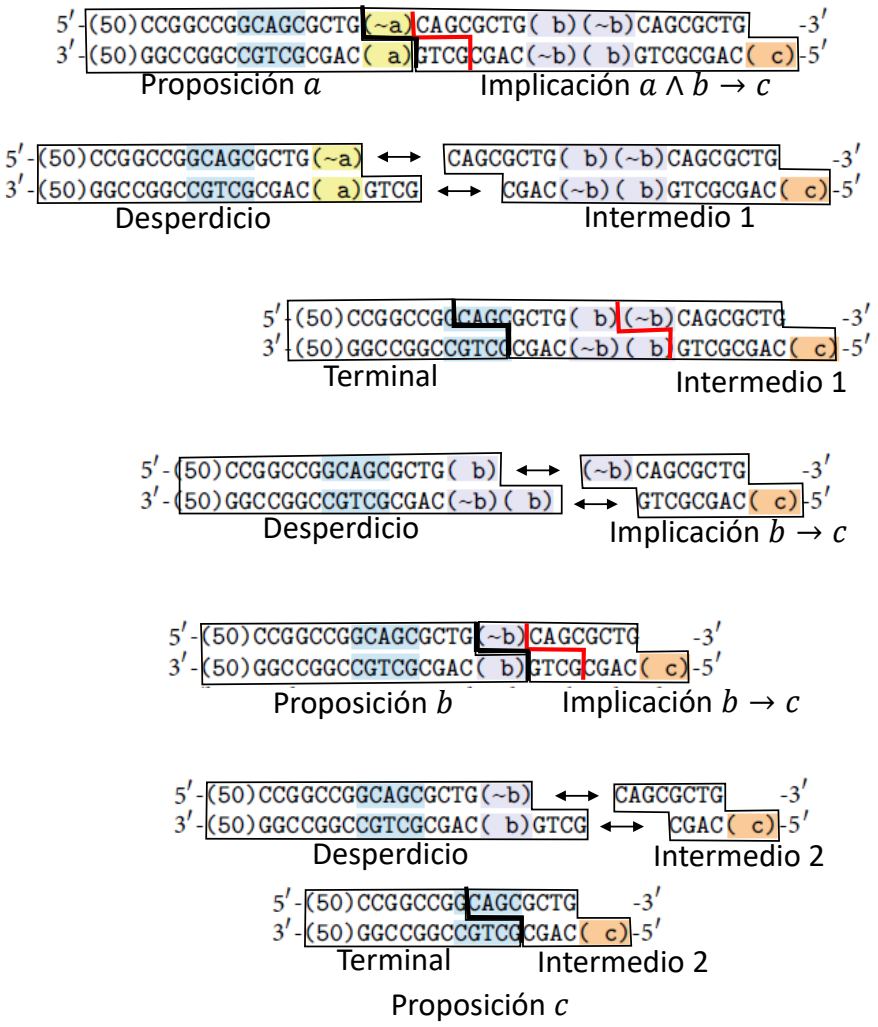

Figura 8. Secuencia de reacciones de concatenación en el modelo de Rogowski y Sosík (2014). Los puntos de hibridación de los extremos pegajosos se marcan en líneas negras y los de corte con enzima BseXI, en rojo. Los extremos pegajosos que codifican las diferentes variables se indican y aparecen resaltados junto con el sitio de reconocimiento para BseXI. Figura adaptada de Rogowski y Sosík (2014).

zadas por ligasa — y corte. En la Figura 8 se muestra un ejemplo de la aplicación de este modelo para la resolución con una regla doble $a \wedge b \rightarrow c$ y las proposiciones a y b, que produce la nueva proposición c. Primero, la proposición a se una a la implicación $a \wedge b \rightarrow c$ por medio de los extremos pegajosos complementarios, se produce el corte y la separación en una molécula no útil (desperdicio) y una molécula intermedio1. Esta última se une a otra molécula constante especial denominada Terminal. La unión genera un nuevo corte que produce otra molécula desperdicio junto con la implicación $b \rightarrow c$. A partir de ahí, se repite el procedimiento con la proposición b para, al final, unirse el intermedio2 resultante con la molécula Terminal. Esta molécula final tiene la

estructura de la nueva proposición *c*. El modelo también incluía implicaciones con disyunción en la premisa y una reacción de inconsistencia (entre una literal positiva y su negación) para aplicar resolución por refutación. La detección de resultados se basaba en el tamaño de las moléculas resultantes, que se distinguía mediante electroforesis en gel.

La inferencia lógica usando reacciones de desplazamiento de hebras se exploraron por primera vez por Rodríguez-Patón *et al.* (2010) con un enfoque que implementa las operaciones de inferencia *modus ponens/tollens*. Entre las principales diferencias que surgieron con este modelo están que (1) no se necesitan enzimas, y (2) los valores lógicos negados (“falso”) se codifican con las secuencias complementarias Watson-Crick de las correspondientes proposiciones verdaderas. En particular, esta característica de codificación dual tenía un importante rol en el sistema, permitiendo la cancelación de errores, inferencia bidireccional y proposiciones negadas como salidas válidas.

Una versión mejorada de este modelo se presentó después en Sainz de Murieta y Rodríguez-Patón (2012) con el propósito de diseñar dispositivos inteligentes en ADN capaces de realizar diagnósticos genéticos *in vitro*. El esquema representaba implicaciones con premisas simples y en conjunción. Las proposiciones se formaban de hebras simples con un par de toeholds concatenados. Las proposiciones negadas se componían de dominios complementarios, de manera que P y $\neg P$ hibridaban en una molécula doble. En la Figura 9 se muestra esta codificación y la operación del modelo para obtener R, con el programa lógico que incluye $P \wedge Q \rightarrow R$, P y Q. Inicialmente, la entrada P se unía a la implicación en el toehold expuesto $\neg p_2$, seguido de la migración de ramas que liberaba el toehold p_1. Una molécula auxiliar podía entonces unirse al toehold liberado para desplazar la hebra formando una molécula doble inerte (desperdicio). A su vez, este paso liberaba un nuevo toehold $\neg q_2$ que permitía que la siguiente entrada Q repitiera el proceso para, finalmente, terminar en una molécula parcialmente doble que contenía un segmento simple con los toeholds r_1 y r_2. Este segmento, que representaba la nueva proposición R, podía entonces reaccionar en cascada con una nueva implicación o reaccionar con alguna molécula detectora para producir una inferencia positiva.

El mismo grupo de trabajo presentó un modelo de cómputo en ADN basado en el desplazamiento de hebras de cuatro vías que realiza deducciones lógicas en fórmulas

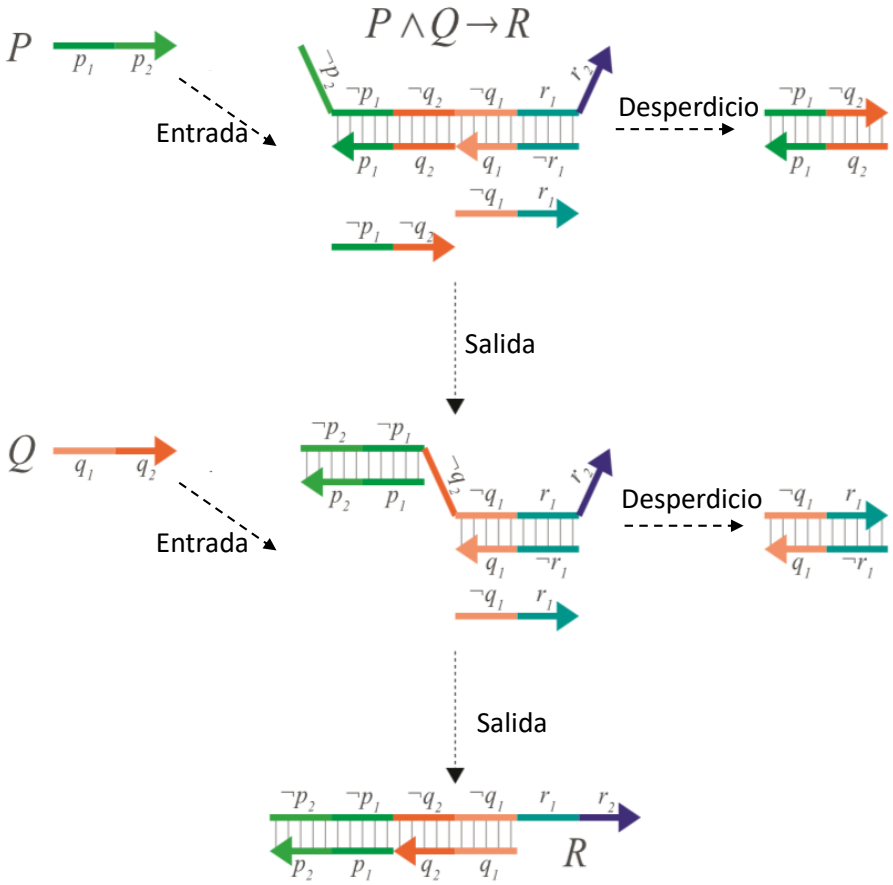

Figura 9. Inferencia hacia adelante con desplazamiento de hebras de ADN propuesto en Sainz de Murieta y Rodríguez-Patón (2012). El programa lógico incluye la regla $P \wedge Q \rightarrow R$ y las proposiciones P y Q. La cascada de reacciones de desplazamiento produce al final una molécula que representa la proposición inferida R. Figura adaptada de Sainz de Murieta y Rodríguez-Patón (2012).

booleanas en forma normal conjuntiva (Rodríguez-Patón *et al.*, 2014). La codificación de las cláusulas incluía una hebra cubrimiento con un extremo pegajoso expuesto para cada literal, manteniendo las literales afirmadas ordenadas a partir del extremo 3’ de la hebra base y las negadas hacia el extremo 5’ (ver Figura 10). Las literales afirmadas y negadas tenían extremos pegajosos con secuencias complementarias. Así, dos cláusulas con literales complementarias podían efectuar un paso de resolución que producía una molécula resolvente. El proceso de resolución por refutación se generaba agregando al sistema el complemento de la proposición objetivo. Si se formaba una molécula doble con un peso (i.e. cantidad de nucléotidos) determinado *a priori* en el modelo, se podía determinar que correspondía a una cláusula nula por medio de electroforesis en gel. Como prueba de concepto para tal esquema, se propuso la solución

teórica para el problema de satisfacibilidad (SAT).

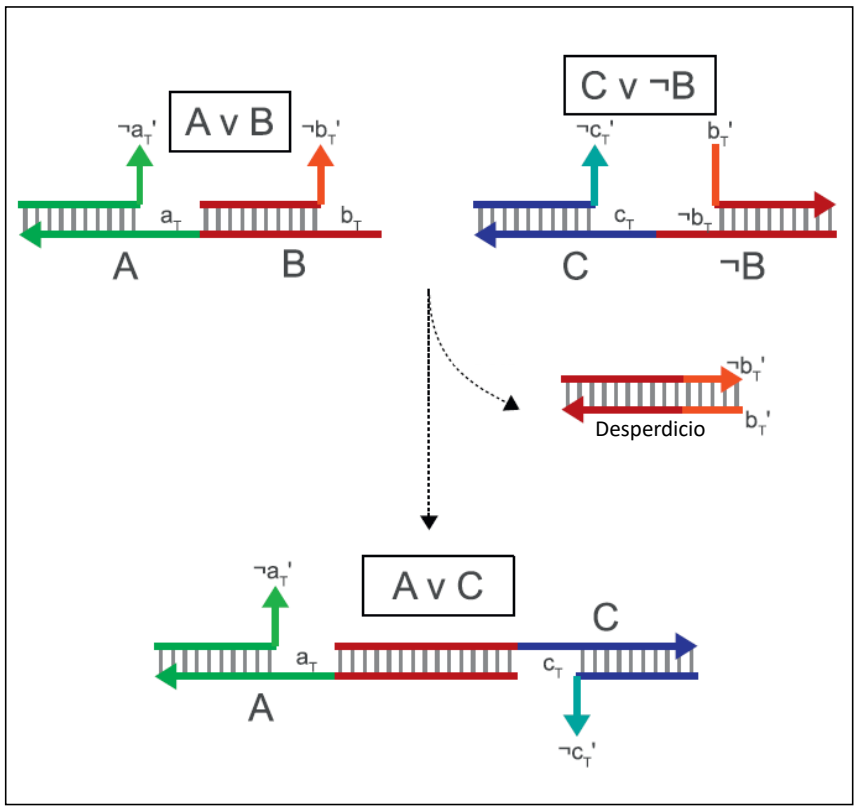

Figura 10. Un paso de resolución entre cláusulas del modelo propuesto por Rodríguez-Patón *et al.* (2014). Las literales complementarias *B* y ¬*B* cuyas hebras cobertura eran complementarias, podían hibridar por sus extremos pegajosos expuestos, produciendo una molécula desperdicio. Dado que las secuencias en la hebra base también eran complementarias, las cláusulas terminaban unidas, representando una molécula resolvente (*A* v *C*, en el ejemplo) . Figura adaptada de Rodríguez-Patón *et al.* (2014).

1.5.2. Circuitos lógicos basados en desplazamiento de hebras

Una de las aplicaciones de cómputo biomolecular más estudiadas consiste en la implementación de circuitos lógicos biomoleculares. En efecto, con los objetivos futuros de aplicar mecanismos de cómputo en el diagnóstico de enfermedades, las funciones lógicas son probablemente el modelo de computación más adecuado. Esto es, muchos objetivos pueden ser trazados en términos de funciones lógicas. Por ejemplo, un sistema reportero capaz de producir GFP cuando la célula está en etapa de mitosis bajo condiciones de estrés osmótico, nececita calcular la siguiente función lógica: GFP = [marcador mitótico] AND [marcador de estrés osmótico] (Benenson, 2012).

Se han desarrollado circuitos lógicos biomoleculares utilizando varios mecanismos biológicos, entre los principales: circuitos basados en redes de regulación genética

(Bonnet *et al.*, 2013; Siuti *et al.*, 2013, 2014), circuitos basados en ribozimas y desoxirribozimas (Stojanovic y Stefanovic, 2003; Macdonald *et al.*, 2006; Win y Smolke, 2008) y circuitos basados en desplazamiento de hebras (Seelig *et al.*, 2006; Zhang *et al.*, 2007; Qian y Winfree, 2011). En el presente documento estamos interesados en estos últimos, particularmente en la compuerta sube-baja de Qian y Winfree (2009).

1.5.2.1. Compuertas sube-baja.

La construcción de circuitos con el motivo sube-baja (del inglés *see-saw*), es uno de los paradigmas más escalables y simples que ha mantenido el mayor interés dentro del área del cómputo biomolecular. Este paradigma tiene cuatro componentes básicos (ver ejemplos en Figura 11):

- Una molécula cable (*wire*) que hace las veces de señal propagándose en cascadas de reacciones entre diferentes compuertas. Tienen un formato uniforme que consiste en un dominio de reconocimiento izquierdo, un toehold y un dominio de reconocimiento derecho. Por ejemplo, la molécula $3'-S_1TS_2-5'$ (abreviada como $w_{1,2}$) es la entrada, mientras que $3'-S_1TS_2-5'$ ($w_{2,3}$) es la salida.
- Una molécula combustible que tiene el mismo formato que las molécula cable, con la diferencia que el único dominio relevante es el izquierdo. Los combustibles intervienen en la amplificación de señales liberando muchas salidas de una compuerta sube-baja con relativamente poca entrada ($w_{2,7}$, en el ejemplo).
- Una molécula compuerta (*gate*) que consiste en un complejo formado por dos hebras: una hebra base (sustrato) con el complemento de un dominio de reconocimiento al centro (S_2^*) y dos toeholds (T^*) en los extremos y una hebra señal de salida que se mantiene unida a la base por su dominio de reconocimiento izquierdo (S_2) y su toehold (T). La notación abreviada para esta molécula es de la forma $G_{2:2,3}$, para indicar el dominio unido a la base (2:2) y el dominio libre (3) de la molécula señal adjunta.
- Una molécula umbral (*threshold*) compuesta por un sustrato que contiene un toehold extendido con algunos nucleótidos del dominio de reconocimiento izquierdo para alguna molécula señal. El dominio de reconocimiento derecho está

inicialmente cubierto por una hebra que se considera desperdicio inerte una vez liberada. En el ejemplo, la notación es $T_{2,3:3}$ para indicar los dominios involucrados en el subíndice, de manera similar a la compuerta.

- Finalmente, el esquema incluye una molécula reportera con la estructura típica. Contiene un toehold libre para iniciar la reacción y un dominio de reconocimiento correspondiente al dominio de reconocimiento derecho de alguna molécula cable. La adición de un fluróforo junto con su supresor al final del dominio de reconocimiento permite la detección visible de concentraciones finales de salida. La notación en la figura de ejemplo (Rep_4) indica en el subíndice el dominio de reconocimiento.

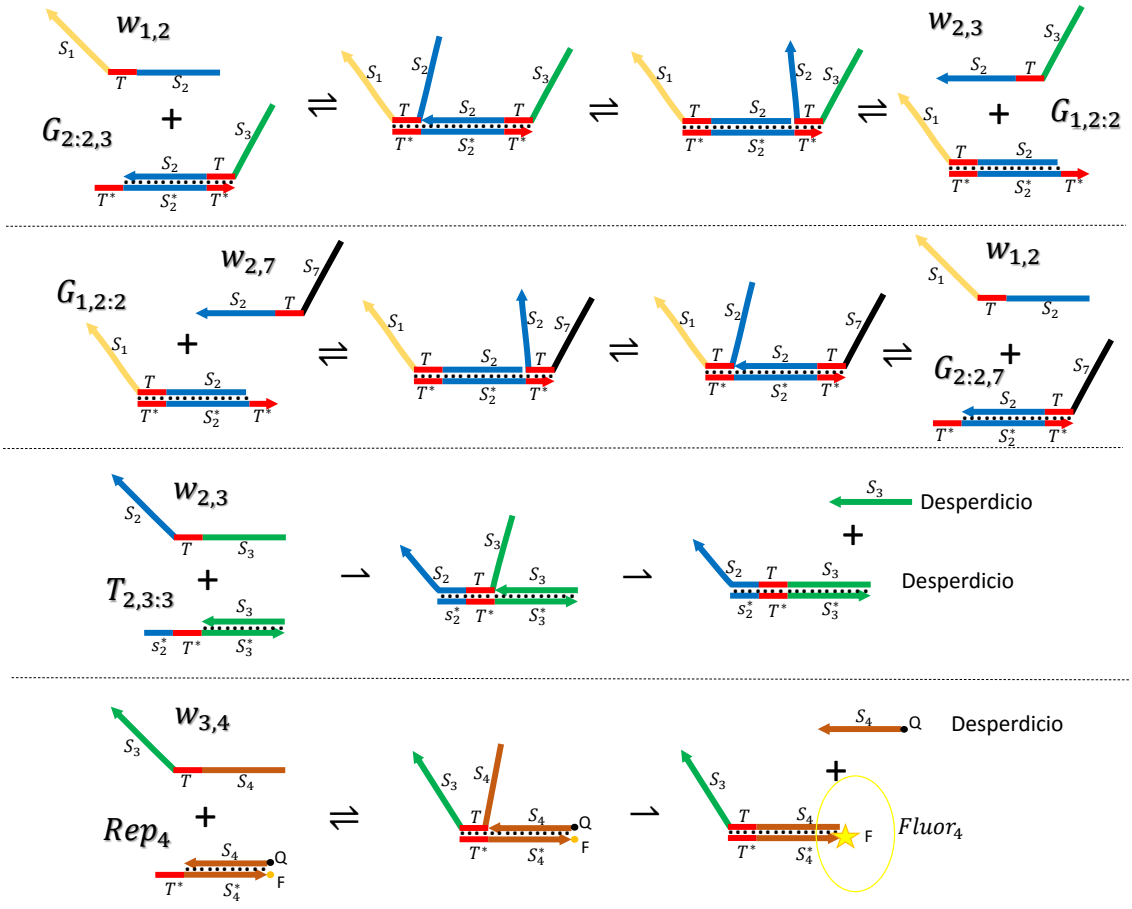

Figura 11. Las cuatro reacciones básicas del esquema sube-baja. Del panel superior al inferior: reacción sube-baja, reacción de repostaje (sube-baja inversa con molécula combustible), reacción de umbralización, reacción de reporte.

La disposición de estas cuatro moléculas permite la implementación de tres reacciones básicas dentro del esquema sube-baja. La primera es la reacción que da nombre

a la compuerta (sube-baja), donde una hebra cable libre de entrada se une, a través del toehold expuesto, a una compuerta coincidente e inicia un paso de migración de ramas de 3 vías. Esta reacción de intercambio de toeholds produce la liberación estequiométrica de la hebra cable de salida en la compuerta. El complejo resultante, contiene ahora la hebra cable de entrada unida y el toehold izquierdo ha sido activado, por lo que la señal de salida puede hibridar de nuevo y liberar la hebra de entrada (i.e., la reacción general es reversible). Este comportamiento es el que inspiró el nombre de la reacción y el motivo principal del esquema. Es importante notar que la inclusión de una molécula combustible, permite la realización de un ciclo catalítico, en el que la señal unida es liberada permitiendo que pueda participar en otra reacción con alguna compuerta nueva. Por otro lado, una reacción de umbralización se produce entre una hebra señal y correspondiente umbral. Aquí, la molécula umbral “se come” la señal de manera irreversible y a una tasa de reacción mucho mayor debido al toehold extendido. Este proceso produce dos moléculas consideradas desperdicio inerte. Por último, la reacción de reporte también se produce de manera irreversible con una señal y una reportera, pero en este caso a una tasa mucho menor que la umbralización.

Con estas reacciones, el diseño de circuitos lógicos se fundamenta en la abstracción de una neurona simple de McCulloch-Pitts (McCulloch y Pitts, 1943). La señal de activación en tales neuronas, se calcula a partir de la suma de las señales de entrada y la comparación contra un valor de umbral, de manera que si la suma es mayor que el umbral, la neurona se activa. Los niveles lógicos de las señales, corresponden a niveles de concentraciones, de manera que, al igual que en los circuitos electrónicos, se definen umbrales para estas concentraciones. Ellos definen el “0” lógico para un valor menor o igual al 10 % y el “1” lógico para un valor mayor o igual al 90 % de una concentración base dada x. Las reacciones de umbralización y amplificación resultan fundamentales para lograr esta abstracción digital.

La Figura 12 muestra el esquema de reacciones para implementar las funciones lógicas AND y OR. Para una compuerta de dos entradas, se consideran dos hebras cable de entrada con los mismos dominios de reconocimiento en la parte derecha ($w_{1,5}$ y $w_{3,5}$, en el ejemplo). Las concentraciones de estas moléculas cable, son sumadas en una primera reacción con una compuerta sube-baja que comparte el dominio de reconocimiento. La concentración de la molécula cable salida de esta reacción, se di-

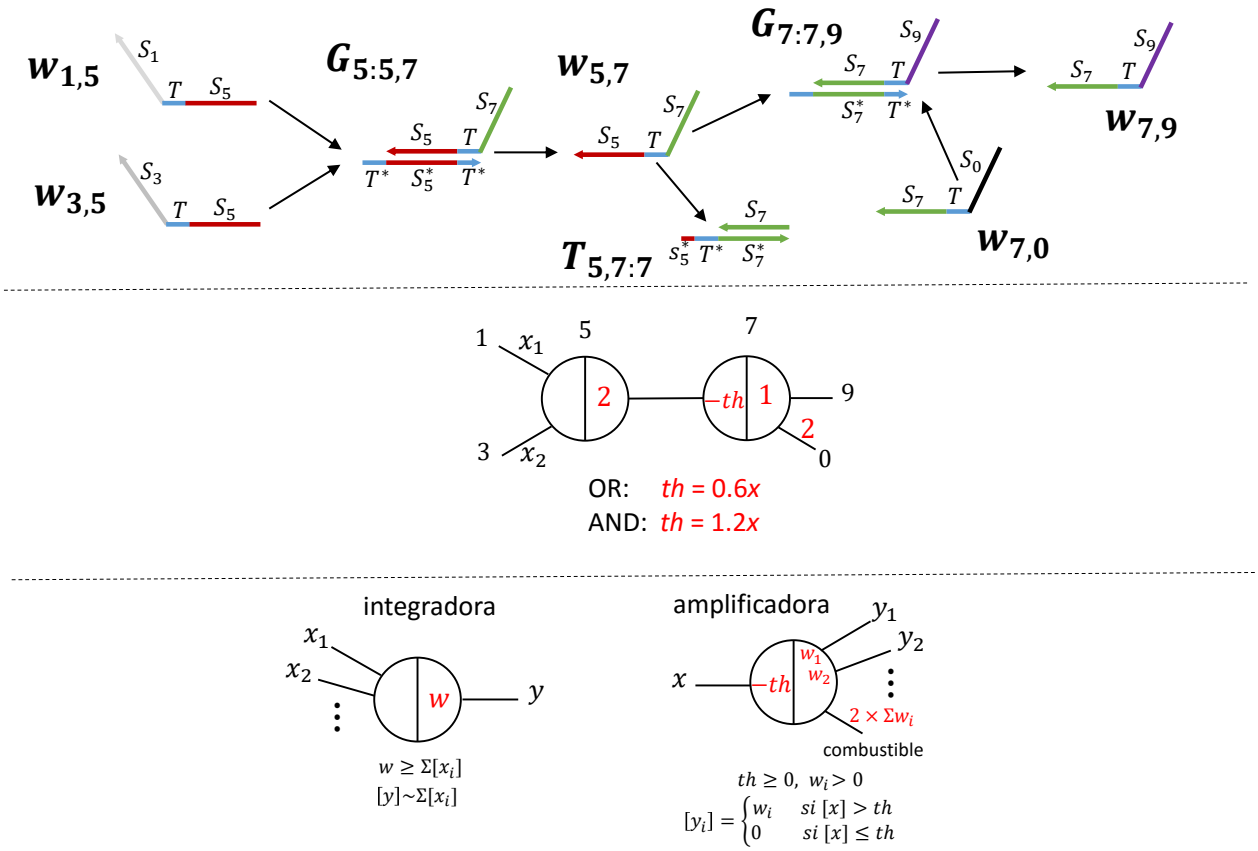

Figura 12. Implementación de funciones AND-OR en el esquema sube-baja. En la parte superior se describe la secuencia de reacciones: suma de las entradas, umbralización y amplificación. El esquema de reacciones es idéntico para ambas funciones lógicas, lo que cambia es la concentración de la molécula umbral. En la parte central se muestra el diagrama abstracto que corresponde a las reacciones del panel superior. Los dominios involucrados aparecen en color negro, las concentraciones relativas en rojo. El nodo de la izquierda representa la operación de suma y el de la derecha las operaciones de umbralización y amplificación. En la parte inferior se muestran los diagramas genéricos para realizar compuertas con múltiple entrada (*fan-in*) y múltiples salidas (*fan-out*). La compuerta integradora realiza la suma de las concentraciones de las entradas, mientras que la amplificadora genera varias salidas a partir de una misma entrada, si esta rebasa la concentración de una molécula umbral. Observe que el diagrama central corresponde a una integradora de dos entradas, conectada a una amplificadora de una salida.

vide entonces en reacciones de umbralización y amplificación, donde la mayor parte se consume en la primera, debido a su mayor tasa de reacción (los autores consideran un factor de duplicación de 20 veces). Por ello, se considera que la reacción de umbralización se realiza primero, restando efectivamente la concentración de salida. El resto de esta concentración pasa por una compuerta con suficiente combustible para ser amplificada (restauración de señal).

Para una concentración de la molécula umbral de $0.6x$, el funcionamiento corresponde a una compuerta OR (la única suma que no rebasa el umbral es $0.1x + 0.1x$ que representa la entrada $x_1 = 0; x_2 = 0$). Para el caso de la compuerta AND, la concentra-

ción de la molécula umbral debe ser de $1.2x$ (la única suma que rebasa el umbral es $0.9x + 0.9x$ que representa la entrada $x_1 = 1; x_2 = 1$). El esquema considera concentraciones adecuadas para las demás moléculas de la compuerta lógica (combustible, sube-baja sumadora, sube-baja amplificadora) de manera que se puedan implementar compuertas de múltiples entradas (*fan-in*) y múltiples salidas (*fan-out*). Este modelo de implementación de circuitos lógicos permite la realización de cómputo universal con compuertas AND-OR-NOT, utilizando la representación de doble carril, como se revisará en el Capítulo 5

Capítulo 2. Marco teórico

En esta sección se revisan algunos conceptos y nociones generales que son utilizados en el desarrollo del presente documento.

2.1. Biomoléculas

Como su nombre lo indica, el cómputo biomolecular se basa en arreglos y concentraciones de biomoléculas para que reaccionen de manera programada. Las principales biomoléculas son las tres conocidas macromoléculas de la célula (Waterman, 1995): ADN, ARN y proteínas.

2.1.1. ADN

El ADN o ácido desoxirribonucleico, cuya estructura química se conoce desde hace más de seis décadas (Watson y Crick, 1953), es el agente que almacena y codifica la información necesaria para la construcción de las células y, por ende, de los organismos vivos. Puede encontrarse en forma de hebras simples (*single-stranded DNA*, abreviado ssDNA) o formando complejos dobles (*double-stranded DNA*, abreviado dsDNA). Una hebra simple de ADN es un polímero formado de una secuencia de cuatro monómeros llamados bases o nucléotidos (nt) encadenadas juntas en una columna vertebral orientada, como si fueran cuentas unidas a un collar (Figura 13). Estas bases están conformadas por un grupo fosfato, una pentosa (desoxirribosa) y una base nitrogenada. Existen cuatro tipos de bases nitrogenadas que se clasifican en dos grupos: en el grupo purinas están la Adenina y la Guanina, mientras que al grupo pirimidinas pertenecen la Citosina y la Timina. Para abreviar las secuencias, se utilizan las iniciales de cada base (A, G, C, T). El grupo fosfato y el azúcar de las bases forma una columna vertebral que las une formando enlaces covalentes del grupo fosfato de una base al grupo hidroxilo de la siguiente. Estos enlaces se conocen como enlaces fosfodiéster y son los que le dan la orientación a la columna vertebral que encadena las bases en las hebras, que inician en un grupo 5’- fosfato y terminan en un grupo 3’-hidroxilo. Esta direccionalidad normalmente se representa esquemáticamente dibujando las hebras de ADN como flechas, donde la punta indica el extremo 3’. Comúnmente, en la literatura se hace referencia a una secuencia corta de ADN como oligonucleótido (o simplemente

oligo). La estructura química de las bases permite la formación eficiente de enlaces de hidrógeno entre dos pares de bases. Así, A se une con un enlace doble de hidrógeno a T, mientras que G se une mediante un enlace triple de hidrógeno a C. Esta propiedad de unión entre pares de bases se conoce como complementariedad de Watson-Crick. Dos hebras simples de ADN con orientación opuesta y con bases complementarias en cada una de sus posiciones (llamadas hebras antiparalelas) se pueden unir para formar una hebra doble de ADN, en un proceso conocido como hibridación. Este proceso también aplica en fragmentos antiparalelos de dos hebras o incluso, dentro de una misma hebra. El proceso de hibridación de hebras simples de ADN en complejos de doble hebra ha sido ampliamente estudiada y termodinámicamente caracterizada (SantaLucia, 1998; SantaLucia y Hicks, 2004; Zhang *et al.*, 2018), permitiendo que la predictibilidad de este comportamiento sea el fundamento físico de la computación basada en ADN. El proceso opuesto de separación, se conoce como desnaturalización. Estos simples procesos son la base para la realización de otros procesos más intrincados.

El ADN no solamente se presenta de forma lineal. Los plásmidos son moléculas de ADN circular extra-cromosómico que se replican y transcriben de forma independiente. Normalmente, se presentan en bacterias y en organismos eucariotas simples como las levaduras. Las moléculas de ADN plasmídico también adoptan una conformación de doble hélice, pero a diferencia del ADN cromosomal, éstas no tienen proteínas asociadas. Los plásmidos tienen un amplio uso en los modelos de cómputo con ADN, especialmente en los sistemas recombinantes.

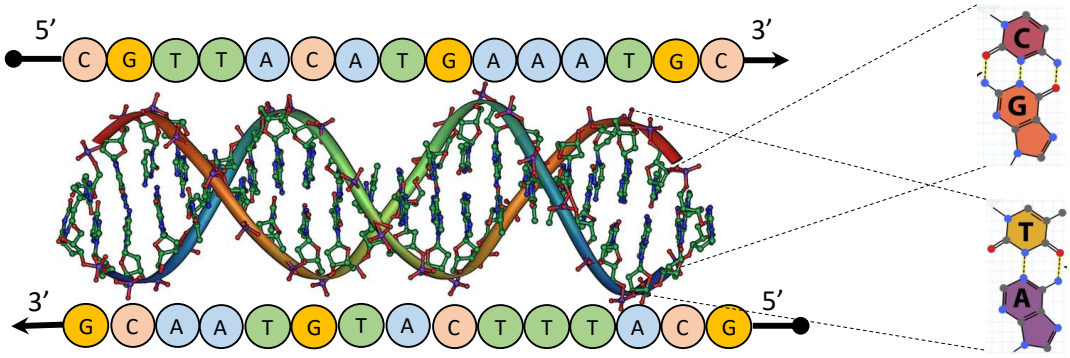

Figura 13. La estructura de la molécula de ADN y la formación de pares Watson-Crick.

2.1.2. ARN

El ácido ribonucleico o ARN es una molécula muy similar al ADN. De hecho, ambas sólo difieren en tres aspectos principales: los nucleótidos en el ARN contienen el azúcar ribosa, mientras que el ADN contiene desoxirribosa; en el ARN la base Uracilo sustituye a la base Timina del ADN; el ARN usualmente consiste en hebras simples mientras que el ADN se presenta generalmente en hebras dobles. Los principios de complementariedad de Watson-Crick se mantienen en el ARN, uniéndose la base Uracilo a la Adenina en este caso.

2.1.3. Enzimas

Las proteínas son polímeros lineales formados por unidades fundamentales llamadas aminoácidos. Existe un total de 20 tipos de aminoácidos que forman parte de las proteínas. La unión entre aminoácidos se da a través de uniones covalentes llamadas enlaces peptídicos. Una característica importante de las proteínas es la estructura tridimensional que adoptan, la cual está determinada por la secuencia de aminoácidos que las conforman. Las proteínas pueden unirse a otras moléculas a través de sus sitios de unión, los cuales son áreas superficiales que embonan con las superficies de las demás moléculas con una alta especificidad. Con esta característica, las proteínas pueden formar complejas estructuras supramoleculares que se encargan de catalizar reacciones químicas, que llevan a cabo los planes codificados en el material genético de la célula.

Las enzimas son proteínas que fungen como catalizadores en las reacciones químicas y como decodificadores de la información contenida en el ADN. Estas se utilizan en el cómputo biomolecular para poder acceder a fragmentos dentro del ADN, realizando una diversidad de funciones que afectan los enlaces fosfodiéster en la columna vertebral. Las más comunes son las endonucleasas, ligasas, exonucleasas, recombinasas y las polimerasas. Las endonucleasas, más comunmente conocidas como enzimas de restricción, reconocen ciertas regiones en el ADN y cortan la secuencia de nucleótidos para formar subconjuntos de secuencias. Estas enzimas reconocen una subsecuencia corta específica dentro de la hebra, llamada sitio de restricción, y corta los enlacen covalentes entre nucleótidos adyacentes. La ligasa crea enlaces covalentes entre los

extremos 3’ de un nucleótido y 5’ de otro, catalizando las reacciones de unión en extremos pegajosos complementarios (ver Figura 14). Las exonucleasas hidrolizan los enlaces fosfodiéster de los extremos 5’ o 3’ tanto de hebras simples como dobles y remueven los residuos uno a la vez. Las recombinasas son enzimas especiales que reconocen sitios dentro del ADN, cortan fragmentos entre estos sitios e introducen nuevos fragmentos. La operación realizada por estas enzimas es equivalente a la realizada por las endonucleasas utilizadas en conjunto con las ligasas. La polimerasa es una enzima que transcribe o replica ácidos nucleicos. Se utiliza principalmente para la replicación de ADN en el procedimiento de reacción en cadena de la polimerasa (Figura 15).

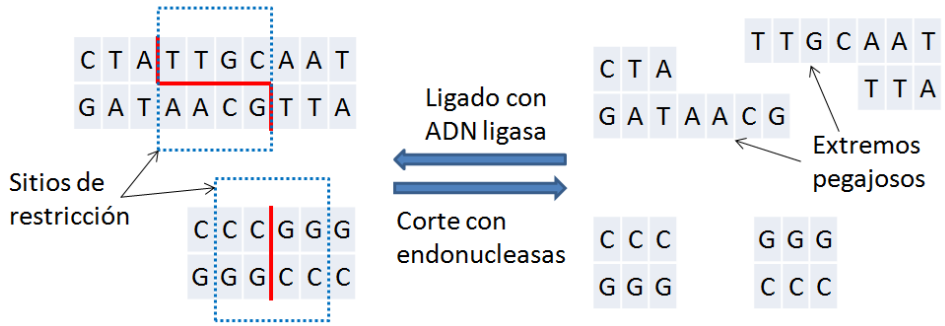

Figura 14. Efectos de las enzimas de restricción y ligasas sobre complejos dobles de ADN. Dos de estas endonucleasas hipotéticas reconocen diferentes sitios de restricción (no mostrados) y cortan la dsADN dejando extremos escalonados (pegajosos) o lisos. La ADN ligasa realiza la operación opuesta uniendo los extremos pegajosos cortados después de volver a hibridar.

2.2. Otras operaciones sobre ADN

Algunos modelos de cómputo biomolecular, particularmente los no autónomos, aplican alguna operación sobre conjuntos de hebras de ADN. Además de las reacciones enzimáticas arriba descritas, son comunes las operaciones de amplificación con la reacción en cadena de la polimerasa (PCR, por sus siglas en inglés) (Figura 15), la electroforesis en gel (Figura 16) y la purificación por afinidad (Figura 17).

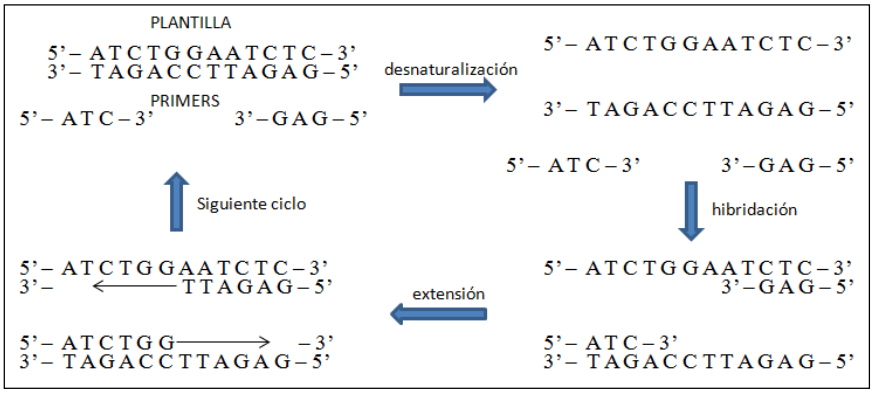

Figura 15. Un ciclo de PCR. Se aplica calor para separar la plantilla en hebras simples. Se enfría la solución para que los iniciadores *primers* hibriden en la región complementaria de la plantilla de ADN. A partir de esta unión, y teniendo libres los extremos 3' de los iniciadores, la ADN polimerasa puede extenderlos agregando nucleótidos en la dirección 5' a 3' hasta que cada hebra forme un complejo doble. El proceso se repite duplicando la cantidad de hebras en cada iteración.

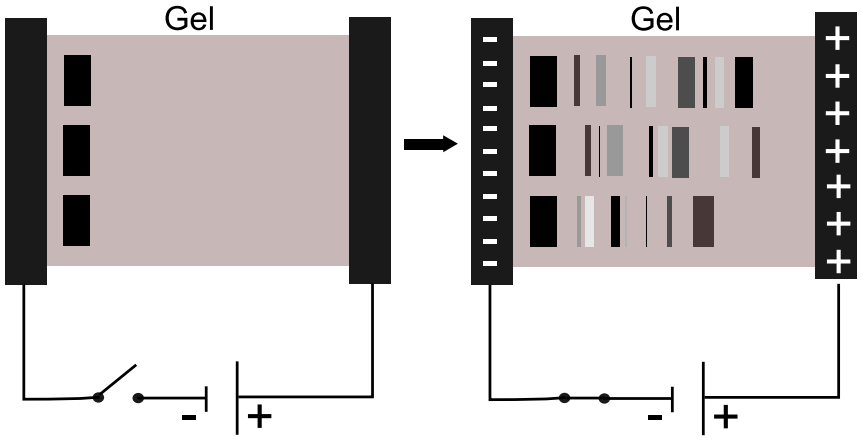

Figura 16. Electroforesis en gel. Para distinguir entre diferentes tamaños de moléculas de ADN, se colocan muestras en gel en diferentes carriles. Entonces, se aplica un campo eléctrico que mueve las moléculas de ADN a través del gel, debido a su carga eléctrica. La fuerza uniforme del campo eléctrico mueve las moléculas de menor tamaño más rápido, quedando separadas las muestras en bandas.

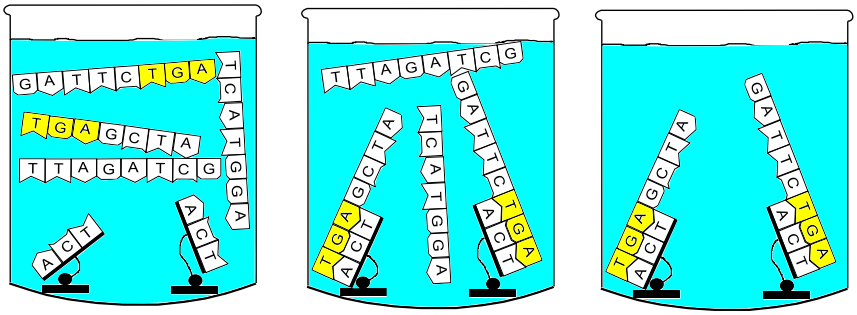

Figura 17. Proceso de purificación por afinidad. Para extraer (separar) las hebras que contienen la subsecuencia 5'- TGA - 3' se agregan las hebras a una solución que contiene sondas adheridas a un soporte sólido. Estas sondas corresponden a la subsecuencia 3'- ACT - 5' complementaria. Las hebras que contienen la subsecuencia deseada hibridan con las sondas. Se lava el contenido removiendo de la mezcla aquellas hebras que no contienen la subsecuencia deseada. En la práctica, se requieren subsecuencias más largas para que el proceso funcione.

2.3. Termodinámica del ADN

El aspecto termodinámico de los procesos fisico-químicos tiene que ver en cómo estos se ven afectados a partir de cambios energéticos (Ignatova *et al.*, 2008). En este documento estamos interesados en el proceso de hibidación de ADN. En este sentido, un concepto importante en la termodinámica del ADN es la temperatura de fusión (*melting*) T_m, la cual es la temperatura en la que la mitad de las hebras en solución están hibridadas. Otros conceptos termodinámicos de amplio uso son la energía libre de Gibbs ($\Delta G°$), que describe el potencial de una reacción de ocurrir de manera espontánea; la entalpía ($\Delta H°$), que proporciona la cantidad de calor liberado o absorbido por el sistema y la entropía ($\Delta S°$) que da una medida el desorden o la aleatoriedad del sistema. Estas medidas, junto con la temperatura T del sistema, están relacionadas en la conocida fórmula

$$\Delta G° = \Delta H° - T\Delta S°. \tag{1}$$

En efecto, el proceso de hibridación de hebras de ADN sencillas para la formación de dúplex está estrechamente ligado a la temperatura del medio. Los modelos de predicción de la llamada estructura secundaria de ADN — a saber, el emparejamiento de las bases — se basan en las interacciones entre pares de vecinos. El modelo del vecino más cercano de SantaLucia y Hicks (2004), ofrece una forma de calcular la estabilidad relativa de las moléculas de ADN en dúplex. Para los diez posibles pares

de vecinos más cercanos, este modelo contiene tabulados valores de energía libre de Gibbs, entropia y entalpía (Tabla 1).

Tabla 1. Parámetros tabulados del modelo del vecino más cercano de SantaLucia y Hicks (2004). Calculados usando un búfer con 1 M de NaCl a 37° C. La corrección de simetría se aplica solo a dúplex autocomplementarios. La penalización por terminación en AT aplica cuando un dúplex termina en tal par. Las unidades para $\Delta G°$ y $\Delta H°$ están dadas en kcal/mol, mientras que para $\Delta S°$ son cal/K por mol de interacción.

Interacción	$\Delta H°$	$\Delta S°$	$\Delta G°$
AA/TT	−7.6	−21.3	−1.00
AT/TA	−7.2	−20.4	−0.88
TA/AT	−7.2	−21.3	−0.58
CA/GT	−8.5	−22.7	−1.45
GT/CA	−8.4	−22.4	−1.44
CT/GA	−7.8	−21.0	−1.28
GA/CT	−8.2	−22.2	−1.30
CG/GC	−10.6	−27.2	−2.17
GC/CG	−9.8	−24.4	−2.24
GG/CC	−8.0	−19.9	−1.84
Iniciación	+0.2	−5.7	+1.96
Penalización terminación AT	+2.2	+6.9	+0.05
Corrección de simetría	0.0	−1.4	+0.43

A partir de la Tabla 1, se puede calcular la energía libre de Gibbs de un dúplex. Por ejemplo, considerando el dúplex de la secuencia $x = 5' - CGTTGA - 3'$ con su complemento $x^* = 3' - GCAACT - 5'$, $\Delta G°$ (a 37°) está dada por:

$$\Delta G°(x) = \Delta g_i + \Delta g_s + \Delta g_t + \Delta G°(\text{pares}), \tag{2}$$

donde Δg_i es la energía de iniciación de la hélice (+1.96), Δg_s es la corrección por simetría (0.0), Δg_t es la penalización por terminación en AT (+0.05) y $\Delta G°(\text{pares})$ es la suma de las $\Delta G°$ para los pares CG/GC, GT/CA, TT/AA, TG/AC, y GA/CT. Sustituyendo todos los valores se obtiene $\Delta G°(x) = -5.35$ kcal/mol.

2.4. Cinética de la hibridación de ADN

La cinética de la hibridación de ADN describe la reacción química reversible de la hibridación entre dos hebras complementarias de ADN para formar un complejo doble. El emparejamiento de oligonucleótidos simples de ADN se describe de manera

esquemática en la Figura 18. En la reacción de hibridación, los oligonucleótidos complementarios S y S^* que tienen las secuencias a y a^* se combinan para formar el dúplex D. Las constantes de reacción hacia delante y hacia atrás son k_{hyb} y k_{Dis}, respectivamente. Cuando la reacción alcanza el equilibrio, las constantes son las mismas, de forma que las concentraciones ya no cambian en el tiempo.

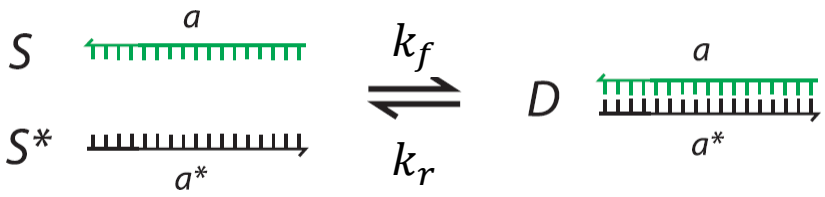

Figura 18. Hibridación y disociación de ADN.

La constante de reacción hacia adelante depende de la longitud del ADN, la secuencia y la concentración de sales:

$$k_f = \frac{k'_N \sqrt{L_s}}{N}, \tag{3}$$

donde L_s es la longitud de la hebra más corta que participa en la formación del dúplex; N es el número total de pares de bases presentes; y K'_n es la constante de nucleación, estimada en $(4.35 \log_{10}[Na^+] + 3.5)$ para $0.2 \leq [Na^+] \leq 4.0$ mol/l. La constante hacia atrás k_r es muy sensible a la longitud y secuencia del ADN:

$$k_r = k_f e^{\Delta G^\circ / RT}, \tag{4}$$

donde R es la constante de los gases y T es la temperatura de incubación. La hibridación *in vitro* normalmente se lleva a cabo a temperatura $T = T_m - 298.15$ K, donde T_m es la temperatura de fusión.

2.5. Desplazamiento de hebras de ADN

Como en la mayoría de los modelos de cómputo biomolecular, en la técnica de desplazamiento de hebras se hace la abstracción de dominios lógicos. Esto es, las hebras de ADN se dividen en segmentos que se etiquetan para identificar unívocamente enti-

dades de información (variables o señales). Se utilizan dominios largos (de 15 a 25 nt) como variables de reconocimiento; mientras que dominios cortos llamados puntos de apoyo (*toeholds*) que usualmente tienen de 4 a 8 nt de longitud, fungen como iniciadores de reacciones. Por convención, se usan letras mayúsculas para denotar dominios largos, minúsculas para toeholds y un dominio D^* expresa el dominio con secuencia complementaria a D.

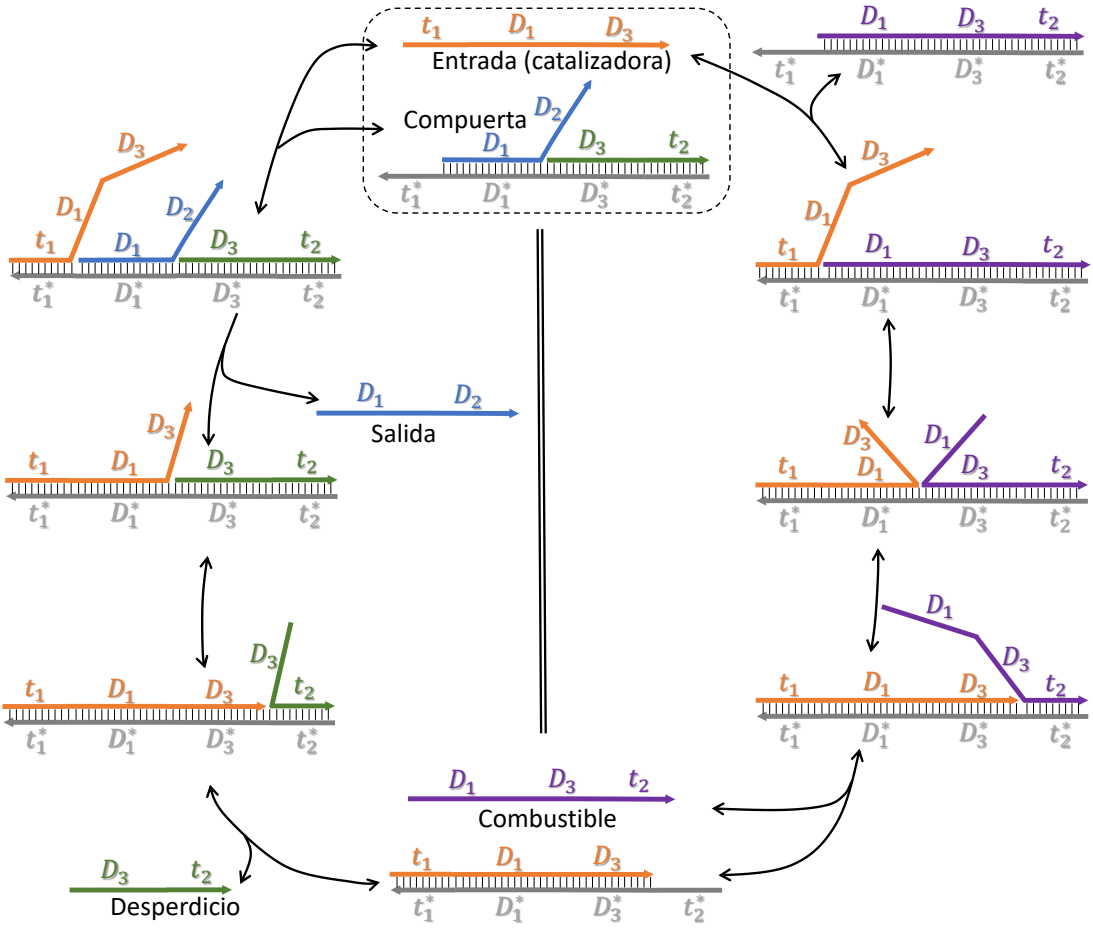

Figura 19. Ejemplificación del mecanismo de intercambio de toeholds. La hebra invasora (catalizadora) se une al complejo compuerta a través de los toeholds complementarios t_1. Enseguida, tanto la invasora como la adjunta se involucran en la reacción competitiva de manera similar al juego "jalar la cuerda" donde el punto de migración va y viene. La hebra invasora toma todos los pares de bases del dominio de migración, liberando la primera hebra (salida). La débil hibridación de la hebra desperdicio a través del toehold t_2 origina su desprendimiento de la hebra sustrato. La molécula combustible puede entonces unirse al toehold recién expuesto disparando un proceso de migración de ramas en reversa que finalmente libera la hebra catalizadora. Estos ciclos catalíticos continúan hasta que las moléculas combustible se agoten en las reacciones.

En una reacción de desplazamiento de hebras de ADN (DSD, por sus siglas en inglés), una hebra invasora remueve gradualmente a otra que estaba previamente adjunta a una hebra sustrato, uniéndose en su lugar con esta última. El proceso inicia con

la co-localización (i.e., colisión) de ambas moléculas y la subsecuente unión a través de sus toeholds complementarios. Una vez hibridados completamente los toeholds, se inicia un proceso llamado migración de ramas de 3 vías. Este paso, similar a una caminata aleatoria, es el fundamental de la reacción de desplazamiento. Consiste en el avance (y retroceso) del punto de hibridación en el que la hebra invasora “le gana” una base a la previamente adjunta (o esta “lo recupera”). Al final, el desplazamiento de la hebra adjunta es impulsado por la ganancia en energía obtenida con el evento de nucleación (toeholds). Un punto crucial en estas reacciones es que la cinética resultante de la reacción general se puede controlar hasta por un factor de 10^6, con diferentes variantes de longitud y composición de los toeholds involucrados (Zhang y Winfree, 2009). En general, mientras más largos sean estos toeholds, mayor será la rapidez cinética, limitado a las condiciones de saturación dadas por toeholds de más de 7 nt. En una variante de reacción DSD conocida como reacción de intercambio de toeholds (Figura 19), el dominio de migración de ramas se extiende unos cuantos nucleótidos solamente para el complejo adjunta-sustrato, en el otro extremo del toehold inicial. Como consecuencia, la hebra adjunta no se libera inmediatamente al final de la migración de ramas, sino que tiene que disociarse del sustrato, revelando un nuevo toehold en el otro extremo y volviendo efectivamente la reacción reversible. El ciclo se completa con una hebra extra denominada combustible que reconoce el toehold activado, actuando como una nueva hebra invasora en sentido inverso. Esta función de intercambio de toeholds ha sido también caracterizada y modelada como un proceso de tres pasos (Zhang y Winfree, 2009), y dicho conocimiento se ha aplicado exitosamente en el diseño de redes de reacciones catalíticas en las que las moléculas de entrada no son consumidas, permitiéndoles participar en múltiples reacciones.

2.6. Inferencia lógica

La tarea de la lógica consiste en clasificar argumentos como válidos o inválidos (Hurley, 2000). Un argumento es un razonamiento que se compone de una colección de premisas y una conclusión. Se dice que un argumento es válido si la conclusión se *deriva lógicamente* de las premisas. Por ejemplo, el siguiente silogismo — conocido universalmente — es un razonamiento válido:

(I) Todos los hombres son mortales.

(II) Sócrates es un hombre.

(III) Por lo tanto, Sócrates es mortal.

Las sentencias (I) y (II) corresponden a las premisas mientras que (III) es la conclusión. El proceso de derivación de la conclusión a partir de las premisas se conoce como inferencia lógica. En términos formales, el problema de la inferencia lógica se puede describir de manera general de la siguiente forma. Dadas una base de conocimientos (BC) compuesta de una colección de premisas $P_1, P_2, \cdots, P_n$ y una consulta Q; se debe demostrar que Q es una consecuencia lógica de la BC. En otras palabras, probar que Q es VERDADERA en cada caso que TODAS las premisas también lo son.

En la lógica más simple, denominada de orden cero o proposicional, las premisas se forman con un conjunto de proposiciones simples, las cuales son sentencias que pueden tomar valores de verdad (verdadero o falso). Estas proposiciones se pueden unir a través de conectivos lógicos para formar proposiones compuestas, que se pueden definir inductivamente de la siguiente forma. Si p y q son dos proposiciones simples o compuestas cualquiera:

- $\neg p$ denota la la negación de p.
- $p \wedge q$ denota la la conjunción, que será verdadera cuando ambas proposiciones lo sean.
- $p \vee q$ denota la disyunción, que será verdadera cuando alguna de las dos proposiciones lo sea.
- $p \rightarrow q$ denota la implicación, que será verdadera cuando p sea falsa o q sea verdadera (ver Tabla 2).
- $p \leftrightarrow q$ denota la doble implicación, que será verdadera cuando ambas proposiciones tengan el mismo valor de verdad (ver Tabla 3).

La lógica proposicional tiene una ontología limitada para la representación y resolución de problemas, pues interpreta cualquier universo de discurso solamente en

Tabla 2. Tabla de verdad para la implicación lógica.

p	q	$p \rightarrow q$
F	F	V
F	V	V
V	F	F
V	V	V

Tabla 3. Tabla de verdad para la doble implicación lógica.

p	q	$p \leftrightarrow q$
F	F	V
F	V	F
V	F	F
V	V	V

términos de hechos (Russell y Norvig, 2009). Esto la vuelve inadecuada incluso para casos tan simples como el ejemplo revisado anteriormente. En cuanto a poder expresivo se refiere, el siguiente nivel es la lógica de primer orden (también llamada cálculo de predicados). El poder expresivo superior de la lógica de primer orden, se obtiene a través de la representación del universo como un conjunto de **objetos** con sus **propiedades**, **relaciones** y **funciones**. Las premisas son expresiones que se construyen así con diferentes conjuntos de símbolos entre los que se incluyen (Lloyd, 1984): (1) constantes, para expresar objetos fijos; (2) variables, para referirse a cualquier objeto; (3) relaciones (o predicados); (4) funciones; (5) conectivos lógicos ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$); (6) cuantificadores, universal ($\forall$) y existencial ($\exists$); y (7) símbolos de puntuación.

La programación lógica es un paradigma de cómputo — orientado principalmente a la lógica de primer orden — donde la base de conocimientos se expresa en forma de hechos y reglas (Sterling y Shapiro, 1994). Los hechos son predicados que expresan relaciones entre — o propiedades de — los objetos. Tienen la forma general *Predicado*$(\text{obj}_1, \text{obj}_2, \cdots, \text{obj}_k)$, donde k representa la aridad del predicado. Por ejemplo, el predicado unario[1] *Filósofo*(Sócrates) establece que el objeto “Sócrates” pertenece a la clase *Filósofo*; o el predicado binario *Maestro*(Sócrates, Platón) establece una relación “Maestro de” entre los objetos “Sócrates” y “Platón”. Las reglas, por su parte, son implicaciones que expresan conocimiento sobre ciertos hechos, de manera que nuevos hechos pueden ser obtenidos a partir de hechos dados. Considerando una base de conocimientos como el medio para almacenar información acerca de objetos (es

[1] Los predicados unarios pueden verse más como propiedades o clases de los objetos, en lugar de relaciones

decir, lo que sabemos acerca de ellos), es intuitivo el realizar ciertas consultas dentro de este dominio de discurso. A través de una consulta como ¿*Griego*(Platón)?, por ejemplo, podemos saber si Platón pertenece a la clase *Griego*. Adicionalmente, el lenguaje de primer orden incluye el uso de — objetos — variables, con el propósito de hacer preguntas generales. Por ejemplo, las consultas ¿*Griego*(X)? y ¿*Hermanos*(X, Y)? se interpretan como las preguntas ¿“Quién es *Griego*?” y “¿Quiénes son *Hermanos*?”, respectivamente.

2.6.1. Procedimientos de inferencia

Para bases de conocimiento en forma de hechos y reglas, existen principalmente dos estrategias para el proceso de inferencia:

- Encadenamiento hacia atrás (también llamada inferencia en retroceso o de arriba hacia abajo). Aquí, se parte de una consulta de la que se van obteniendo nuevas consultas, hasta que una de ellas se responde con su hecho respectivo (inferencia positiva) o hasta que ya no se puedan obtener nuevas consultas (inferencia negativa).
- Encadenamiento hacia adelante (también conocido como inferencia de abajo hacia arriba). En este proceso, se parte de hechos iniciales para ir generando nuevos hechos. El proceso termina con una inferencia positiva cuando se produce el hecho que responde la consulta, o con inferencia negativa cuando ningún hecho corresponde a esta consulta.

La Figura 20 muestra dichos procesos para una BC en forma proposicional. Cada uno de ellos se puede descomponer en dos operaciones básicas:

- Para ambos procesos aplica la operación **consulta-proposición**, cuando una consulta se puede responder con su hecho correspondiente. Esto produce una inferencia positiva que puede ser parcial (para una ruta de inferencia) o total (para el programa lógico). Por ejemplo, una consulta ¿q? se puede resolver con la proposición q.
- Encadenamiento hacia atrás se basa en la operación **consulta-implicación** en la que una consulta se aplica sobre reglas coincidentes (aquellas que comparten

la misma proposición en la parte de la conclusión). Esto produce las premisas de cada regla como nuevas consultas. Por ejemplo, una consulta q sobre las reglas $p \rightarrow q$ y $r \wedge s \rightarrow q$, arrojaría las nuevas consultas p? y $(r \wedge s)$?, respectivamente.

- Encadenamiento hacia adelante se basa en la operación **proposición-implicación** que se lleva a cabo cuando las proposiciones se aplican sobre reglas con premisas coincidentes para derivar consecuencias lógicas en forma de nuevos hechos. Por ejemplo, en una BC con hechos r y s, se pueden disparar las reglas $r \rightarrow q$, $s \rightarrow p$, y $r \wedge s \rightarrow n$, generando los nuevos hechos q, p, y n, respectivamente.

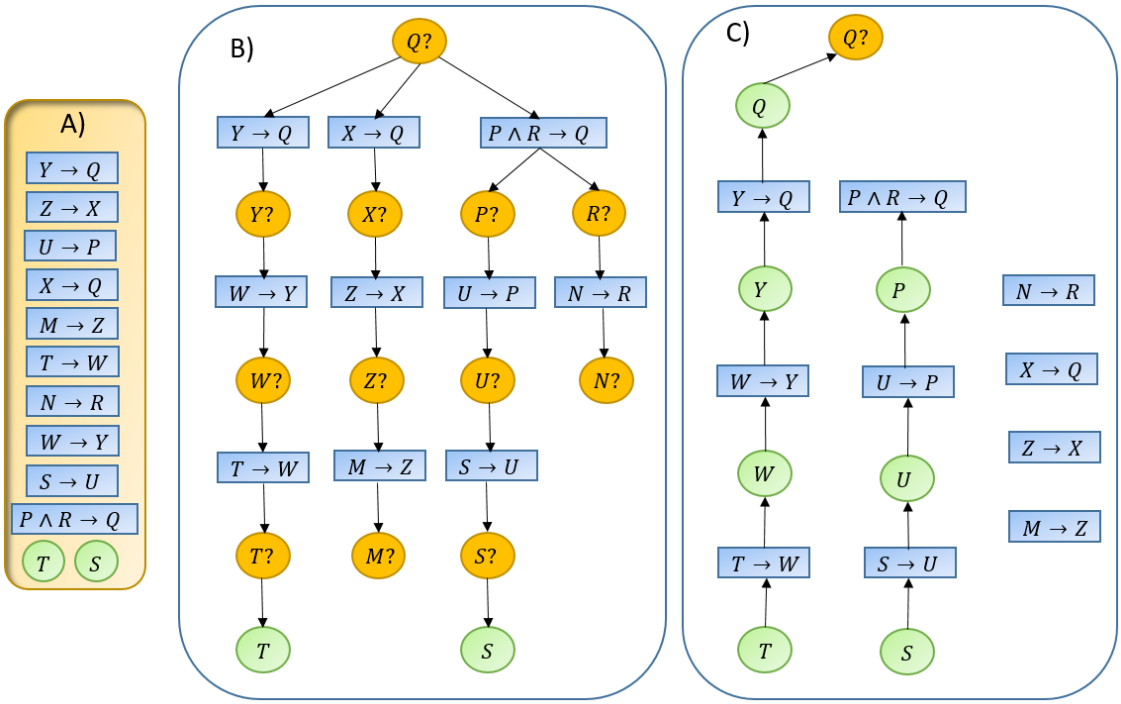

Figura 20. Procedimientos de inferencia. A) Ejemplo de base de conocimientos en lógica proposicional. B) Proceso de encadenamiento hacia atrás (inferencia arriba-hacia-abajo). C) Proceso de encadenamiento hacia adelante (inferencia abajo-hacia-arriba).

2.7. Herramientas de simulación

Los resultados obtenidos en el presente documento fueron obtenidos de algunas herramientas de simulación termodinámica y cinética química. La simulación termodinámica se enfoca en la estructura secundaria de los complejos (es decir, la formación de pares de bases) involucrados en un esquema y las diferentes energías que un complejo particular adopta al variar las secuencias de sus hebras constituyentes, de

acuerdo a un modelo de energía definido. Por su parte, la simulación cinética tiene que ver con la enumeración de las diferentes reacciones posibles que un sistema puede producir y con los parámetros cinéticos para la evolución en el tiempo de los diferentes reactivos y productos. Las principales características de las herramientas utilizadas serán brevemente descritas en lo siguiente.

2.7.1. Visual DSD

Visual DSD es un enumerador/simulador cinético implementado por Lakin *et al.* (2011, 2012) como una interfaz de programación del lenguaje DSD (Phillips y Cardelli, 2009). Los programas en DSD se compilan en reacciones siguiendo cuatro semánticas que proporcionan diferentes niveles de detalle en los modelos generados. Después de la compilación, el sistema de reacciones puede ser simulado con diferentes parámetros cinéticos configurables a través del código. El software permite tres modos de simulación: estocástica, que ejecuta el algoritmo de Gillespie (1977); determinista y una ejecución especial denominada justo-en-tiempo para sistemas grandes. A la fecha, este es el único simulador cinético que trabaja a nivel de dominios para sistemas de desplazamiento de hebras (Grun *et al.*, 2015) que ha sido ampliamente utilizado como herramienta de apoyo en varios dispositivos del estado del arte (Qian y Winfree, 2011; Qian *et al.*, 2011b; Chen *et al.*, 2013; Amir *et al.*, 2014; Chatterjee *et al.*, 2017).

2.7.2. NUPACK

NUPACK (Dirks *et al.*, 2007; Zadeh *et al.*, 2010) es un paquete de análisis termodinámico para ácidos nucleicos que viene integrado de varias herramientas, entre las que destacan la función de partición y la estructura secundaria de energía libre mínima (MFE, por sus siglas en inglés). Para una entrada consistente en una colección de hebras $s_1, s_2, \ldots, s_n$ en solución, con sus correspondientes concentraciones $c_1, c_2, \ldots, c_n$, NUPACK puede calcular las estructuras secundarias y las concentraciones finales de cada complejo en equilibrio, dadas las condiciones de temperatura y concentración de sales del búfer. Así, esta herramienta resulta útil para obtener indicios computacionales de la correcta formación de estructuras deseadas, en diseños de dispositivos basados en ácidos nucleicos.

2.7.3. MULTISTRAND

Desarrollada por Schaeffer (2013), esta herramienta permite tanto análisis termodinámico (de acuerdo al mismo modelo de Dirks *et al.* (2007) como cinético. La máquina de simulación ejecuta una versión de algoritmo de Gillespie donde la cinética de un conjunto de hebras interactuando se modela como un proceso de Markov en tiempo continuo a través del espacio de estados de todas las estructuras secundarias. Las transiciones de estado a estado en las trayectorias simuladas consideran cambios de un simple par de bases, de manera que esta es la única herramienta que trabaja con una granularidad a nivel de secuencia (Grun *et al.*, 2015). La manera de explorar la cinética de sistemas de ADN, consiste en realizar múltiples trayectorias para obtener resultados estadísticos. Esta información estadística se usa entonces para ajustar los parámetros cinéticos de un modelo químico masa-acción.

2.7.4. Simuladores cinéticos masa-acción

En este tipo de simuladores se introduce manualmente el modelo químico enumerando todas sus reacciones y correspondientes constantes. Se pueden escoger distintos solucionadores tanto estocásticos como deterministas para obtener un análisis del comportamiento cinético. El software DIZZY (Ramsey *et al.*, 2005) proporciona una herramienta para el modelado estocástico y determinista de la cinética de redes genéticas, metabólicas y de señalización, a gran escala. Otro paquete utilizado fue CAIN[2], que agrega mayor funcionalidad con respecto a Dizzy, permitiendo, por ejemplo, la simulación de varias copias del mismo circuito y ejecución paralela de varias trayectorias simultáneas.

[2]Software de licencia libre disponible en: http://cain.sourceforge.net/

Capítulo 3. Metodología

Los métodos empleados en el desarrollo de la presente investigación se dividen en dos, de acuerdo a ambos objetivos planteados. Aunque son similares, ambos métodos se detallan en dos secciones por separado.

3.1. Un esquema de inferencia lógica con cascadas de desplazamiento de hebras de ADN

En esta sección, se presenta la metodología empleada en la implementación de un nuevo esquema autónomo de inferencia lógica, en el que las rutas de inferencia, ya sean del algoritmo de encadenamiento hacia adelante o hacia atrás, se producen a través de reacciones de desplazamiento de hebras. Además de presentar una nueva codificación para abordar programación lógica biomolecular, lo novedoso de este enfoque consiste en aprovechar las reacciones de intercambio de *toeholds* para introducir comportamiento catalítico.

3.1.1. Metodología general

Las etapas de la metodología empleada fueron las siguientes.

- **Estudio del estado del arte.** El problema abordado es, básicamente, un problema de diseño en el que se requiere entender a profundidad el estado del arte para identificar metodologías, herramientas y resultados obtenidos, así como posibles brechas en la investigación. Los trabajos previos identificados en tal revisión fueron incluidos en la introducción (Sección 1.5), describiendo los mecanismos empleados y los problemas de inferencia lógica abordados.
- **Estudio de herramientas de simulación.** De la revisión en el paso anterior, se seleccionó el mecanismo de desplazamiento de hebras de ADN, como la primitiva de cómputo. En base a esta selección, se estudió la literatura relacionada, particularmente el lenguaje DSD de Phillips y Cardelli (2009) y su implementación en el simulador Visual DSD (Lakin *et al.*, 2011). Para la validación termodinámica se revisó el funcionamiento de NUPACK (Zadeh *et al.*, 2010) y de este mismo

paquete integrado en la herramienta MULTISTRAND (Schaeffer, 2013; Schaeffer *et al.*, 2015). Esta última, que integra un simulador cinético que ejecuta trayectorias a nivel de nucleótidos, también fue estudiada. Finalmente, se revisaron herramientas de simulación de cinética masa-acción, seleccionando entre estas los paquetes DIZZY (Ramsey *et al.*, 2005) y CAIN.[1]

- **Diseño de compuertas con desplazamiento de hebras.** Esta etapa consistió en un proceso iterativo de prueba y error en la que los diseños planteados se simularon a nivel de dominios lógicos en la herramienta Visual DSD, para obtener indicios de su funcionamiento correcto y desempeño. Los casos de prueba y la configuración experimental se detallan abajo (Sección 3.1.2).

- **Validación termodinámica.** Una vez obtenido un diseño final, se procedió a su validación termodinámica. Para ello se escribió código utilizando la interfaz de Python de MULTISTRAND. Este programa procesaba iterativamente secuencias generadas aleatoriamente para medir su aptitud en base a funciones NUPACK. Tal procedimiento de búsqueda arrojó conjuntos de secuencias candidatas que fueron probadas en la herramienta en línea NUPACK que analiza la hibridación de hebras en contexto de tubo de ensayo. El objetivo fue obtener un conjunto de secuencias que garantizaran la correcta formación de las moléculas en los casos de prueba analizados. Este procedimiento junto con parámetros de configuración se revisan en la Sección 3.1.3.

- **Estudio de propensión de reacciones de interferencia.** En esta parte, se realizaron simulaciones de trayectorias a nivel de nucleótidos en la herramienta MULTISTRAND. Estas simulaciones incluyeron dos reacciones de interferencia identificadas en el diseño obtenido. En la Sección 3.1.4 se revisan los detalles.

- **Simulaciones masa-acción.** Por último, el modelo que incluía las reacciones de interferencia se simuló junto con las reacciones diseñadas para estudiar su impacto en un contexto químico masa-acción. La configuración para estas simulaciones se revisa en la Sección 3.1.5.

[1] Software sin literatura relacionada de licencia libre. Disponible en: http://cain.sourceforge.net/

3.1.2. Simulación en Visual DSD

3.1.2.1. Casos de prueba

En la simulación con Visual DSD se consideraron tres casos de prueba. Dos de ellos (Ejemplos E1 y E2) fueron tomados del estado del arte (Ran *et al.*, 2009) y el tercero (Ejemplo E3) es un caso especial diseñado para demostrar las ventajas del funcionamiento catalítico en un escenario crítico (ver Figura 21).

Ejemplo E1: consultas existenciales. Resolver consultas existenciales consiste en determinar si existen objetos en la BC que satisfagan algún criterio específico. En nuestro ejemplo particular, deseamos determinar si existe una persona X dentro del dominio {Sócrates, Alejandro, Hércules} que posea un atributo adecuado (sabio o fuerte) para asistir ya sea a la academia o al ejército. En este sentido, asignamos a Sócrates el atributo de sabio, a Hércules el de fuerte y a Alejandro ambos. La consulta inicial ¿*Academia*(X)? debe resover cuáles individuos tienen la cualidad para asistir a la academia.

Ejemplo E2: inferencia hacia adelante con reglas compuestas. Este caso se enfoca en analizar el desempeño del mecanismo de encadenamiento hacia adelante e incluyendo reglas con conjunción en el antecedente. El programa consiste en una regla general que establece dos condiciones para que una persona sea feliz (estar enamorado y ser acaudalado). Esta implicación se puede ser establecida como *Enamorado*(X)∧ *Acaudalado*(X)→*Feliz*(X). Para un programa que produzca una inferencia positiva, supongamos que un individuo (Maslow) cumple con estas condiciones requeridas. Así, la consulta inicial ¿*Feliz*(Maslow)? debe producir una respuesta positiva.

Ejemplo E3: caso crítico. Este ejemplo representa un escenario crítico, donde la ruta de inferencia positiva se ve afectada por varias rutas negativas. La consulta inicial ¿*p*? tiene tres implicaciones coincidentes donde sólo una de ellas forma parte de la ruta de inferencia positiva. Esta misma situación ocurre para las consultas intermedias ¿*q*? y ¿*r*?. De esta forma, se espera que en la implementación molecular solo una

fracción de la consulta terminal ¿s? será liberada y respondida con su correspondiente proposición s.

Figura 21. Los tres programas lógicos utilizados como casos de prueba.

3.1.2.2. Configuración experimental

Los códigos de los tres programas implementados en Visual DSD se incluyen en el Anexo A. Todos los programas se compilaron con la semántica *detallada*, incluyendo reacciones no productivas y simulados en el modo estocástico. Se compararon dos variantes de cada programa, la catalítica (CAT) y la no catalítica (NCAT) que difieren entre sí solamente por la inclusión de moléculas combustible para la primera. Las concentraciones iniciales se establecieron conforme a una concentración base $1x = 20$nM para las implicaciones, hechos y consultas. En el caso de las moléculas combustible, estas fueron agregadas a una concentración de $2x$ para cada implicación correspondiente. En estas simulaciones, cada nM de concentración equivale a 1000 moléculas, es decir, el parámetro de factor de escala s fue establecido en 1000. De esta forma, los términos concentración y población de moléculas se usarán sin distinción. Los resultados que se presentan corresponden a los promedios de 30 simulaciones independientes utilizando 1000 puntos de datos.

Los valores de las constantes de reacciones se tomaron en base a los resultados de Zhang y Winfree (2009) como se indica a continuación. Una constante de unión $k_f = 3.5 \times 10^{-3}$ nM^{-1} s^{-1} para ambos toeholds t_1 y t_2. La constante de desunión fue calculada para cada toehold promediando las energías de unión producidas por 10000 muestras de secuencias generadas aleatoriamente (configurando la función de cálculo de energía de NUPACK a 25°C, 1.0 M Na^+, and 11.5 mM Mg^{2+}). Sustituyendo

tales energías de unión en (5) (Zhang y Winfree, 2009)

$$k_{r(\delta)} = k_f \times 2/x \times e^{\Delta G^\circ(\delta)/RT}, \tag{5}$$

se obtuvieron los valores $k_{r(t_1)} = 1.92$ y $k_{r(t_2)} = 1.35$ s^{-1}. Finalmente, para la constante de migración de ramas (paso elemental de 1 nt) se utilizó el valor de 400 s^{-1}, sugerido en el mencionado trabajo.

3.1.3. Validación termodinámica con NUPACK

En esta fase del estudio, se buscaron secuencias adecuadas que pudieran garantizar la formación de estructuras termodinámicamente estables, haciendo particular énfasis en las moléculas implicación. El motivo de esto es que, en el modelo, tales moléculas poseen una estructura que podría considerarse compleja, dado que están constituidas por al menos cuatro hebras. Estas deben hibridar correctamente en los sitios deseados, de otra forma el sistema no operaría de la forma diseñada. Lograr este objetivo, sin embargo, se complica de cierta forma, debido al uso de segmentos de migración de ramas de la forma [X—GRC—X] en lugar de dominios simples. La repetición del dominio representativo (X) en tales segmentos conlleva un riesgo inherente de hibridación espuria de las hebras cobertura izquierda y derecha de la implicación. Además, el hacer uso de hebras más largas incrementa las posibilidades de formación de estructuras secundarias dentro de ellas, las cuales deben ser minimizadas para un ensamblaje correcto.

Con este fin, se desarrolló un procedimiento simple para demostrar la viabilidad termodinámica del modelo, mediante la búsqueda de secuencias adecuadas para los dominios. El procedimiento de búsqueda se basó en las herramientas de NUPACK (Zadeh *et al.*, 2010) que vienen incluidas como paquete dentro del simulador MULTISTRAND (Schaeffer *et al.*, 2015). Estas funciones comprenden: (1) la energía libre mínima (MFE) y la(s) estructuras(s) secundaria(s) que alcanzan esta energía; (2) la energía libre de Gibbs ΔG°(S) de una estructura objetivo dada S; (3) el *defecto*(S) de ensamblaje, definido como el número de pares de bases incorrectos comparado con una estructura objetivo dada S; y (4) la *probabilidad* de equilibrio para una cierta estructura objetivo

S (Dirks *et al.*, 2007). Basado en estas funciones, se implementó el procedimiento de búsqueda en un script usando la interfaz Python de MULTISTRAND.

3.1.3.1. Configuración de parámetros

Los parámetros utilizados en el algoritmo de búsqueda son el máximo número de secuencias para iterar (MS), la distancia de Hamming mínima entre secuencias dentro de una muestra (MHD), la máxima longitud aceptada para la subsecuencia común (MCS) entre dominios y el contenido de C's al crear nuevas secuencias (dado como un intervalo CC). La secuencia para el dominio genérico no se incluye, sino que se utiliza una secuencia constante expresamente diseñada manualmente. Tal secuencia contiene un mayor contenido contenido de C's (60 %), diseminado a lo largo en tripletas "CCC". La razón detrás de esta elección fue asegurar una fuerte unión entre las hebras de salida con su respectivos sustratos.

Después de realizar pruebas preliminares (cuyos resultados no se incluyeron en este documento), la configuración de estos parámetros quedó de la siguiente manera. Para la generación de secuencias se utilizó MHD=75 % y MCS=4nt, mientras que el valor de CC se seleccionó aleatoriamente en el intervalo [30, 40] (%) para cada secuencia generada. El número de muestras MS se estableció en 10000. Adicionalmente, se verificó la influencia de diferentes longitudes de dominios haciendo ensayos con tres combinaciones de pares (20,20), (20,14) y (25,15). Cada combinación representa la longitud de los dominios genérico GRC y representativo, respectivamente. Como criterio de evaluación de las muestras, se consideraron tres funciones NUPACK: (1) minimizar $\Delta G°(\text{MFE}) - \Delta G°(\text{str})$; maximizar la probabilidad de equilibrio; y (3) minimizar el defecto de ensamblaje comparado con una implicación perfectamente ensamblada. Las condiciones experimentales del búfer para estas simulaciones fueron establecidas en 37°C, 1.0 M Na^+ y 11.5 mM Mg^{2+}.

En total, las tres opciones de longitud de dominios junto con las tres funciones de evaluación produjeron nueve configuraciones experimentales que fueron probadas con el algoritmo de búsqueda de secuencias, arrojando un conjunto de secuencias resultado cada una. Como programas lógicos de caso de prueba se utilizaron los ejemplos E1 y E2. Por tanto, los conjuntos resultantes consisten en siete secuencias para E1 y cuatro para E2. Cabe señalar que estos resultados son producto de un análisis *a nivel*

de complejo lo que significa que no se consideran concentraciones de entrada para las diferentes hebras de las implicaciones. Por este motivo, para verificar la correcta hibridación de las hebras en un contexto de *tubo de ensayo*, cada conjunto resultante de secuencias fue evaluado usando la herramienta NUPACK en línea, que permite estos análisis (Zadeh *et al.* (2010)).

El objetivo final fue obtener el mejor conjunto de secuencias, es decir, aquel cuyas concentraciones en equilibrio producen una molécula objetivo. En otras palabras, se buscó que la mayor parte de la concentración de las hebras constitutivas de una implicación se consumiera formando implicaciones, eliminando al máximo las concentraciones de hebras sin hibridar. Para este análisis a nivel de tubos se utilizaron las mismas concentraciones de sales descritas arriba pero a una $T = 25°C$. El motivo de este cambio fue que inicialmente se probó con $T = 37°C$ y, aunque también arrojó formaciones correctas en general, se produjeron formaciones marginales de moléculas basura (hebras sueltas). Otra razón que soporta esta decisión es que la mayoría de trabajos experimentales utiliza la temperatura de 25°C para demostrar sus esquemas.

3.1.4. Simulación en MULTISTRAND

Las reacciones de interferencia identificadas en el modelo se describen en la Sección 4.6.1. Una característica destacable de estas, es que ambas incluyen un paso de migración de ramas de 4 vías en el dominio genérico. En estudios experimentales, se ha obtenido que las tasas de reacción de estos pasos, son algunos órdenes de magnitud más lentas que su contraparte de 3 vías (Panyutin y Hsieh, 1994; Dabby, 2013). Independientemente de estos resultados, en esta parte nos enfocamos en obtener una medida de su propensión, a través de simulaciones de trajectorias de pasos elementales con MULTISTRAND (Schaeffer *et al.*, 2015). Se consideraron dos escenarios para estas simulaciones: (1) tomando las reacciones globales (en un solo paso), y (2) siguiendo un enfoque paso a paso a nivel de dominios. Las secuencias utilizadas, configuración de parámetros y el código principal de implementación en MULTISTRAND se encuentran en el Anexo C. En estas simulaciones, se incluyeron también las reacciones diseñadas, con el propósito de comparar la propensión contra las reacciones no deseadas. Para el escenario de reacciones en un solo paso, se ejecutaron 1000 trajectorias de cada reacción, mientras que para el escenario de reacciones por pasos se ejecuta-

ron 2400 trayectorias de cada paso, exceptuando el paso de migración de 4 vías en la reacción implicación-implicación, del que se simularon solamente 600 trayectorias, debido a su costo computacional.

3.1.5. Simulación masa-acción en DIZZY

En esta parte, nos interesamos en estimar el efecto de las dos reacciones no deseadas en el contexto de programas lógicos específicos. Para esto, se consideró de particular interés el programa E1, dado que presenta varias reacciones de este tipo. El modelo bioquímico fue el mismo que se obtuvo de la compilación con Visual DSD, con las mismas constantes de reacción (Sección 3.1.2.2), pero agregando manualmente las reacciones no deseadas y las que se derivan de estas. Para la constante de migración de ramas de 4 vías en las reacciones no deseadas, se tomó el valor de 2.7 s (tiempo promedio de un paso elemental) que fue sugerido en los resultados de Dabby (2013). Se realizaron 20 simulaciones estocásticas independientes (algoritmo Gibson-Bruck) para obtener un promedio. Se consideraron tres escenarios para las concentraciones base de x ={5, 10, 20} nM. Conforme a estas concentraciones base, las concentraciones iniciales de las moléculas del programa fueron: $1x$ para todos los hechos y la consulta inicial, mientras que $2x$ para las reglas generales. El código de implementación en DIZZY se incluye en el Anexo D.

3.2. Representación de lógica de doble riel basada en toeholds

En esta sección, se describe la metodología empleada en una adaptación al modelo de circuitos lógicos con las llamadas compuertas sube-baja de Qian y Winfree (2009). Tal modelo es, a la fecha, uno de los más prometedores en el área del cómputo biomolecular. El cambio propuesto se basa la observación de que la lógica de doble carril utiliza dos señales por separado para los estados alto (“1”) y bajo (“0”). Primero, se explica la lógica de doble carril. Posteriomente, se detallan los procedimientos para la simulación de los circuitos en dos escenarios: la simulación de una compuerta básica se realizó en Visual DSD y la de circuitos a mayor escala en el software CAIN. Se describe la configuración de los modelos y parámetros bioquímicos para ambos casos.

3.2.1. Lógica de doble carril

En los modelos de circuitos lógicos basados en desplazamiento de hebras de ADN, la implementación de la función NOT es complicada (Qian y Winfree, 2011). Para la compuerta NOT se requiere que una baja — o nula — concentración de una molécula de entrada, produzca una alta concentración de otra de salida, y viceversa. Para ambas situaciones se tienen ya los mecanismos bioquímicos necesarios por separado: una reacción de amplificación (suponiendo que el modelo para el "0" lógico considere bajas concentraciones) para la primera y una de inhibición o umbralización, para la segunda. El problema es que los componentes moleculares actúan en difusión, de manera que no ha sido posible implementar un mecanismo que distinga entre ambas situaciones, en un tiempo determinado de la computación.

Para solventar esta cuestión, la alternativa más socorrida es utilizar la lógica de doble carril, que permite traducir circuitos AND-OR-NOT a circuitos AND-OR equivalentes. En esta función, las entradas y salidas se duplican, representando explícitamente las señales para unos y ceros lógicos. Es decir, para cada entrada x_i se asigna $x_i^1 = 1$ cuando $x_i = 1$ y $x_i^0 = 1$ cuando $x_i = 0$ (lo mismo para cada salida y_j). Con esto, cada compuerta AND, OR, NAND, NOR del circuito, se sutituye por un par de compuertas AND-OR (Figura 22).

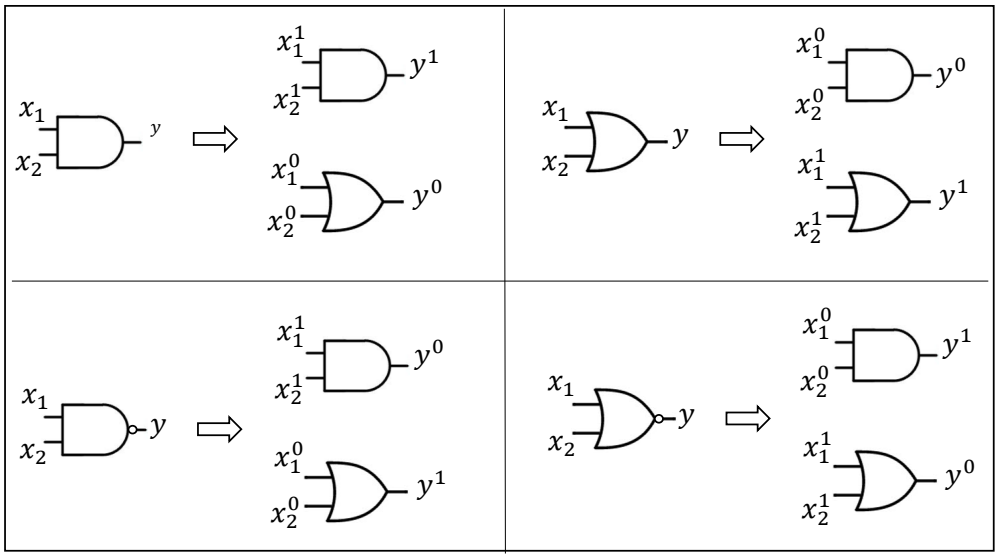

Figura 22. Equivalencia de compuertas en lógica de doble carril.

3.2.2. Simulación de compuertas básicas

En una primera instancia, se consideró el funcionamiento de una sola compuerta doble carril. Para ello, se utilizó el compilador *see-saw* (Qian y Winfree, 2011) disponible en línea[2] que permite generar código en Visual DSD para implementar compuertas doble carril, a partir de la definición de circuitos AND-OR-NOT. La utilidad de tal compilador ya ha sido demostrada en estudios previos para la construcción sistemática de circuitos a gran escala utilizando hebras de ADN sin purificar (Thubagere *et al.*, 2017).

Con el código generado automáticamente por el compilador, se tomaron los bloques de construcción para el esquema original de Qian y Winfree, y se agregaron manualmente los bloques para el esquema modificado de doble toehold. La compilación de cada una de las compuertas doble carril (AND, OR, NAND, NOR) en la semántica más simple de Visual DSD — "infinito" — produjo 33 especies diferentes y 22 reacciones para el modelo de doble toehold. Para el modelo original, esta misma semántica generó 34 especies y 22 reacciones. Este tipo de compilación es la más cercana a un modelo ideal que no incluye reacciones improductivas, por lo que ambos diseños tuvieron resultados de simulación cuasi ideales e idénticos (no incluidos aquí).

La siguiente semántica de compilación — denominada "por defecto" — de Visual DSD, es la que resulta interesante analizar, dado que incluye las reacciones espurias. Al compilar en este modo, se produce un aumento considerable en la complejidad del modelo generado incluyendo 271 especies en 1052 reacciones para el esquema doble toehold y 462 especies en 1908 reacciones para el no modificado. Tal aumento se debe a que, para incluir reacciones no productivas, es necesario permitir también reacciones de polimerización, en la que dos compuertas se unen a través de una molécula cable. A su vez, esto se debe a la forma en que se construyen los dominios de reconocimiento en el compilador *see-saw* para su simulación en Visual DSD, en donde cada dominio se tiene que poner entre dos puntos de apoyo que forman parte del mismo, con el fin de poder simular la reacción competitiva de una molécula cable con sus respectivas moléculas umbral y amplificadora.

[2] http://www.qianlab.caltech.edu/SeesawCompiler/

3.2.2.1. Configuración de parámetros

La simulación de una compuerta NOR en Visual DSD se llevó a cabo tomando en cuenta dos configuraciones. En la primera, se asignaron las constantes de unión k_f y desunión k_r de los toeholds, con unión lenta de 0.000015 nM^{-1} s^{-1} y desunión rápida de 0.95 s^{-1}. En la segunda, se utilizaron las constantes normales con valores de $k_f = 0.0003$ nM^{-1} s^{-1} y $k_r = 0.1126$ s^{-1}. En ambas simulaciones, la constante de migración de ramas es el valor por defecto en Visual DSD de $k_{bm} = 8000$ s^{-1}. Estos valores son los sugeridos por los autores de la compuerta sube-baja e incluidos en el código de Visual DSD generado por su herramienta de compilación en línea.

3.2.3. Simulación de circuitos lógicos de mayor escala

La simulación en Visual DSD está limitada a circuitos muy simples o a utilizar la semántica de compilación más básica, que no incluye las reacciones improductivas. Por esta razón, con el propósito de obtener resultados de simulación de circuitos a mayor escala y explorar el efecto de la segmentación en el esquema de doble toehold, se utilizaron otros paquetes de simulación masa-acción (DIZZY de Ramsey *et al.* (2005) y CAIN[3]).

Como caso de referencia se seleccionó el circuito que calcula $\lfloor \sqrt{n} \rfloor$, donde n es un número de cuatro bits $x_4x_3x_2x_1$ (Qian y Winfree, 2011). Tal circuito es, a la fecha, el de mayor escala implementado usando reacciones de desplazamiento de hebras en laboratorio. La Figura 23 muestra dicho circuito en lógica AND-OR-NOT y su correspondiente representación con lógica de doble carril.

El modelo químico correspondiente para la simulación de este sistema (sin considerar las reacciones de fuga (*leakage*), así como las constantes de reacción y una simplificación para reducir la cantidad de reacciones no productivas fueron tomadas según lo sugerido en el material suplementario de Qian y Winfree (2011).

Tal simplificación fue indispensable puesto que la cantidad de reacciones improductivas que se generan en el circuito es muy grande, lo que genera un alto costo computacional para los simuladores estocásticos y aún para los determinísticos. En

[3]Software disponible en: http://cain.sourceforge.net/

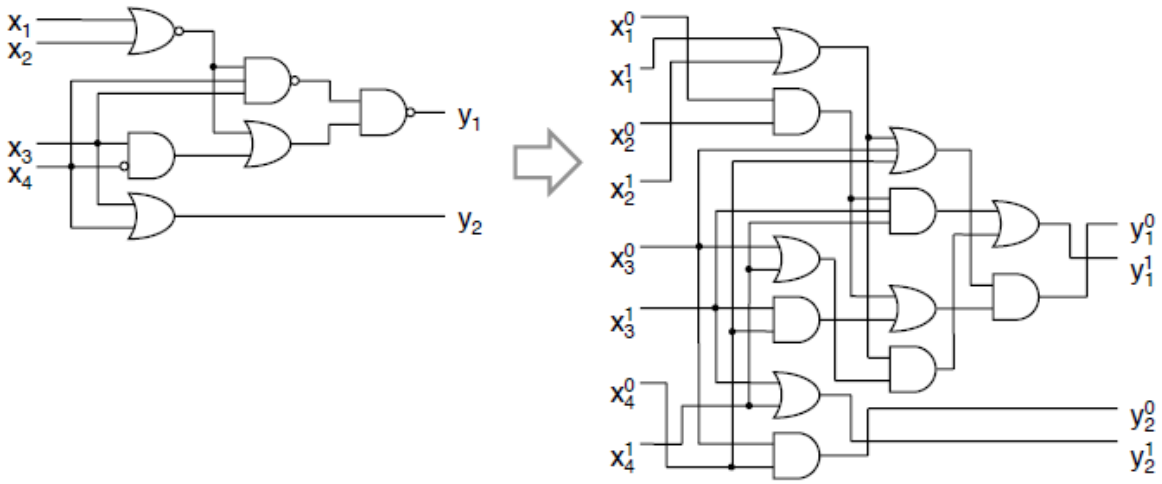

Figura 23. Circuitos para calcular $\lfloor \sqrt{n} \rfloor$, donde n es un número de cuatro bits.

efecto, esta cantidad se calcula a partir de las formas en que una molécula cable $w_{i,j}$ se puede unir a otra que tenga algún toehold complementario expuesto. En el circuito, por ejemplo, se tienen 62 diferentes moléculas cable (ver Figura 84 en Anexo F) y un total de 116 moléculas que incorporan algún toehold complementario. Así, de manera burda, se obtendrían 14384 reacciones improductivas de unión y desunión ($62 \times 116 \times 2$).

Es importante notar que para el caso del modelo con doble toehold, dichas reacciones disminuyen considerablemente. Esto es debido a la segmentación que se establece en el circuito gracias al uso de diferentes toeholds. Tal segmentación consiste en que una molécula $w^0_{i,j}$ nunca podrá generar reacciones improductivas con compuertas de toehold T_1, y viceversa. Dado que en el circuito doble toehold las 62 diferentes moléculas cable se dividen en 31 con T_0 y 31 con T_1, la cantidad de reacciones no productivas se reduce a la mitad. Lo anterior puede generalizarse para cualquier circuito. Si un sistema contiene m moléculas cable y n moléculas con toehold complementarios, las reacciones improductivas serán aproximadamente $m \times n$ en el modelo original, mientras que en doble toehold se tendrán $m/2 \times n/2$ (para T_0) $+ m/2 \times n/2$ (para T_1) $= (m \times n)/2$.

No obstante, como se mencionó anteriormente, para simular ambos esquemas con el mismo modelo químico se aplicó la simplificación sugerida por Qian y Win-

free (2011). Esta consiste en utilizar especies “acumuladas” (“*lumped*”) $\mathbb{W}$, $\mathbb{TH}$ y $\mathbb{G}$, respectivamente, para las moléculas cable, umbral y compuertas (incluyendo reporteras). La concentración de estas especies acumuladas se inicializa con la suma de las concentraciones iniciales de las moléculas que representan. De esta manera, el modelo queda de la siguiente forma:

$$w_{i,j} + G_{j:j,k} \underset{k_s}{\overset{k_s}{\rightleftharpoons}} G_{i,j:j} + w_{j,k}, \tag{6}$$

$$w_{j,F} + G_{i,j:j} \underset{k_s}{\overset{k_s}{\rightleftharpoons}} G_{j:j,F} + w_{i,j}, \tag{7}$$

$$w_{i,j} + T_{i,j:j} \xrightarrow{k_f} \text{desperdicio}, \tag{8}$$

$$w_{i,j} + \text{Rep}_j \xrightarrow{k_s} \text{Fluor}_j, \tag{9}$$

$$\mathbb{W} + G_{i:i,k} \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} \mathbb{W} : G_{i:i,k}, \tag{10}$$

$$G_{j,i:i} + \mathbb{W} \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} G_{j,i:i} : \mathbb{W}, \tag{11}$$

$$\mathbb{W} + T_{j,i:i} \underset{k_{rs}}{\overset{k_f}{\rightleftharpoons}} \mathbb{W} : T_{j,i:i}, \tag{12}$$

$$w_{j,i} + \mathbb{G} \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} w_{j,i} : \mathbb{G} \tag{13}$$

$$\text{y } w_{j,i} + \mathbb{TH} \underset{k_{rs}}{\overset{k_f}{\rightleftharpoons}} w_{j,i} : \mathbb{TH}. \tag{14}$$

En estas reacciones, el símbolo “:” representa la concatenación de los reactivos, es decir, una especie intermedia que contiene ambas especies de entrada unidas por el toehold. La especie Fluor_j denota la emisión de fluroescencia al separase el fluoróforo de su respectivo supresor. Las ecuaciones (6)-(9) representan las reacciones productivas de sube-baja, repostaje, umbralización y reporte, respectivamente. Las ecuaciones (10)-(14) introducen de manera aproximada el comportamiento ruidoso que causan las reacciones no productivas. El circuito original QW suma un total de 156 reacciones de los tipos (6)-(9) y 474 de los tipos (10)-(14) (separando las reacciones reversibles).

El modelo bioquímico para el sistema con doble toehold resulta más extenso, dado que se tienen que considerar todas las combinaciones de los toeholds en ca-

da una de las ecuaciones (6)-(14). Por ejemplo, la ecuación (6), tendrá ocho combinaciones en el esquema doble toehold (es decir, las combinaciones $\{w^0_{i,j}, w^0_{i,j}\} \times \{G^{00}_{j:j,k}, G^{01}_{j:j,k}, G^{10}_{j:j,k}, G^{11}_{j:j,k}\}$). A pesar de ser un modelo más intrincado, la suma de todos los tipos de reacciones (6)-(9) y de los tipos (10)-(14) es 152 y 476, respectivamente. Finalmente, los valores para las constantes de reacción utilizados en las simulaciones de ambos modelos son:

- $k_s = 5 \times 10^4$ M^{-1} s^{-1}
- $k_f = 2 \times 10^6$ M^{-1} s^{-1}
- $k_{rs} = 1.3$ s^{-1}
- $k_{rf} = 26$ s^{-1}

Capítulo 4. Resultados y discusión: Un esquema de inferencia lógica con cascadas de desplazamiento de hebras de ADN

En esta sección se presentan los resultados del esquema de inferencia lógica. Primero se describe la configuración de las moléculas y reacciones del esquema. Después se explica un procedimiento sencillo para la validación termodinámica del modelo. Por último, se presentan resultados de simulación que incluyen reacciones de interferencia identificadas y su afectación al desempeño. Los resultados obtenidos de este esquema, han sido publicados en Ordóñez-Guillén y Martínez-Pérez (2019).

4.1. Codificación molecular

El esquema de inferencia lógica que se presenta a continuación toma inspiración del trabajo más representativo del área (Ran *et al.*, 2009), el cual es, a la fecha, la única implementación de laboratorio dentro de los modelos autónomos de inferencia. También se adopta la misma convención de tal modelo, en la que los predicados instanciados se representan a través de expresiones proposicionales. Con esta convención, un programa lógico que consiste en un conjunto finito de hechos y reglas, corresponde a un conjunto de proposiciones $P = \{p_1, p_2, \cdots p_n\}$ donde cada una de ellas corresponde a su vez a un dominio representativo único. El esquema de codificación también incluye tres secuencias fijas: dos dominios toehold $\{t_1, t_2\}$ y un dominio largo identificado como GRC (genérico) que no codifica ninguna variable del sistema. Este último dominio siempre aparece en medio de dos dominios representativos iguales X para formar un segmento de migración de ramas de la forma $[X{-}\text{GRC}{-}X]$. La razón o necesidad detrás del uso de un dominio genérico en el diseño, tiene que ver con la restricción de mantener una configuración universal en las entradas/salidas.

Una vez revisadas las convenciones del esquema a nivel de dominios, pasamos a la configuración molecular. El esquema de inferencia lógica del presente estudio se implementa con tres moléculas básicas: consulta, proposición e implicación. Se agrega además una molécula extra, denominada “combustible”, que no interviene en el funcionamiento lógico del esquema pero es fundamental para el comportamiento

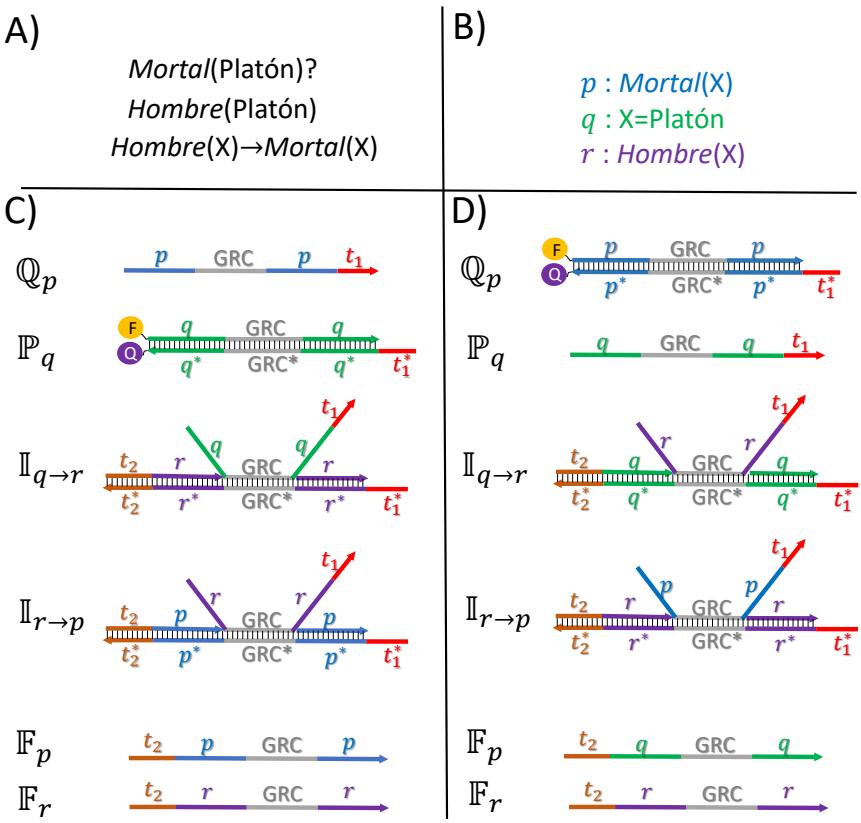

Figura 24. Los bloques de construcción para representar programas lógicos. A) Un programa simple de ejemplo. B) Las proposiciones involucradas y su forma abreviada. C) Representación para encadenamiento hacia atrás. D) Representación para encadenamiento hacia adelante.

catalítico, como se verá más adelante. Para una notación abreviada, se utilizarán las letras $\mathbb{Q}$, $\mathbb{P}$, $\mathbb{I}$ y $\mathbb{F}$, respectivamente, para cada una de estas cuatro moléculas básicas. Se incluirán subíndices para indicar la proposición (o proposiciones, en el caso de las implicaciones) que codifica cada molécula. Para ilustrar la configuración molecular, consideraremos el mismo ejemplo de programa lógico de la Sección 1.5.1 (mostrado en la Figura 24A). Este programa requiere de tres proposiciones (i.e. dominios) que en este caso identificamos como *p, q* y *r*, cada uno con un color representativo (Panel B). Dependiendo de la estrategia de inferencia aplicada, encadenamiento hacia adelante o hacia atrás, la codificación molecular es diferente. Los paneles C y D de la Figura 24 la muestran para cada uno de estos casos, respectivamente. En desplazamiento hacia atrás:

- Una molécula ssDNA que contiene un segmento de migración de ramas [*p*—GRC—*p*] y un toehold t_1 en el extremo 3', representa a la consulta $\mathbb{Q}_p$.

- La proposición $\mathbb{P}_q$ se representa con una molécula detectora, es decir, un fragmento dsDNA con el segmento de migración [q—GRC—q] y un toehold t_1 expuesto. Para propósitos de detección FRET, tal molécula contiene agregados un fluoróforo y un supresor en su extremo obtuso.

- Las implicaciones tales como $\mathbb{I}_{q\to r}$ y $\mathbb{I}_{r\to p}$ son complejos compuestos por cuatro hebras, tres de ellas en la parte superior unidas a una hebra sustrato complementaria. Tomando de ejemplo $\mathbb{I}_{r\to p}$, esta tiene una hebra de cobertura en el lado izquierdo para el segmento [$t_2 - p$], una hebra salida unida por el dominio GRC y una hebra de cobertura en el lado derecho para el dominio p. El segmento de migración [p—GRC—p] contenido en la hebra sustrato se corresponde al de una consulta de entrada $\mathbb{Q}_p$ (el consecuente de la implicación). La hebra de salida corresponde a una nueva consulta $\mathbb{Q}_r$ (el antecedente de la implicación). Note como el diseño de las implicaciones asegura la universalidad de las hebras salida, sin importar de cuál regla se trate (en otras palabras, sin importar el dominio del consecuente).

- Una molécula combustible $\mathbb{F}_p$, es una ssDNA que siempre se asocia con las implicaciones que contienen el segmento de migración de ramas [p—GRC—p] (el consecuente, en este caso) y contiene un toehold t_2 en el extremo 5’.

Para el caso de encadenamiento hacia adelante, los cambios necesarios en la representación molecular se interpretan de manera evidente (Figura 24D). En primer lugar, las representaciones de consulta y proposición se intercambian de lo dispuesto para encadenamiento hacia atrás. Así, las consultas son moléculas detectoras y las proposiciones son moléculas ssDNA. En segundo lugar, los dominios representativos en las implicaciones también se intercambian, de manera que el dominio del antecedente ahora se encuentra en el segmento de migración de ramas, mientras que el del consecuente forma parte de la hebra de salida. Las moléculas de combustible mantienen su configuración, siempre correspondiendo al segmento de migración de ramas de su respectiva implicación asociada.

4.2. Reacciones básicas

De acuerdo a lo revisado en la Sección 2.6.1, las operaciones para la implementación de encadenamiento hacia atrás son dos: **consulta-implicación** y **consulta-proposición**. A estas operaciones se añade una tercera denominada **repostaje** (del inglés *fueling*) en la que intervienen una molécula combustible con el producto de la reacción consulta-implicación. Estas reacciones se ilustran en la Figura 25 para el programa lógico más simple que incluye una consulta como ¿*Hombre*(Sócrates)? que se responde con su correspondiente hecho *Hombre*(Sócrates). Las moléculas $\mathbb{Q}_q$, $\mathbb{P}_r$ y $\mathbb{I}_{r\to q}$ codifican este programa con las proposiciones r:"X=Sócrates" y q:"*Hombre*(X)". Agregando concentraciones adecuadas de estas moléculas, junto con el combustible $\mathbb{F}_q$ las reacciones proceden de la siguiente manera:

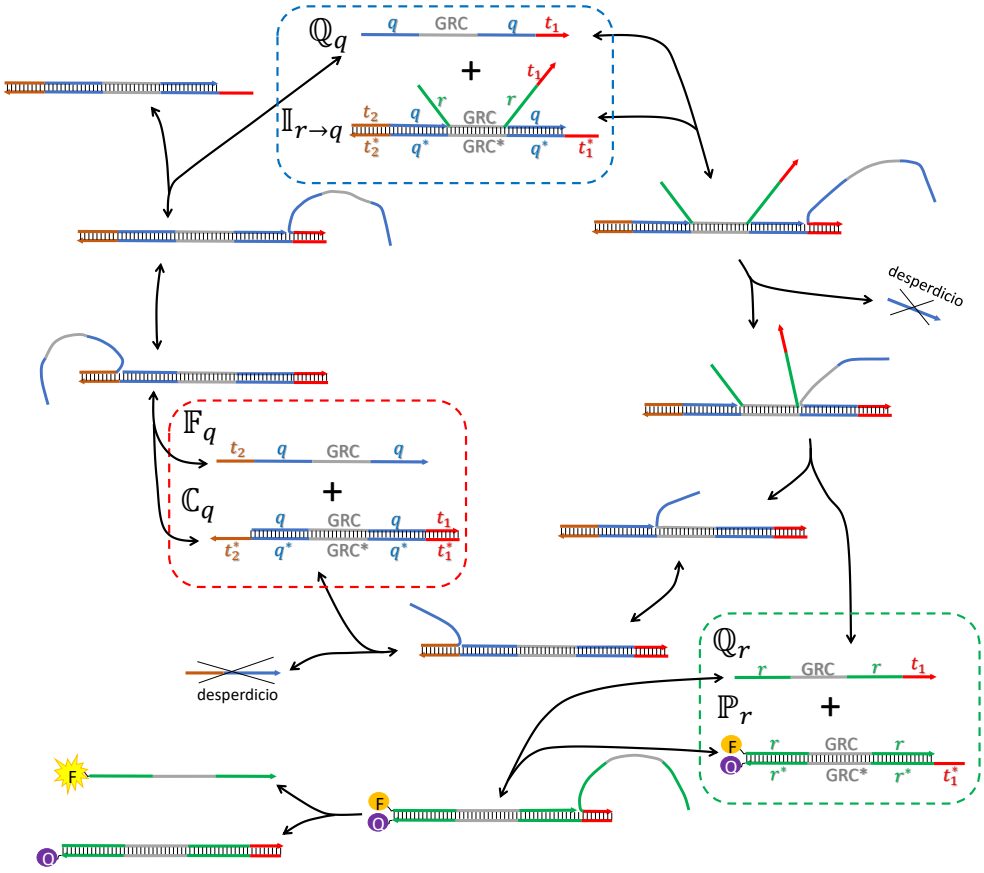

Figura 25. Las tres reacciones básicas para encadenamiento hacia atrás. Los recuadros con líneas punteadas indican el inicio de cada una de ellas.

- Una reacción consulta-implicación ($\mathbb{QI}$) inicia con $\mathbb{Q}_q$ y $\mathbb{I}_{r\to q}$ (recuadro azul) mediante la unión de sus toeholds complementarios t_1. Posteriormente, el proceso

de migración de ramas del segmento [q—GRC—q] empieza con dos pasos irreversibles: primero desplazando la hebra de cobertura derecha (que se considera desperdicio), seguido de la consulta de salida $\mathbb{Q}_r$. El último paso consiste en un intercambio de toeholds con la hebra de cobertura izquierda. A pesar de que esta última es una molécula reactiva, se considera también desperdicio. Además de la consulta salida, al término de la reacción se produce el complejo $\mathbb{C}_q$ que contiene la consulta de entrada completamente unida a la hebra sustrato de la implicación, que ahora mantiene el toehold t_2 expuesto.

- Una reacción consulta-proposición ($\mathbb{QP}$) se da entre la consulta de salida $\mathbb{Q}_r$ de la reacción anterior y la proposición $\mathbb{P}_r$ (recuadro verde). En esta, a diferencia de la reacción consulta-implicación, el paso de migración de ramas del segmento [r—GRC—r] es solamente uno. El desplazamiento de la hebra de cobertura que contiene adjunto el fluoróforo, produce la fluorescencia que se interpreta como una inferencia positiva.

- Una reaccion de repostaje se produce entre la molécula combustible $\mathbb{F}_q$ y el complejo $\mathbb{C}_q$ resultante de la reacción $\mathbb{Q}_q$+$\mathbb{I}_{r\to q}$ (recuadro rojo). Iniciando con el toehold t_2 expuesto para continuar con la migración de ramas del segmento [q—GRC—q], esta reacción intercambia el combustible con la consulta de entrada $\mathbb{Q}_q$. La recuperación de $\mathbb{Q}_q$ induce una reacción en una nueva implicación, convirtiendo efectivamente esta consulta en un agente catalítico.

Las reacciones para implementar encadenamiento hacia adelante proceden exactamente con la misma dinámica. La diferencia consiste únicamente en la interpretación. Al estar intercambiadas las representaciones de consulta y proposición, la reacción $\mathbb{QI}$ se convierte en $\mathbb{PI}$ (**proposición-implicación**). La reacción $\mathbb{QP}$ sigue involucrando las mismas moléculas solo que intercambiadas, así que mantiene su acrónimo. Lo mismo para la reacción de repostaje, que en este caso libera una proposición de entrada.

El esquema de inferencia de encadenamiento hacia adelante puede permitir la introducción de reglas con conjunción de proposiciones como antecedente, aumentando el poder computacional del modelo. Una regla con la conjunción $p \wedge q$ como premisa, denotada por $\mathbb{I}_{p\wedge q\to r}$, se puede definir como una implicación sencilla $\mathbb{I}_{q\to r}$ extendida con un segmento dsDNA [p—GRC—p] y un toehold t_1 expuesto. El motivo para esta

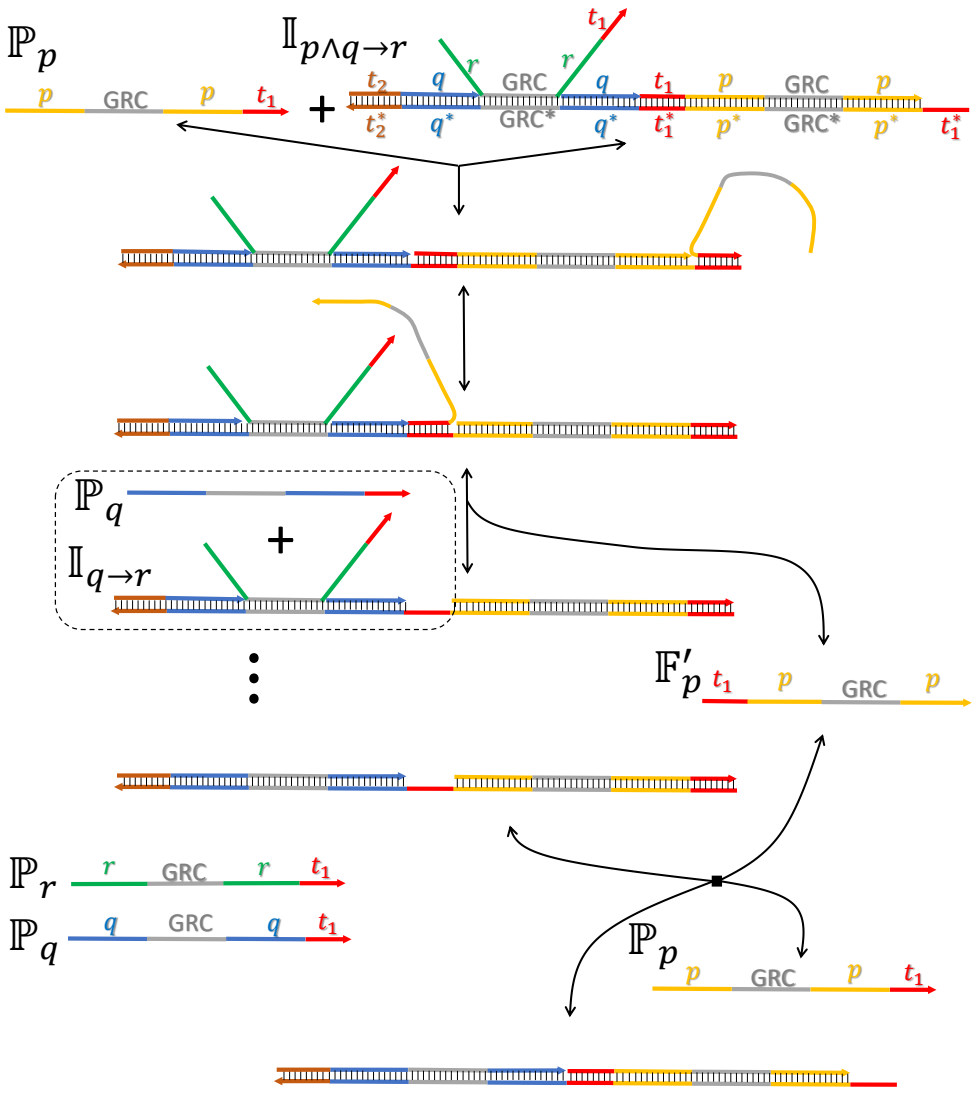

Figura 26. Diagrama esquemático de las reacciones con reglas que contienen conjunción de proposiciones como premisa.

regla doble, así como la secuencia de reacciones que liberan la proposición del consecuente $\mathbb{P}_r$ se puede apreciar en la Figura 26. En presencia de la proposición $\mathbb{P}_p$, una reacción de intercambio de toeholds se inicia, liberando la hebra que cubre el segmento $[t_1 - p\text{—GRC—}p]$ y activando el toehold t_1 interno en el proceso. En este punto, la acción de la proposición $\mathbb{P}_q$ para liberar la salida $\mathbb{P}_r$ y la recuperación de $\mathbb{P}_q$ con el repostaje vía $\mathbb{F}_q$, procede exactamente igual que en el caso de la implicación simple (nótese la subestructura idéntica encerrada en el recuadro). Cabe señalar que, aunque la recuperación de la primera proposición $\mathbb{P}_p$ no puede realizarse con una molécula combustible, es posible agregar una concentración extra de la hebra cobertura del

segmento $[t_1 - p\text{—GRC—}p]$, de manera que haga las veces de combustible (denotada por $\mathbb{F}_p^{'}$ en esta parte de la reacción.

4.3. Implementación de programas lógicos

Para efectos de aplicar el esquema lógico aquí introducido a programas lógicos específicos, se seleccionaron los casos de prueba de Ran *et al.* (2009). Para demostrar el funcionamiento del algoritmo de encadenamiento hacia atrás, se tomaron los casos de prueba denominados “consultas existenciales”. Por otro lado, para probar el caso de encadenamiento hacia adelante usando reglas con conjunción, se incluye el único ejemplo que produce inferencia positiva de tal trabajo.

4.3.1. Ejemplo E1: consultas existenciales

La Figura 27 muestra la correspondiente representación molecular y la cascada de reacciones simplificada que resuelve la consulta “¿Existe alguien apto para asistir a la academia?” con encadenamiento hacia atrás. Para resolver tal cuestión, se debe añadir algunas moléculas *suposición*, que asigna la respuesta a cada uno de los elementos del dominio. Así, estas moléculas corresponden a las proposiciones X=Sócrates, X=Alejandro y X=Hércules; y deben ser identificadas cada una con un fluróforo de color diferente, de manera que la respuesta sea distinguible.

Las siete proposiciones que contiene este programa {*Ejército*(X), *Fuerte*(X), *Academia*(X), *Sabio*(X), X=Sócrates, X=Alejandro, X=Hércules} se asignarán respectivamente a los símbolos $\{p, q, r, s, t, u, v\}$. De esta forma, las moléculas del programa son $\{\mathbb{I}_{q\to p}, \mathbb{I}_{s\to r}, \mathbb{I}_{t\to s}, \mathbb{I}_{u\to s}, \mathbb{I}_{u\to q}, \mathbb{I}_{v\to q}, \mathbb{P}_t, \mathbb{P}_u, \mathbb{P}_v\}$, y la consulta inicial es $\mathbb{Q}_r$. Agregando concentraciones adecuadas de estas moléculas en solución, el proceso de inferencia en retroceso inicia con una reacción $\mathbb{QI}$ entre $\mathbb{Q}_r$ e $\mathbb{I}_{s\to r}$. El producto resultante $\mathbb{Q}_s$ puede entonces participar en dos reacciones QI de manera simultánea, una con la implicación $\mathbb{I}_{t\to s}$ y la otra con $\mathbb{I}_{u\to s}$. Al ser dos las implicaciones objetivo, la concentración producto de $\mathbb{Q}_s$ se reparte entre ellas. Las subsecuentes consultas $\mathbb{Q}_t$ y $\mathbb{Q}_u$ son terminales, de manera que cada una de ellas se responde vía reacciones QP con $\mathbb{P}_t$ y $\mathbb{P}_u$, respectivamente . Así, se produce la fluorescencia indicando las respuestas “Sócrates” y “Alejandro”.

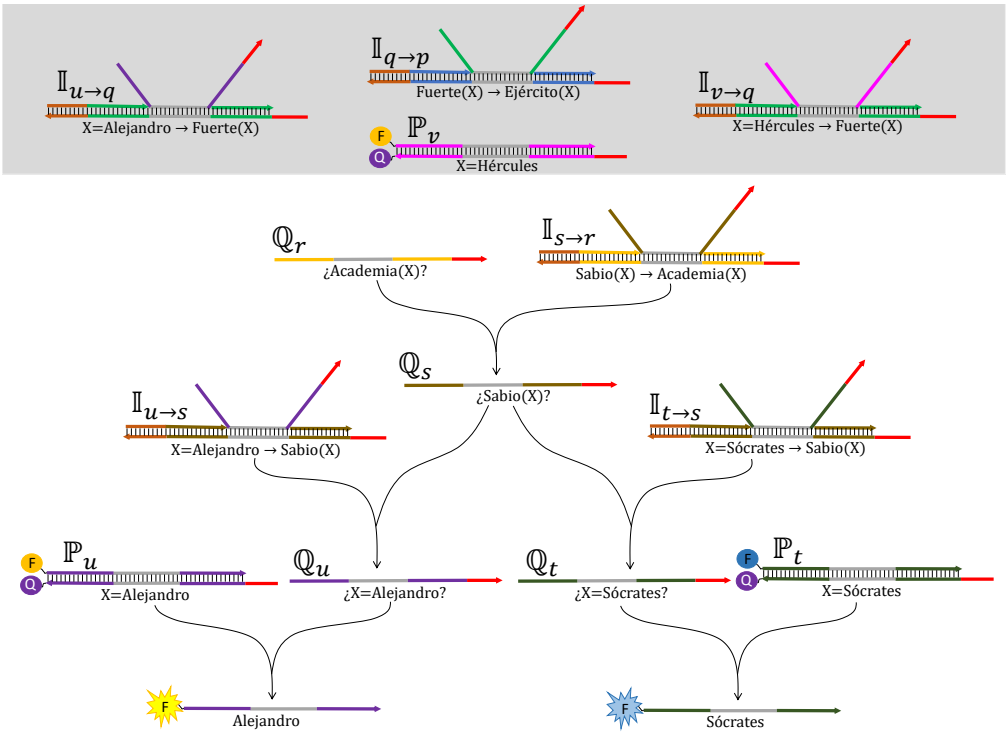

Figura 27. Proceso de encadenamiento hacia atrás para el ejemplo E1 con la consulta inicial "¿Quién es apto para la academia?". Al final del proceso, el sistema responde "Sócrates" y "Alejandro". Las moléculas no reactivas aparecen en el rectángulo sombreado. Las reacciones de repostaje se omiten.

4.3.2. Ejemplo E2: inferencia hacia adelante con reglas compuestas

La Figura 28 muestra la implementación de este programa en el que se incluyen moléculas para la regla general *Enamorado*(X)∧ *Acaudalado*(X)→*Feliz*(X) y los hechos *Enamorado*(Maslow) y *Acaudalado*(Maslow). La consulta inicial ¿*Feliz*(Maslow)? se codifica con la proposición X=Maslow y la molécula detectora que representa a la consulta ¿*Feliz*(X)? Una vez más, se abrevian las proposiciones involucradas {X=Maslow, *InLove*(X), *HasMoney*(X), *Happy*(X)} con los símbolos $\{w, x, y, z\}$, respectivamente. Así, el programa se compone de las moléculas $\{\mathbb{I}_{x\wedge y\to z}, \mathbb{I}_{w\to x}, \mathbb{I}_{w\to y}, \mathbb{Q}_z\}$ junto con la proposición inicial $\mathbb{P}_w$.

Supongamos que tales moléculas se encuentran en solución bajo condiciones experimentales adecuadas. El proceso de inferencia hacia adelante se activa con $\mathbb{P}_w$ que efectúa operaciones PI de manera paralela con las implicaciones $\mathbb{I}_{w\to x}$ e $\mathbb{I}_{w\to y}$. Las proposiciones de salida $\mathbb{P}_x$ y $\mathbb{P}_y$ actúan, en ese orden, sobre la regla con conjunción. A partir de esto, la proposición terminal $\mathbb{P}_z$ se libera para finalmente reaccionar con la

consulta $\mathbb{Q}_z$ y producir la fluorescencia que indica la inferencia positiva.

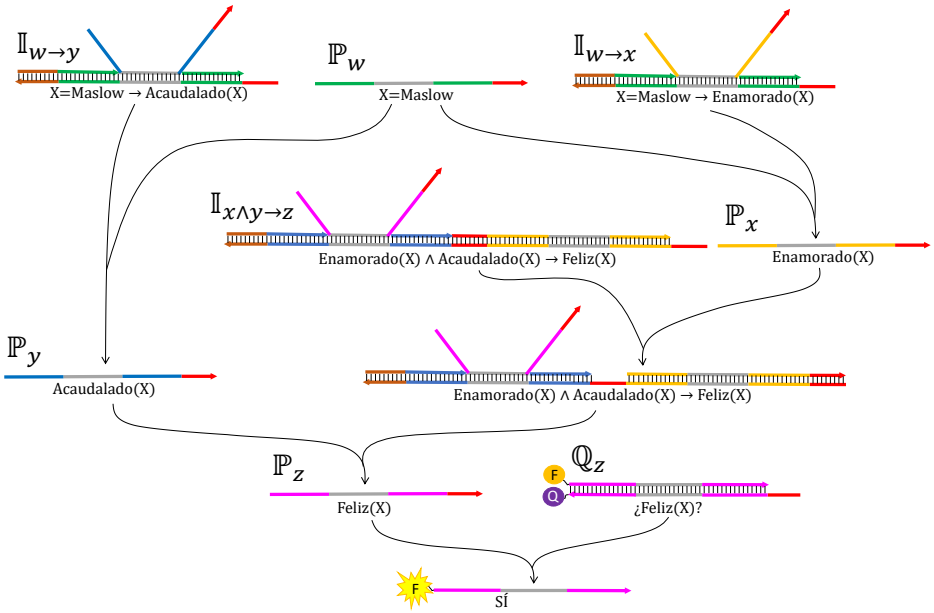

Figura 28. Implementación de encadenamiento hacia adelante para el ejemplo E2 con la consulta inicial *¿Feliz*(Maslow)? La consulta se responde positivamente dado que Maslow cumple con los atributos necesarios definidos. Las reacciones de repostaje se omiten.

4.4. Simulación en Visual DSD

Para evaluar la funcionalidad y el desempeño del diseño, se llevaron a cabo simulaciones utilizando Visual DSD (Lakin *et al.*, 2011), una herramienta ampliamente aceptada para la aproximación del comportamiento de los sistemas basados en desplazamiento de hebras de ADN (Qian y Winfree, 2011; Qian *et al.*, 2011b; Chen *et al.*, 2013; Amir *et al.*, 2014; Chatterjee *et al.*, 2017).

4.4.1. Resultados de casos E1 y E2

La compilación del ejemplo E1 generó, en total, 128 reacciones y 90 especies para el modo CAT. Para la contraparte NCAT, las reacciones y especies fueron 92 y 68, respectivamente. La Figura 29 despliega los resultados de simulación para ambas versiones. En el Panel A, se delinea la evolución cinética de las consultas. Al principio, la consulta inicial *¿Academia*(X)? disminuye drásticamente en ambas versiones (curvas rojas). Sin embargo, para el programa NCAT tal reducción es exponencial, alcanzando

un agotamiento substancial después de 2000 s. En contraste, en la versión CAT, la concentración para esa misma molécula se eleva hasta cerca de 11000 moléculas al final de la simulación. En efecto, esta marcada diferencia puede ser explicada por el efecto de la reacción de repostaje, que estimula la recuperación de las moléculas de entrada. La concentración de las consultas de la primera cascada (curvas amarillas) también muestra un comportamiento totalmente opuesto, permaneciendo prácticamente constante a ∼9900 moléculas en el caso CAT y terminando casi agotada en el otro, después de ∼1000 s de simulación. La concentración de las consultas terminales (curvas traslapadas magenta y verde) se desarrolla similarmente en ambos casos, pero a una escala mayor en la versión CAT.

El desempeño final se ilustra en el Panel B. Se muestran solamente las señales que producen inferencia positiva, dado que la negativa (“Hércules”, en este caso), permaneció constante en cero en todas las simulaciones. Se puede observar que las señales para las versiones CAT y NCAT del programa presentan diferencias significativas en el rendimiento cinético, alcanzando una eficiencia de 83 % (∼16800 de 20000 moléculas) en la primera. En el segundo, en contraste, se obtuvo una eficiencia limitada de 46 % (∼9200 de 20000 moléculas). Esta disparidad, puede proporcionar una idea de la conveniencia de un esquema catalítico para la implementación de encadenamiento hacia atrás.

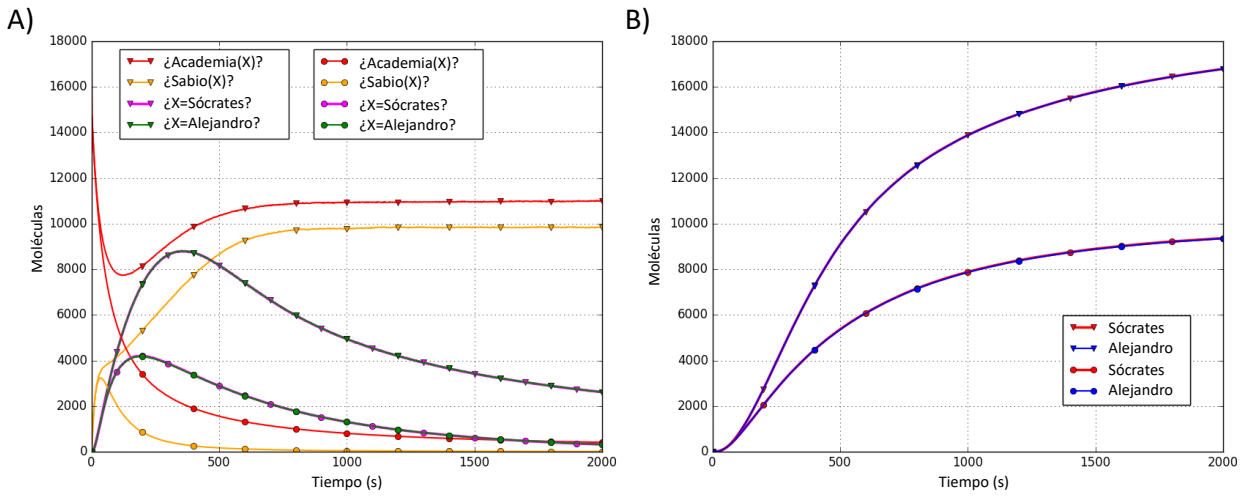

Figura 29. Resultados de simulación para el ejemplo E1 con la consulta inicial “¿Quién es apto para la academia?”. A) Comportamiento cinético de las consultas. B) Comportamiento cinético de las señales de salida. Los marcadores triangulares y circulares indican el modo CAT y NCAT, respectivamente.

Para el ejemplo E2, el modelo CAT contiene 115 reacciones con 76 especies, mien-

tras que el programa NCAT incluye 71 reacciones y 52 especies. La Figura 30 muestra el comportamiento cinético para este ejemplo. En particular, el Panel A muestra la evolución de las proposiciones participantes. Observe que la proposición inicial X=Maslow (curvas rojas) se divide en dos implicaciones. A consecuencia de esto, su concentración de agota rápidamente (después de ∼280 s) en el caso NCAT, mientras que en el otro, es estimulada por las moléculas combustible hasta alcanzar un estado casi estable a ∼11700 moléculas. Las proposiciones intermedias *Enamorado*(X) (curvas amarillas) y *Acaudalado*(X) (curvas magenta) se liberan casi a la misma tasa, aunque la sus curvas cinéticas difieren debido a que la primera empieza a ser consumida antes por la implicación de conjunción. Es evidente en las gráficas que la concentración de estas moléculas permanece mucho mayor en el caso CAT que en el NCAT. Finalmente, la concentración de las proposiciones terminales (curvas verdes) empieza a decrecer después de alcanzar un pico en ambos casos. Esto se debe a que se consumen directamente con su respectiva detectora, y no se contempla la recuperación en esta etapa. El efecto global de las reacciones de repostaje se refleja en la mejora significativa en la eficiencia final del enfoque CAT sobre la referencia NCAT (Panel B).

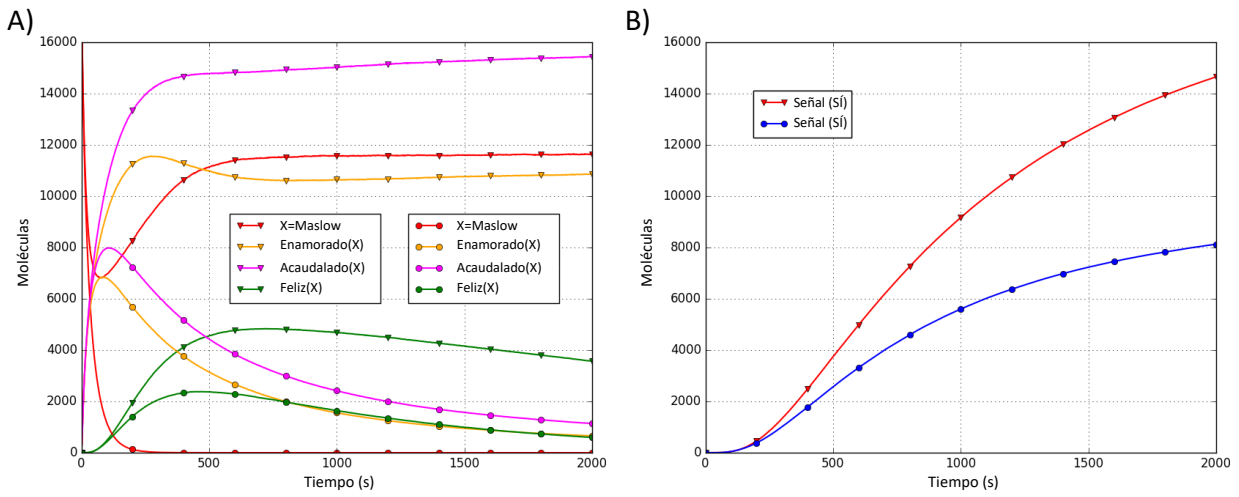

Figura 30. Resultados de simulación para el ejemplo E2. A) Comportamiento cinético de las proposiciones. B) Comportamiento cinético de las señales de salida. Los marcadores triangulares y circulares indican el modo CAT y NCAT, respectivamente.

4.4.2. Resultados de caso crítico E3

Con el propósito de obtener mayor evidencia del desempeño de los sistemas CAT, se incluye la simulación del escenario crítico, donde la ruta de inferencia positiva se ve afectada por varias rutas negativas (31A). En este programa, la concentración de la

consulta inicial ¿p? se divide en aproximadamente tres partes iguales para compartirse entre las implicaciones coincidentes. Esta misma situación ocurre para las consultas intermedias ¿q? y ¿r?. La consecuencia es que al final de la ruta de inferencia positiva, solo una fracción de la consulta terminal ¿s? será liberada y respondida con su correspondiente proposición s.

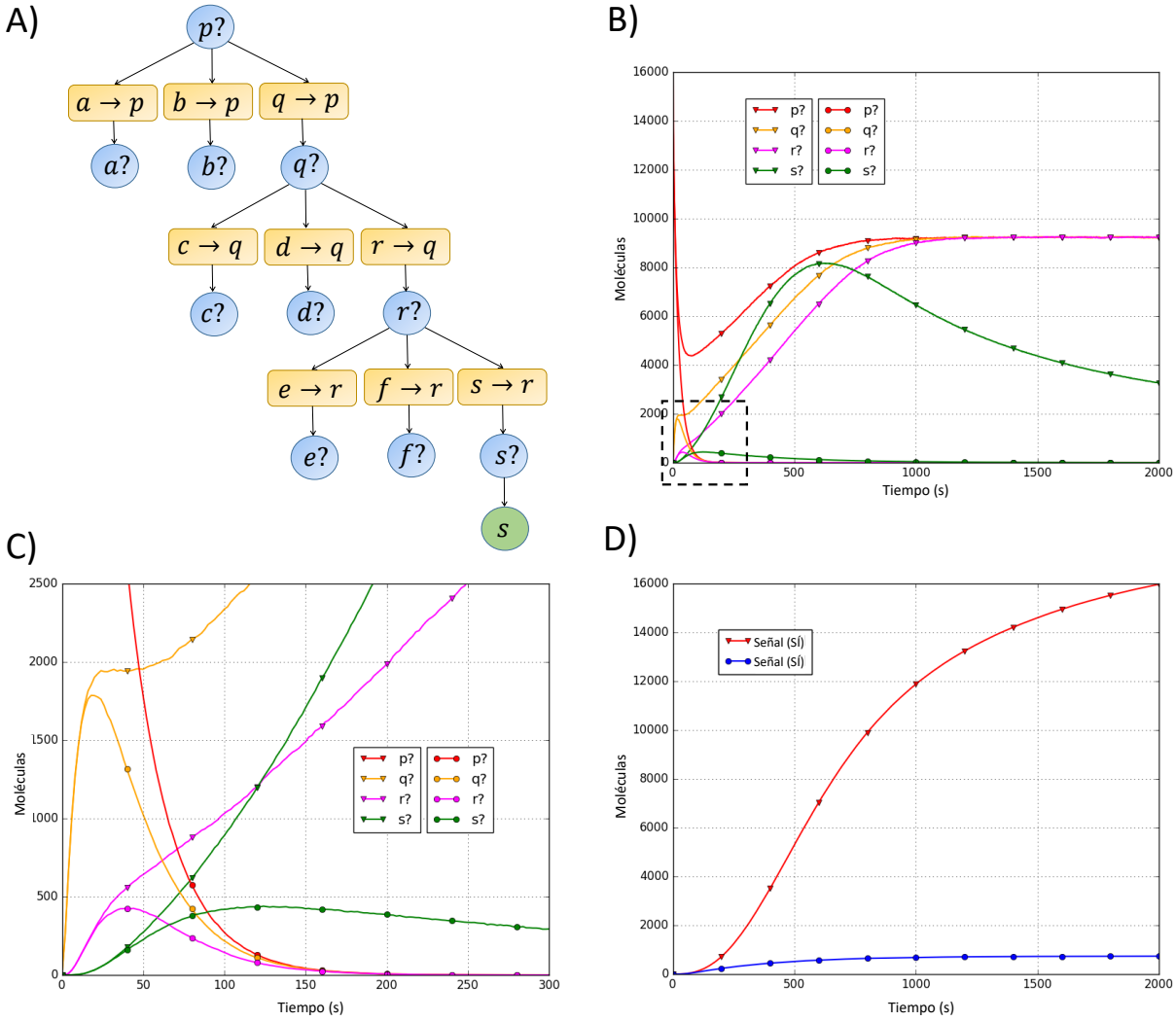

Figura 31. Ejemplo E3 y sus resultados de simulación. A) Programa lógico E3 y cascada de inferencias. B) Cinética de las consultas relevantes. C) Acercamiento de la región enmarcada en el Panel B. D) Cinética de las señales de salida.

Los modelos CAT y NCAT de este programa contienen, respectivamente, 327 reacciones con 127 especies y 243 reacciones con 152 especies. La Figura 31B muestra la cinética de las consultas involucrada en la ruta de inferencia positiva. Como puede observarse, la concentración de la consulta inicial p? (curvas rojas) cae drásticamente en ambas versiones, pero tiene un punto de inflexión en el modo CAT a ~78 s, donde

empieza a aumentar hasta oscilar alrededor de 9200 moléculas. Las consultas intermedias *q*? y *r*? se incrementan a tasas similares, alcanzando también un estado cuasi estacionario en 9200 moléculas. La consulta terminal *s*?, que es consumida directamente con la detectora, empieza a consumirse después de alcanzar un pico en ∼8100 moléculas.

En contraste, sin la acción de las moléculas combustible, las consultas permanecen en bajas concentraciones. La región enmarcada de la Figura 31B exhibe esta drástica disparidad contra las curvas CAT. Esta misma porción aparece magnificada en el Panel C, donde se puede distinguir como todas las curvas NCAT se encuentran abajo de 500 moléculas después de solo ∼75 s de simulación. Más aún, después de ∼200 s, las consultas *p*?, *q*? y *r*? ya están agotadas, mientras que la consulta final *s*? se mantiene en menos de 400 moléculas (lo que representa menos del 2 % de la máxima concentración posible).

En la Figura 31D se aprecia el comportamiento cinético de las señales de salida para ambas versiones. La amplia brecha entre ambas curvas, demuestra el mayor rendimiento de la versión catalítica, alcanzando hasta un 80 % de eficiencia (16000 de 20000 moléculas). Observe que este rendimiento aún podría mejorar considerando un mayor tiempo para que las reacciones se lleven a cabo. Esto se hace evidente analizando la pendiente de ambas curvas al final de la simulación. Por el contrario, el desempeño NCAT resultó en un escaso 4 % (735 de 20000 moléculas). Este número concurre con lo esperado, dado que solo una tercera parte de la concentración se aplica en cada una de las tres cascadas. De esta forma, la salida final se ve atenuada por un factor de aproximadamente 1/27 con respecto a la concentración de la consulta inicial.

4.5. Validación termodinámica con NUPACK

En esta sección se describen los resultados obtenidos en la validación termodinámica de las estructuras propuestas. En primer término, se desarrolló un procedimiento para determinar conjuntos de secuencia que resultaran adecuadas — a nivel de complejo — para la correcta formación de las implicaciones. Estos conjuntos fueron posteriormente probados — a nivel de tubo — para verificar que las concentraciones

en equilibrio correspondieran a la molécula objetivo. El Algoritmo 1 describe un procedimiento iterativo de muestreo y prueba que recibe una lista de implicaciones como entrada y devuelve los dominios representativos involucrados. Los parámetros de búsqueda en el algoritmo, las funciones NUPACK y condiciones de búfer utilizados fueron descritos en la Sección 3.1.3.1.

Algorithm 1 Búsqueda de secuencias (MS, MHD, MCS, CC)

Require: Lista $\mathcal{R} = \{R_1, R_2, ..., R_m\}$ de moléculas implicación.
Ensure: Lista $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ de dominios para $\mathcal{R}$

```
1: for each R_i ∈ R do
2:     bestSample = ∅
3:     for j = 1 to MS do
4:         S_j ← Generar_secuencias(R_i, MHD, MSS, CC)
5:         Compute function f(S_j)
6:         if f(S_j) es mejor que f(bestSample) then
7:             bestSample ← S_j
8:     D = D ∪ bestSample
```

El ciclo principal del Algoritmo 1 (líneas 1-8) procesa una lista $\mathcal{R}$ de implicaciones pertenecientes a un programa lógico. El ciclo interior (líneas 3-7) itera sobre muestras S_j que se componen de secuencias candidatas para la implicación actual R_i. En la línea 4, se llama a una función para generar S_j de conformidad con las restricciones planteadas por Qian y Winfree (2011). Estas restricciones consisten en: usar los parámetros MHD, MCS y CC; excluir la letra G del alfabeto y limitar la repetición de letras (hasta cuatro A's, cuatro T's y tres C's). En la siguiente línea (5) se llama a una función NUPACK para evaluar la muestra actual. En caso de que el valor regresado represente una mejora respecto a la mejor muestra encontrada al momento, esta se actualiza (líneas 6-7). Finalmente, la mejor muestra encontrada en las MS iteraciones se registra en la lista definitiva a devolver $\mathcal{D}$.

Como se mencionó en la Sección 3.1.3.1, el Algoritmo 1 se ejecutó para nueve configuraciones experimentales (tres opciones de longitud de dominios en combinación con tres funciones de evaluación NUPACK). Cada configuración arrojó un conjunto de secuencias resultado para cada uno de los casos de prueba E1 y E2 descritos en la Sección 3.1.2.1. Cada uno de estos conjuntos fue evaluado con la herramienta en línea NUPACK para determinar concentraciones en equilibrio. La configuración que produjo mejores ensamblajes fue con los dominios de longitud $(20, 20)$ junto con la función de

probabilidad de equilibrio. El resto, arrojó ya sea moléculas no deseadas o implicaciones incorrectas. Las Tablas 4 y 5 muestran las secuencias obtenidas para los ejemplos E1 y E2, respectivamente. La estructura secundaria de cada implicación así como su energía libre de Gibbs y probabilidad de equilibrio se incluyen en el Anexo B (Figuras 57−62 para E1 y 63−65 para E2). Las formaciones marginales de moléculas basura que se produjeron a $T = 37°C$, con mismas secuencias de la Tabla 4 se muestran en la Figura 66.

Tabla 4. Mejores secuencias obtenidas para cada dominio del programa E1. Las secuencias de los dominios t_1, t_2, and GRC fueron constantes.

Dominio	Secuencia	Proposición
t_1	CCTATT	None
t_2	CATCG	None
GRC	CCCTTACCCAACCCAATCCC	None
D_0	TTTTCATTCACTACCTCCAA	p: Ejército(X)
D_1	CCACAACAATCAATCAAAAC	q: Fuerte(X)
D_2	TTTCCAACACACACTCAAAT	r: Academia(X)
D_3	CCCTACTAATTTAACTAACC	s: Sabio(X)
D_4	TCTTTTCTCTCTTCTCTCTA	t: X=Sócrates
D_5	ATCTTCTTCCATTCACTCTT	u: X=Alejandro
D_6	TTTTCTTTCTCCTCTCCTTA	v: X=Hércules

Tabla 5. Mejores secuencias obtenidas para cada dominio del programa E2. Las secuencias de los dominios t_1, t_2, and GRC fueron constantes.

Dominio	Secuencia	Proposición
t_1	CCTATT	None
t_2	CATCG	None
GRC	CCCTTACCCAACCCAATCCC	None
D_0	TTTCTCCATCACATCCAAAT	w: X=Maslow
D_1	CTTCCTTCCTTTTATCTTCA	x: Enamorado(X)
D_2	CCTTAATTCCTACAACTAAC	y: Adinerado(X)
D_3	TTTTCTCTTTACCTCACTCA	z: Feliz(X)

4.6. Simulación con reacciones de interferencia

A tenor del esquema de codificación aquí propuesto, inherentemente contiene dos interacciones que potencialmente pueden influir sobre el desempeño del sistema. En

esta parte, se describen las reacciones, se estima su propensión mediante simulaciones a nivel de nucleótido y se estudia su efecto en el contexto de programas lógicos con el mecanismo de encadenamiento hacia atrás[1]

4.6.1. Descripción de las reacciones de interferencia

4.6.1.1. Interferencia Implicación-Implicación (𝕀𝕀)

La reacción 𝕀𝕀 puede ocurrir entre una implicación invasora $\mathbb{I}_{p\to\alpha}$ y una implicación receptora $\mathbb{I}_{\beta\to p}$, donde α y β pueden ser cualquier proposición. La Figura 32 muestra los pasos de esta interacción. En el Panel B, el segmento colgante que contiene el toehold t_1 activo, se une al toehold complementario de la implicación receptora. Entonces, dado que los dominios representativos coinciden, el proceso de migración de ramas de 3 vías inicia, desplazando la hebra de cobertura derecha de la implicación receptora (Panel C). Es importante notar que en este punto, la reacción se vuelve irreversible, dado que existen suficientes pares de bases entre las moléculas para que ocurra una desunión espontánea. Así, la reacción continúa ahora con un paso de migración de ramas de 4 vías en el dominio GRC. Al término de este paso, se produce el intercambio de las hebras que representan las consultas de salida $\mathbb{Q}_p$ y $\mathbb{Q}_\beta$ de ambas implicaciones. Al final del intercambio (Panel D), las moléculas se separan quedando como moléculas subproducto: una molécula con la estructura de implicación $\mathbb{I}_{\beta\to\alpha}$ y un complejo que continúa reaccionando intramolecularmente para finalmente separarse entre la hebra de cobertura izquierda (desperdicio) y el complejo reactivo $\mathbb{C}_p$.

Un aspecto interesante radica en el hecho de que esta reacción colateral, se puede interpretar como una operación de inferencia lógica bien conocida llamada *silogismo hipotético*. En efecto, la implicación producida $\mathbb{I}_{\beta\to\alpha}$ es una consecuencia lógica de la BC a partir de las implicaciones de entrada. No obstante, en el caso del otro complejo $\mathbb{C}_p$ resultante de la interferencia, este es el mismo que se produce de una reacción $\mathbb{Q}\mathbb{I}$ entre $\mathbb{Q}_p$ y cualquier implicación de la forma $\mathbb{I}_{\beta\to p}$. Como consecuencia, un potencial problema radica en que si $\mathbb{Q}_p$ no se libera en ninguna otra ruta de inferencia del sistema, podría reducir el rendimiento final en caso de una inferencia positiva, o incluso provocar falsos positivos.

[1] Un análisis similar se aplicaría para encadenamiento hacia adelante, de manera que se omite aquí por brevedad.

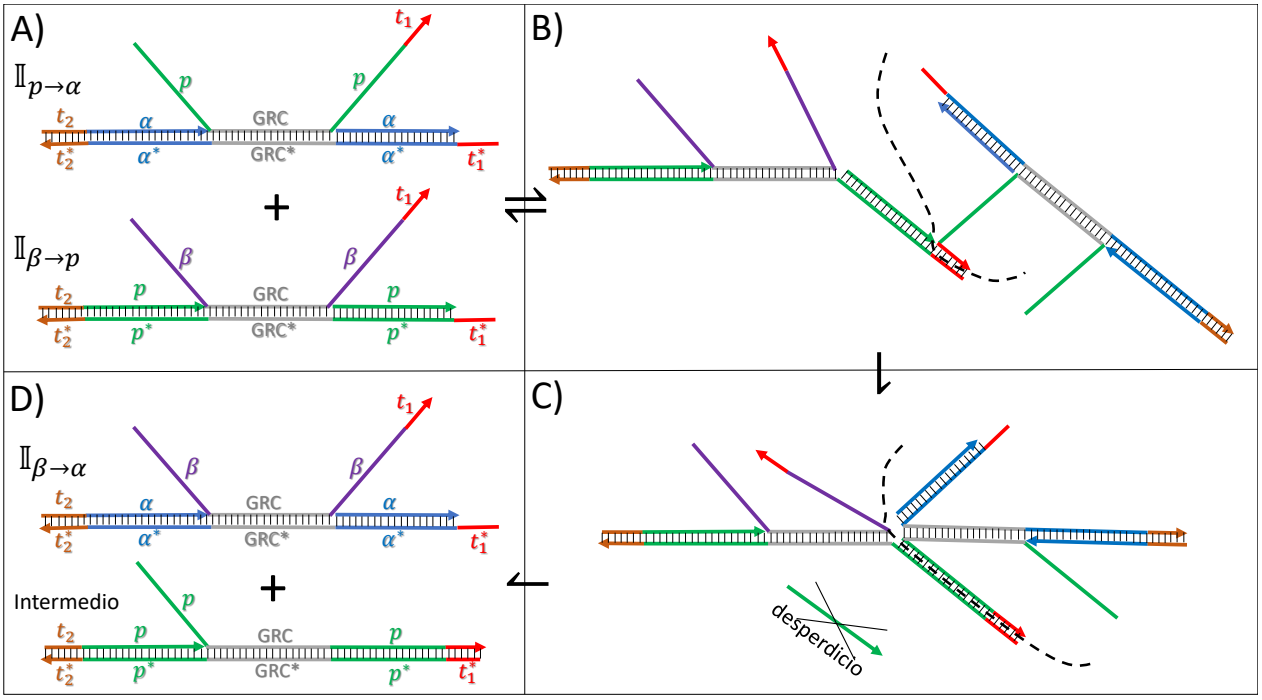

Figura 32. Pasos de la reacción de interferencia 𝕀. Las líneas punteadas separan la implicación invasora de la receptora. A) Moléculas implicación involucradas $\mathbb{I}_{p\to\alpha}$ y $\mathbb{I}_{\beta\to p}$. B) Paso de unión de toeholds. C) Estado al término del paso de migracion de ramas de 3 vías que desplaza la hebra de cobertura derecha, justo antes de iniciar la migración de 4 vías. D) Moléculas resultantes al término del paso de migración de ramas de 4 vías: una nueva implicación $\mathbb{I}_{\beta\to\alpha}$ y una molécula intermedia que eventualmente producirá un complejo reactivo $\mathbb{C}_p$ (no mostrado en la imagen).

4.6.1.2. Interferencia Implicación-Proposición (𝕀ℙ)

Este tipo de reacción (𝕀ℙ) toma lugar entre una implicación invasora $\mathbb{I}_{p\to\alpha}$ y una proposición receptora $\mathbb{P}_p$ (Figura 33). Procede de manera similar que la reacción de cruce 𝕀𝕀 en el paso de unión de toeholds (Panel B). En este caso, sin embargo, los pasos de migración de ramas de 3 y 4 vías son reversible (Paneles C y D), lo cual significa que la reacción global puede regresar al estado inicial en cualquier etapa antes del final. Al término de la migración de ramas del segmento [p—GRC—p], la hebra con el fluoróforo añadido de la detectora se intercambia con la consulta de salida $\mathbb{Q}_p$ de la implicación. Como producto se obtienen una hebra dsDNA inerte (desperdicio) y un complejo emisor de fluorescencia (Panel E). Esta fluorescencia consistirá en una respuesta errónea (falso positivo) si $\mathbb{Q}_p$ no es derivada lógicamente en ninguna otra parte del programa. En caso de que sí lo fuera, la respuesta se integraría a la fluorescencia ya producida por una ruta de inferencia válida.

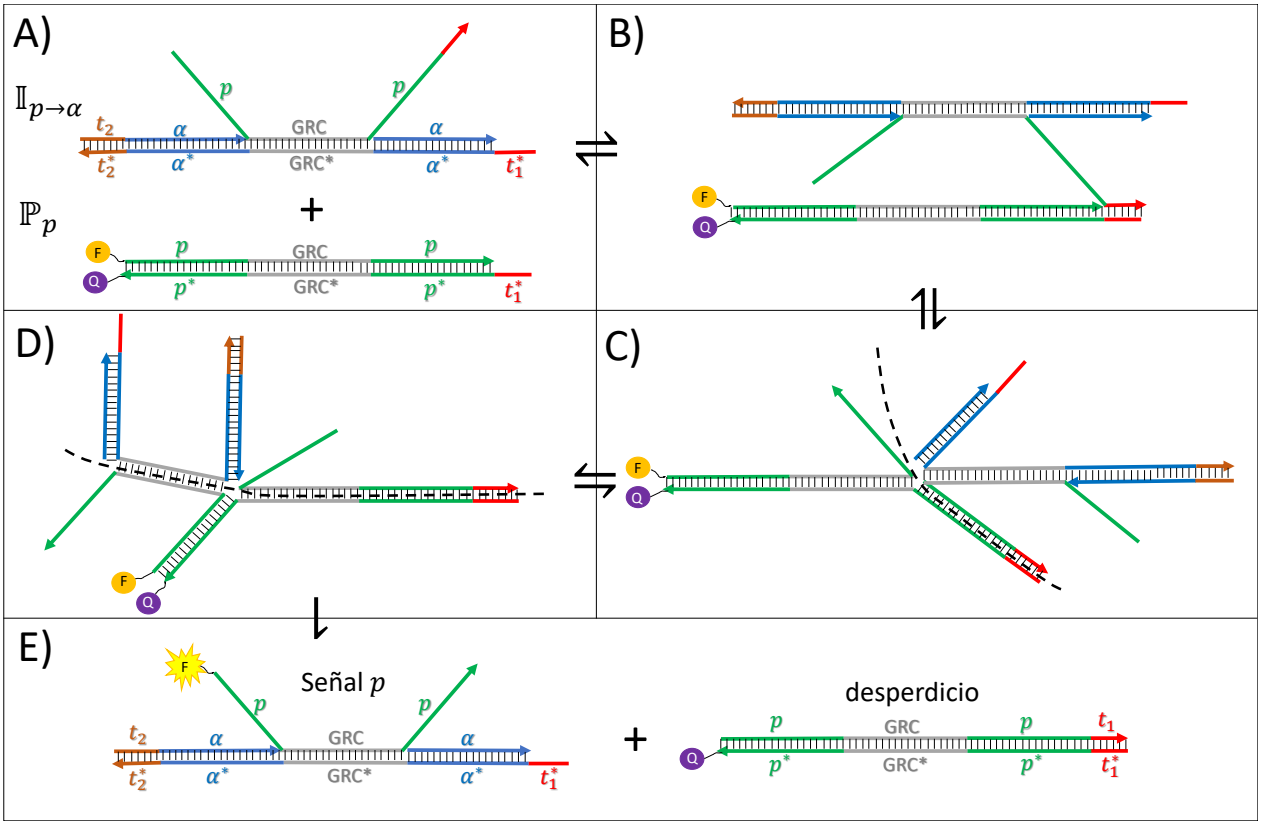

Figura 33. Pasos de la reacción de interferencia $\mathbb{IP}$. Las líneas punteadas separan la implicación invasora de la proposición receptora. A) Moléculas involucradas, implicación $\mathbb{I}_{p\to\alpha}$ y proposición $\mathbb{P}_p$. B) Paso de unión de toeholds. C) Estado al término del paso de migracion de ramas de 3 vías del dominio p. D) Estado al término del paso de migración de ramas de 4 vías. E) Moléculas resultantes: un complejo dsDNA desperdicio y una molécula fluorescente.

4.6.2. Estimación de la propensión en MULTISTRAND

A partir de las simulaciones realizadas en MULTISTRAND, la Tabla 6 presenta las cantidades de trayectorias exitosas (n_{exit}) y fallidas (n_{fall}), así como sus respectivos tiempos promedio de finalización: E(Δt_{exit}) y E(Δt_{fall}) (segundos). Las reacciones $\mathbb{QI}$ tuvieron una notable mayor propensión de finalizar en estado exitoso que el resto de las reacciones, aunque el tiempo promedio fue un orden de magnitud más lento que $\mathbb{QP}$. Otro resultado importante es que ninguna de las trayectorias para $\mathbb{IP}$ alcanzó una terminación exitosa, lo que proporciona indicios de su baja propensión. Por otro lado, al comparar $\mathbb{QI}$ contra $\mathbb{II}$, puede observarse que ambas obtuvieron probabilidades similares de completarse. Sin embargo, el tiempo esperado para la reacción no deseada fue cuatro órdenes de magnitud mayor que la diseñada.

Tabla 6. Resultados de simulación de trayectorias para reacciones de un paso.

Reacción	n	$n_{\mathbf{exit}}$	$n_{\mathbf{fall}}$	**E($\Delta t_{\mathbf{exit}}$) (s)**	**E($\Delta t_{\mathbf{fall}}$) (s)**
ℚℙ	1000	239	761	2.24e-4	5.65e-5
ℚ𝕀	1000	33	967	3.55e-5	2.41e-6
𝕀ℙ	1000	0	1000	NA	4.02e-3
𝕀𝕀	1000	44	956	1.61e-1	8.37e-5

Con respecto a la simulación dividida por pasos, los resultados se muestran en la Tabla 7. En este escenario se pudieron ejecutar una mayor cantidad de trayectorias, a excepción del paso de migración de 4 vías en 𝕀𝕀. El paso de unión de toeholds tuvo notoriamente más casos exitosos para las reacciones que involucran la molécula proposición (ℚℙ e 𝕀ℙ), que para las que tienen a la implicación como receptora (ℚ𝕀 e 𝕀𝕀). El segundo paso, tuvo tanto casos exitosos como tiempos promedio similares para las cuatro reacciones. El tercer paso es que resulta de mayor interés en el análisis, pues es el que corresponde a una migración de ramas de 3 vías en las reacciones programadas y de 4 vías para las no programadas. Para las primeras, también finalizaron exitosamente la gran mayoría de trayectorias. En contraste, para 𝕀ℙ solo una de las 2400 trayectorias finalizó, y lo hizo con dos órdenes de magnitud más tiempo que ℚℙ y ℚ𝕀. La reacción 𝕀𝕀, en cambio, terminó todas sus trayectorias. No obstante, su tiempo promedio fue cuatro órdenes de magnitud mayor que las reacciones válidas. En general, estos resultados sugieren que el paso de migración de ramas de 4 vías, introduce una especie de barrera cinética a las reacciones de interferencia, particularmente para 𝕀ℙ. Además, la relación de los tiempos promedio entre migración de ramas de 3 y 4 vías, son consistentes para lo reportado por Panyutin y Hsieh (1994). Por lo tanto, se podría inferir una respuesta mucho más rápida para las reacciones previstas del modelo, que para las no planeadas.

4.6.3. Simulación masa-acción en DIZZY

En esta parte, nos interesamos en estimar el efecto de las dos reacciones no deseadas en el contexto de programas lógicos específicos. Para esto, se consideró de particular interés el programa E1, dado que presenta varias reacciones de este tipo. Usando las formas abreviadas de proposiciones de la Tabla 4, las interacciones 𝕀𝕀

Tabla 7. Resultados de simulación de trayectorias para reacciones a nivel de dominios. (*) Debido al costo computacional, se tuvo que simular menos trayectorias en este paso.

Paso	n	$n_{\mathbf{exit}}$	$n_{\mathbf{fall}}$	**E($\Delta t_{\mathbf{exit}}$) (s)**	**E($\Delta t_{\mathbf{fall}}$) (s)**
Reacción $\mathbb{QP}$					
Unión toeholds	2400	578	1822	3.94e-5	5.05e-5
MR de 3 vías	2400	2388	12	2.43e-5	8.87e-5
MR de 3 vías	2400	2390	10	6.59e-5	1.06e-4
MR de 3 vías	2400	2389	11	8.25e-5	2.01e-4
Reacción $\mathbb{QI}$					
Unión toeholds	2400	103	2297	9.10e-5	8.51e-6
MR de 3 vías	2400	2395	5	2.16e-5	4.53e-5
MR de 3 vías	2400	2400	0	2.00e-5	–
Reacción $\mathbb{IP}$					
Unión toeholds	2400	669	1731	3.69e-5	6.82e-5
MR de 3 vías	2400	2395	5	2.33e-5	9.54e-5
MR de 4 vías	2400	1	2399	2.93e-3	1.65e-2
MR de 3 vías	2400	2400	0	2.00e-5	–
Reacción $\mathbb{II}$					
Unión toeholds	2400	113	2287	4.05e-5	9.80e-5
MR de 3 vías	2400	2393	7	2.13e-5	4.49e-5
MR de 4 vías	$600^{(*)}$	600	0	2.08e-1	–

son: $\mathbb{I}_{q\to p}+\mathbb{I}_{u\to q}$, $\mathbb{I}_{q\to p}+\mathbb{I}_{v\to q}$, $\mathbb{I}_{s\to r}+\mathbb{I}_{t\to s}$, and $\mathbb{I}_{s\to r}+\mathbb{I}_{u\to s}$; mientras que las reacciones $\mathbb{IP}$ están dadas por: $\mathbb{I}_{t\to s}+\mathbb{P}_t$, $\mathbb{I}_{u\to s}+\mathbb{P}_u$, $\mathbb{I}_{u\to q}+\mathbb{P}_u$, y $\mathbb{I}_{v\to q}+\mathbb{P}_v$. Estas reacciones se agregan al mismo modelo cinético de este programa que se obtuvo a partir de la compilación en Visual DSD (Sección 4.4). Los resultados promedio de 20 simulaciones independientes se presentan en las Figuras 34, 35, and 36; cada una correspondiendo a una concentración base de x ={5, 10, 20} nM, respectivamente. Estas concentraciones base representan el máximo de concentración alcanzable para las señales de salida en cada caso. Las concentraciones iniciales se establecieron en $1x$ para todos los hechos y la consulta inicial (es decir, para $\mathbb{I}_{t\to s}$, $\mathbb{I}_{u\to s}$, $\mathbb{I}_{u\to q}$, $\mathbb{I}_{v\to q}$, and $\mathbb{Q}_p$); mientras que en $2x$ para las reglas generales ($\mathbb{I}_{q\to p}$ and $\mathbb{I}_{s\to r}$). Para analizar el efecto de las moléculas combustible ($\mathbb{F}_p$, $\mathbb{F}_q$, $\mathbb{F}_r$, and $\mathbb{F}_s$), se simularon varios escenarios con concentraciones de 0.0, 0.4, 0.8, 1.2, 1.6, 2.0 y 4.0 nM. Nótese que el escenario con 0.0 nM de combustible, corresponde a la versión NCAT del programa.

En los tres casos de concentraciones base, se puede apreciar el compromiso que se crea al incrementar el uso de moléculas combustible: mayor concentración mejora la respuesta de las señales de inferencia positiva (X=Alejandro y X=Hércules); pero al

mismo tiempo, empeora la señal de inferencia negativa (X=Sócrates), pues también se incrementa. Esto se debe al efecto de los combustibles en las molécula subproducto que se generan en las interferencias $\mathbb{I}\mathbb{I}$, como se discutió anteriormente (Sección 4.6.1.1). La respuesta al incremento de combustible es drástica de cero a $0.4x$ nM, causando un salto en la señal de falso positivo de aproximadamente 7.0, 4.8 y 4.6 %; para cada concentración base $x = \{5, 10, 20\}$ nM, respectivamente. Sin embargo, este impacto parece atenuarse gradualmente para sucesivas concentraciones de combustible. Por ejemplo, nótese como el salto provocado por aumentar el combustible de $0.4x$ a $0.8x$ es mayor al del de doblar la concentración de $2.0x$ a $4.0x$ en las tres figuras. También es interesante notar como la señal de inferencia negativa no se ve significativamente influida por la interferencia $\mathbb{I}\mathbb{P}$ sola (observe las curvas del escenario NCAT en todas las figuras). Esto se debe a la baja propensión de la reacción de cruce.

Por otro lado, de acuerdo a la ley de masa-acción, un incremento de la concentración base produciría un mejor desempeño general dentro de los tiempos de reacción fijados. Este efecto significa que en regímenes de bajas concentraciones, se debe extender los tiempos de reacción para que las salidas se estabilicen alcanzando concentraciones aceptables. Las Figuras 82 y 83 (Anexo E) muestran este comportamiento en bajas concentraciones de 500 y 50 pM, respectivamente.

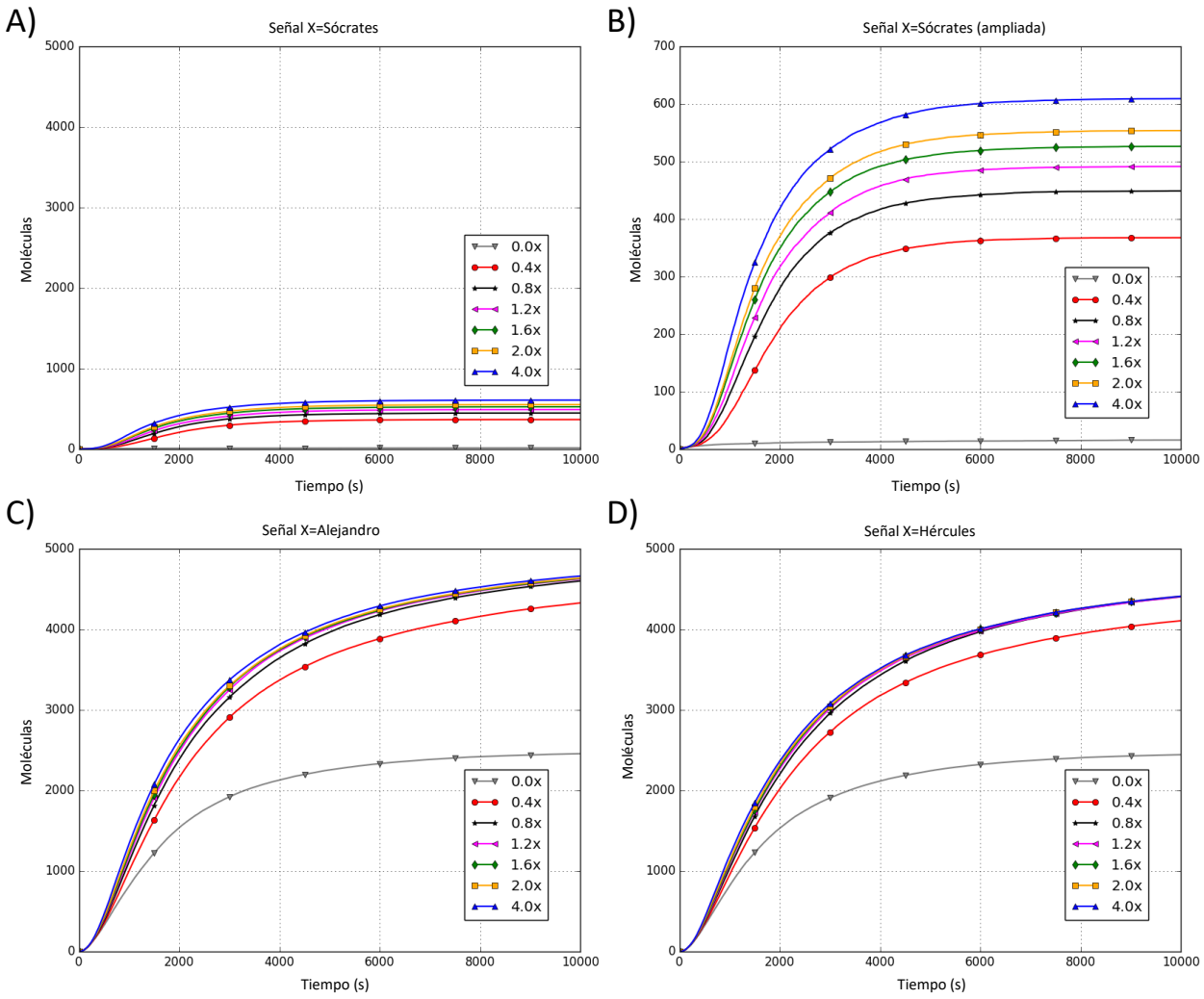

Figura 34. Efecto de las reacciones de cruce en el programa E1 a diferentes concentraciones de combustibles. La consulta inicial es "*Ejército*(X)?". A) Señal de salida X=Sócrates (inferencia negativa). B) Vista ampliada del Panel A. C) Señal de salida X=Alejandro (inferencia positiva). D) Señal de salida X=Hércules (inferencia positiva). En estas simulaciones la concentración base es $x = 5$nM (5000 moléculas).

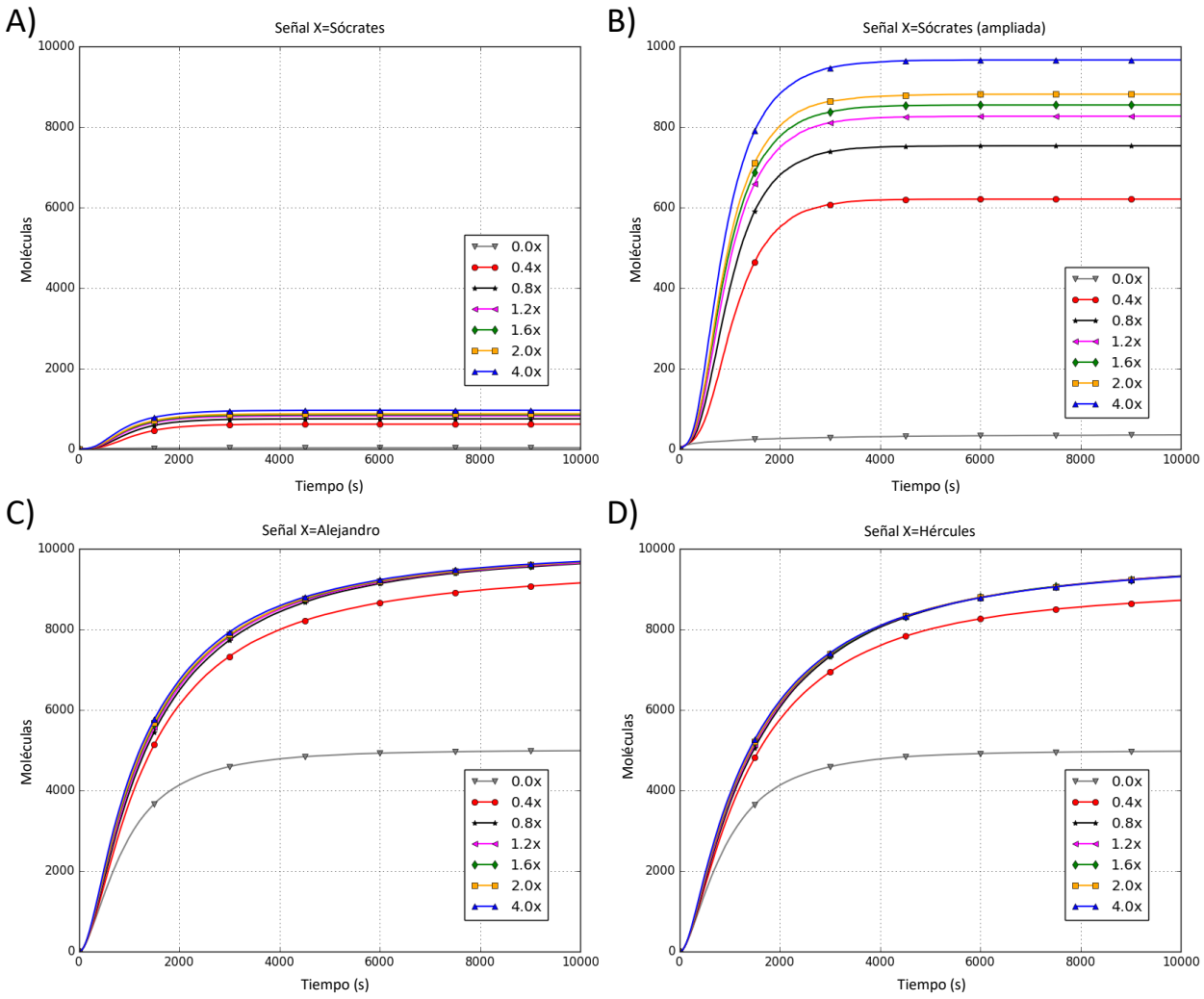

Figura 35. Efecto de las reacciones de cruce en el programa E1 a diferentes concentraciones de combustibles. La consulta inicial es "*Ejército*(X)?". A) Señal de salida X=Sócrates (inferencia negativa). B) Vista ampliada del Panel A. C) Señal de salida X=Alejandro (inferencia positiva). D) Señal de salida X=Hércules (inferencia positiva). En estas simulaciones la concentración base es $x = 10$nM (10000 moléculas).

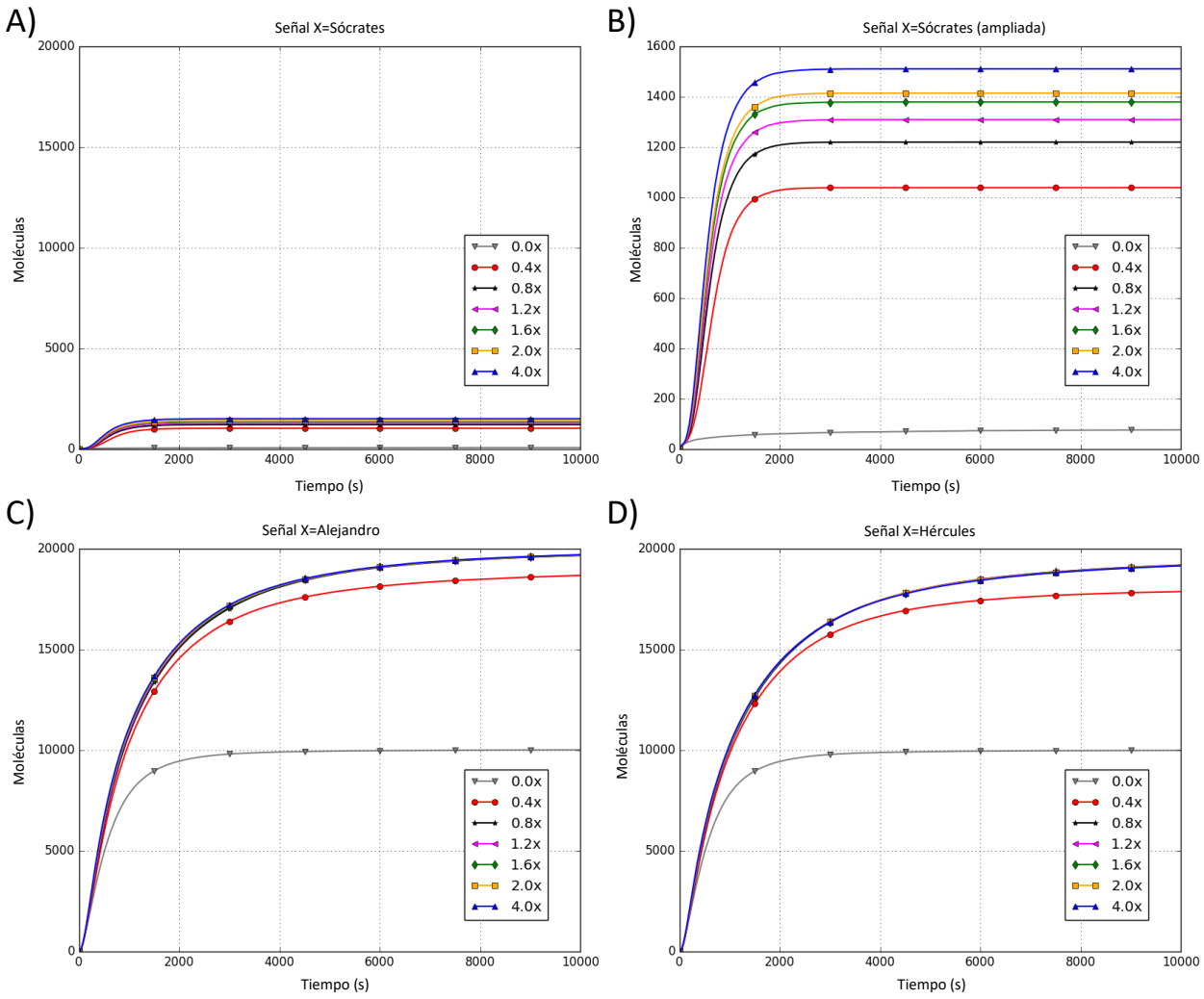

Figura 36. Efecto de las reacciones de cruce en el programa E1 a diferentes concentraciones de combustibles. La consulta inicial es "*Ejército*(X)?". A) Señal de salida X=Sócrates (inferencia negativa). B) Vista ampliada del Panel A. C) Señal de salida X=Alejandro (inferencia positiva). D) Señal de salida X=Hércules (inferencia positiva). En estas simulaciones la concentración base es $x = 20$nM (20000 moléculas).

Capítulo 5. Resultados y discusión: Representación de lógica de doble riel basada en toeholds

En este capítulo, se introduce la codificación de doble toehold en correspondencia a la lógica de doble carril. La adaptación aquí introducida permite implementar circuitos lógicos con la mitad de los dominios representativos. Además de esta ventaja obvia en cuanto a la escalabilidad, se demuestra que el desempeño de los circuitos se mejora, al establecerse una especie de segmentación en la que las reacciones espurias se dividen. Se presentan resultados de simulación de una compuerta, de un circuito que calcula la raíz cuadrada de un número de cuatro bits y un circuito escalado con cinco copias de este último reaccionando en paralelo.

5.1. Codificación básica doble toehold

Basados en la codificación de las compuertas sube-baja (Sección 1.5.2.1) y tomando inspiración en el trabajo de tesis de Velázquez Sánchez (2014), se propone una modificación simple que consiste en usar dos toeholds, uno para manejar los “unos” (T_1) y otro para los “ceros” lógicos (T_0).

La Figura 37 ilustra este cambio en el esquema sube-baja, para la implementación de lógica de doble carril. Para cada tipo de molécula del modelo original de Qian y Winfree (2009), se muestran las correspondientes posibilidades para el esquema de doble toehold.

Como se puede observar, una molécula cable $w_{1,5}$ tiene dos representaciones, con los mismos dominios de reconocimiento pero con distinta secuencia de toehold, una para la salida en estado “1” ($w^1_{1,5}$) y otra para el estado “0” ($w^0_{1,5}$). Lo mismo ocurre para las moléculas umbral $T_{5,7:7}$ y reportera Rep_7. En el caso de las moléculas compuerta (i.e. $G_{5:5,7}$, se tienen cuatro posibles representaciones para las combinaciones de T_1 y T_0 de los toeholds en los extremos (T_1T_1, T_0T_0, T_0T_1 y T_1T_0). Note que los superíndices en la notación de cada molécula indican los toeholds involucrados.

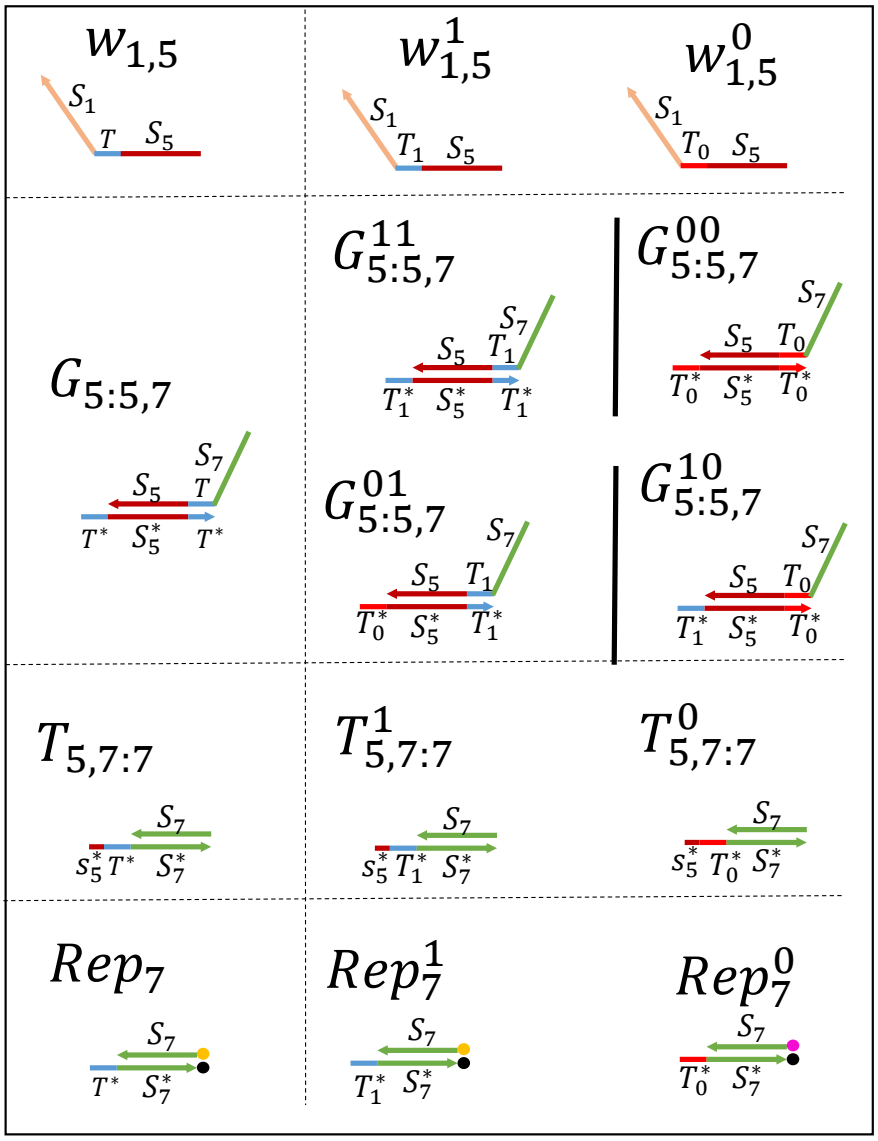

Figura 37. Codificación utilizando doble toehold. En el lado izquierdo se muestran los bloques de construcción originales de Qian y Winfree (2009) y del lado derecho el esquema de doble toehold.

5.2. Implementación de compuertas de doble carril

A partir de las definiciones de los bloques de construcción en el esquema de doble toehold, las reacciones para implementar las cuatro compuertas en lógica de doble carril proceden de manera idéntica que el modelo original.

La Figura 38 ilustra la configuración para realizar la operación lógica NAND doble carril, con esta codificación de doble toehold, agregando las compuertas umbral $T^1_{5,7:7}$ y $T^0_{5,7:7}$ a concentraciones de $0.6x$ y $1.2x$, respectivamente. Note que el resto de las

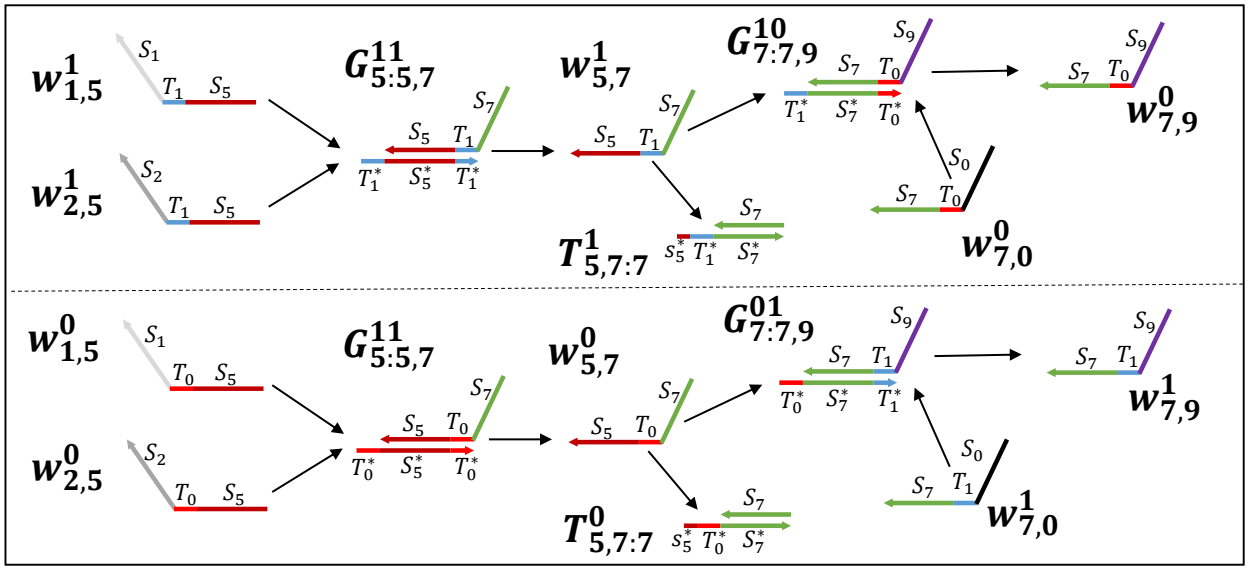

Figura 38. Secuencia de reacciones para la compuerta NAND en doble toehold. Panel superior: operación $y_0 = x^1_1$ AND x^1_2. Panel inferior: operación $y_0 = x^0_1$ OR x^0_2. Dominios utilizados: $\{S_0, S_1, S_2, S_5, S_7, S_9\}$

compuertas básicas pueden implementarse haciendo los siguientes cambios, en correspondencia con la Figura 22:

- Para la compuerta AND doble carril, se requiere intercambiar las salidas y_0 y y_1 respecto a la NAND doble carril. Esto es, reemplazar las compuertas amplificadoras $G^{10}_{7:7,9}$ por $G^{11}_{7:7,9}$ y $G^{01}_{7:7,9}$ por $G^{00}_{7:7,9}$.
- Para la compuerta OR doble carril, es necesario intercambiar las compuertas AND y OR respecto a la AND doble carril. Es decir, intercambiar las concentraciones de las moléculas umbral.
- Para la compuerta NOR de doble carril, basta intercambiar las salidas y_0 y y_1, respecto a la OR doble carril. Esto es, reemplazar las compuertas amplificadoras $G^{00}_{7:7,9}$ por $G^{01}_{7:7,9}$ y $G^{11}_{7:7,9}$ por $G^{10}_{7:7,9}$.

Una ventaja que se puede observar de inmediato con este esquema, es la reducción en las secuencias necesarias, a la mitad de aquellas del modelo original (ver Figura 39). Otra observación importante, es que la operación molecular no se altera, respecto a la versión original ya experimentalmente demostrada, por lo que se espera que el esquema modificado funcione al menos con la misma eficiencia. Adicionalmente, la ventaja principal en la idea propuesta, es que se establece una segmentación en los

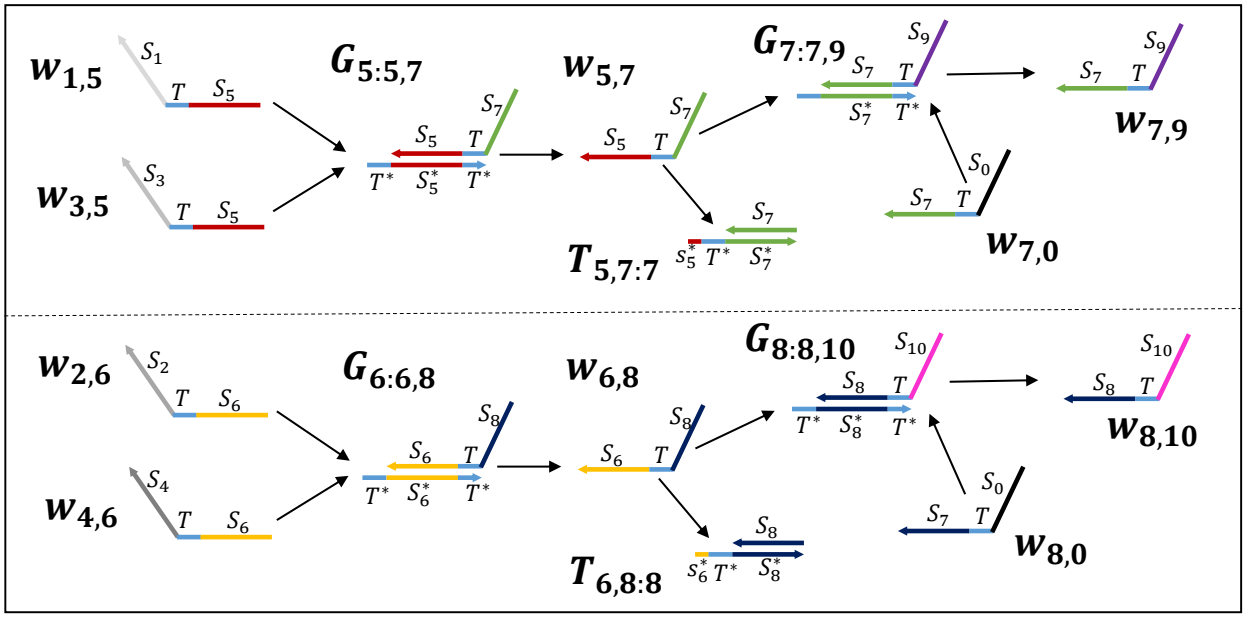

Figura 39. Secuencia de reacciones para la compuerta NAND en el esquema original de Qian y Winfree (2011). Panel superior: operación $y_0 = x_1^1$ AND x_2^1. Panel inferior: operación $y_0 = x_1^0$ OR x_2^0. Dominios utilizados: $\{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}\}$

circuitos, de manera que se espera que se reduzcan a la mitad las reacciones espurias (reacciones improductivas en las que una molécula cable, se une por el toehold a moléculas compuerta, umbral o reporteras, sin compartir el dominio de reconocimiento; ver Figura 40).

Al consumir un tiempo manteniendo moléculas en un estado de no disponibilidad, este tipo de reacciones afecta el desempeño de los circuitos. Este efecto es particularmente perjudicial en el caso de las moléculas umbral, dado que la abstracción digital depende en buena medida de las concentraciones adecuadas, y por ende, de la disponibilidad de estas moléculas. Además, la extensión del toehold en estas compuertas puede causar que algunos nucleótidos del dominio de reconocimiento izquierdo de la molécula cable coincidan, haciendo más lenta su liberación o incluso provocando que la reacción sea irreversible (ver Figura 41). En las siguientes secciones, se verificará el efecto de estas reacciones en ambos diseños, mediante simulaciones cinéticas.

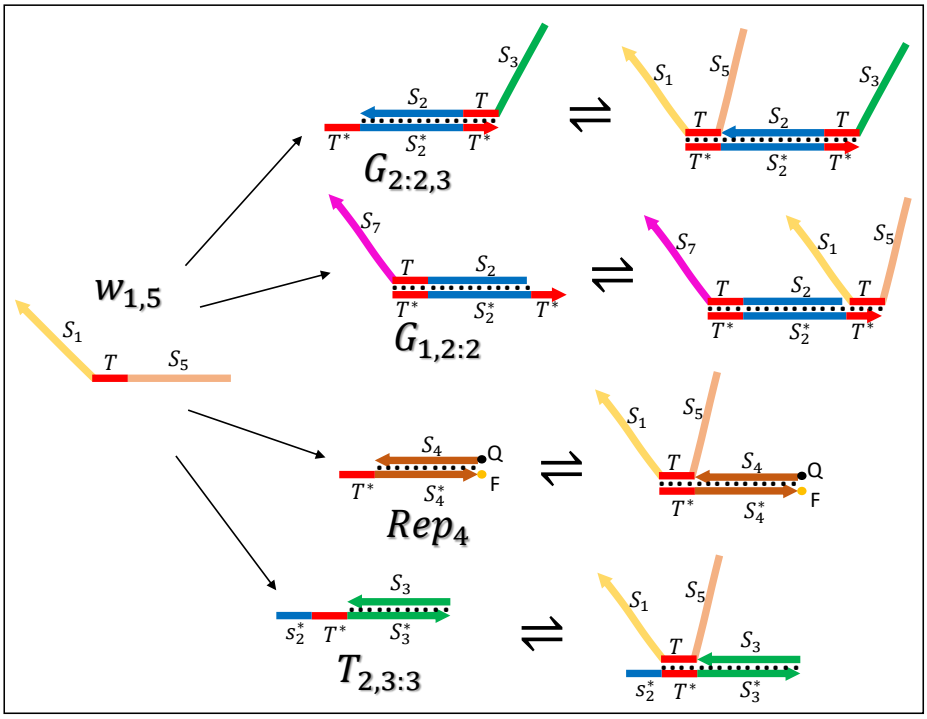

Figura 40. Los cuatro tipos de reacciones no productivas de una molécula cable ($w_{1,5}$) con: compuerta con toehold izquierdo ($G_{2:2,3}$), compuerta con toehold derecho ($G_{1,2:2}$), reportera (Rep_4) y umbral ($T_{2,3:3}$).

5.3. Simulación de compuertas básicas

En las Figuras 42 y 43 se muestran los resultados de simulación de una compuerta NOR de doble carril en ambos esquemas: el original de Qian y Winfree (QW) y el modificado con doble toehold (dToe).

Se puede apreciar el contraste de ambos escenarios de simulación. En primer lugar, la respuesta en el primero es más lenta, lo cual es natural, dado que las reacciones generales tardan más al ralentizar el paso hibridación de los toeholds y agilizar su desunión. Además, esta simulación en el contexto de rápida desunión de toeholds, minimiza las fugas para las salidas en estado lógico “0”. Dado que las moléculas cable duran menos tiempo unidas a los distintos complejos, el efecto de las reacciones no productivas también se minimiza. Como resultado, las señales de salida en ambos esquemas son muy similares, aunque ligeramente con mejor eficiencia en los estados “1” para el sube-baja original.

En el escenario de simulación con constantes normales, en cambio, las salidas requieren menos tiempo para alcanzar niveles de saturación. Las salidas en estado “1”

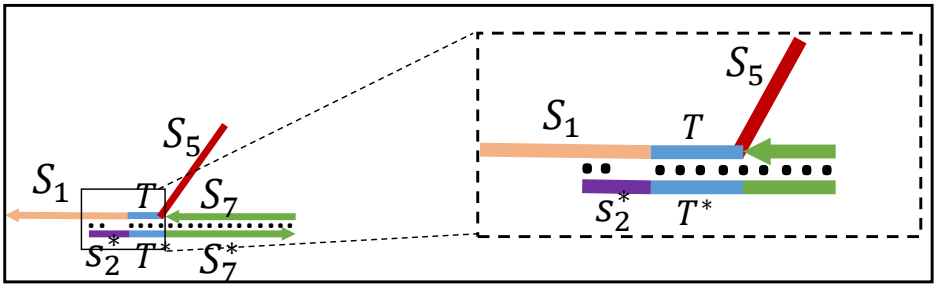

Figura 41. Reacciones no productivas en compuertas umbral. El toehold extendido (s_2^*, en este ejemplo) podría compartir algunos nucleótidos con el dominio de reconocimiento (S_1, en este caso) de la molécula cable, lo que retardaría la desunión para liberar las moléculas.

muestran un ligero pero observable mejor comportamiento en el esquema de doble toehold. No obstante, las salidas en estado "0" también se elevan más que su contraparte, lo cual es indeseable. Para ambos esquemas, estas salidas tienen considerablemente mayores fugas, particularmente para las entradas 01 y 10, dado que la configuración de constantes le confiere mayor propensión también a las reacciones espurias. La razón de que en estos casos sean mayores es que la salida y_1 corresponde a la operación AND entre las entradas x_1^0 y x_2^0. Si consideramos que estas entradas suman una concentración aproximada de $1.0x$, la cual es muy cercana a la concentración del umbral ($1.2x$, para la función AND), aunado además a la condición de que las reacciones improductivas mantienen una parte de las moléculas umbral en estado de indisponibilidad, resulta evidente que una mayor parte de la suma reaccione con la compuerta amplificadora, elevando la concentración de salida final y_1.

5.4. Simulación de circuitos lógicos de mayor escala

Como caso de referencia se seleccionó el circuito que calcula $\lfloor \sqrt{n} \rfloor$, donde n es un número de cuatro bits $x_4x_3x_2x_1$ (Qian y Winfree, 2011). Tal circuito es, a la fecha, el de mayor escala implementado usando reacciones de desplazamiento de hebras en laboratorio. La Figura 23 muestra dicho circuito en lógica AND-OR-NOT y su correspondiente representación con lógica de doble carril.

La Figura 84 (Anexo F) ilustra la representación abstracta del circuito tal como fue introducida en Qian y Winfree (2009). Tomando la misma idea para los circuitos modificados con doble toehold, la Figura 44 muestra su representación abstracta equivalen-

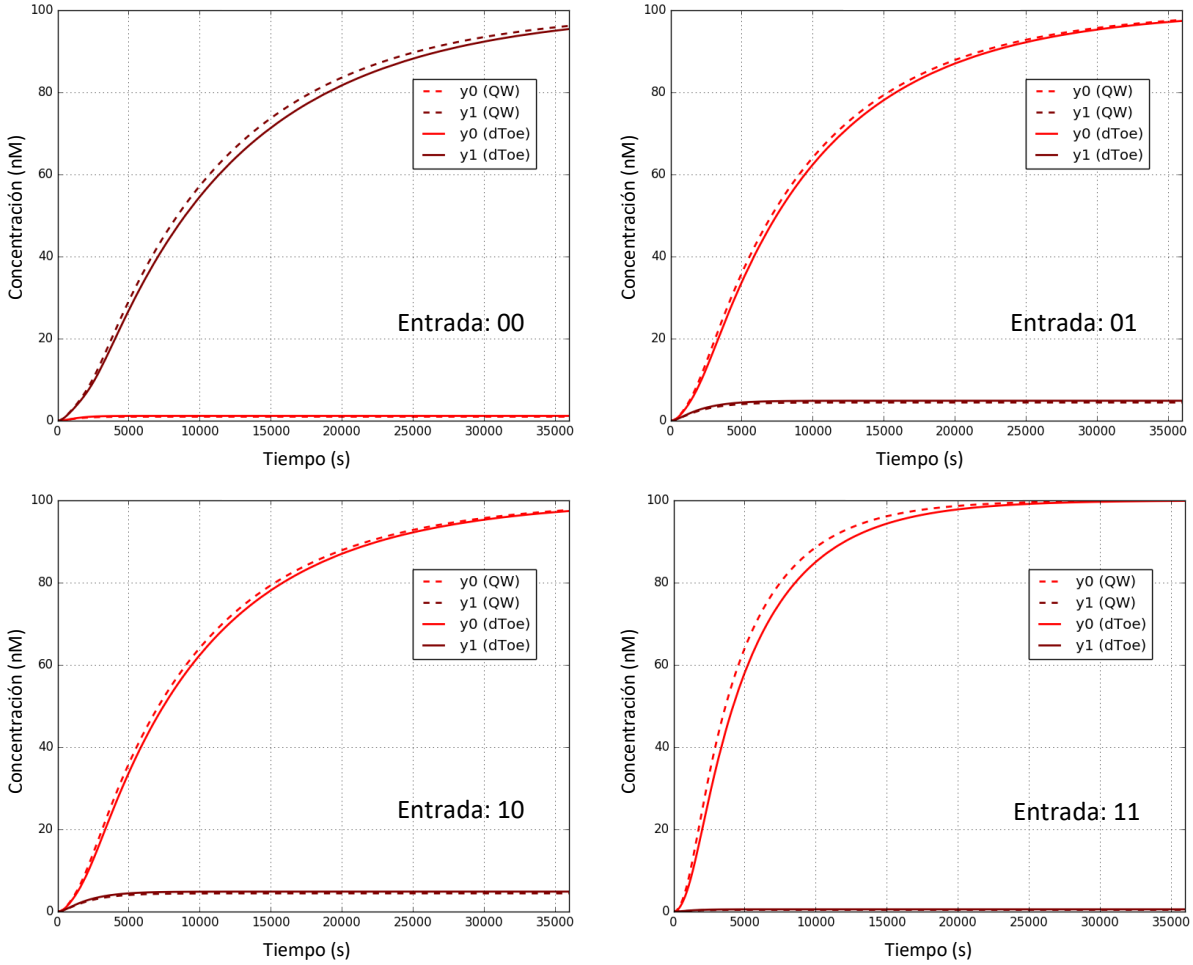

Figura 42. Simulación de compuerta NOR en Visual DSD. Escenario de simulación con constantes de unión lenta y desunión rápida.

te. Observe como en ambas representaciones, la disposición de los nodos abstractos coincide con las compuertas del circuito mostrado en la Figura 23. Los nodos sombreados en el diagrama abstracto doble toehold representan compuertas con toehold T_0, mientras que los blancos representan compuertas T_1. Los nodos que aparecen con una parte sombreada y otra blanca, indican que se realiza un cambio de toehold (es decir, la compuerta amplificadora es de la forma $G^{01}_{i:i,j}$ o $G^{10}_{i:i,j}$). Observe que tales cambios de toehold se requieren en las compuertas que tienen inversión (NAND y NOR).

El modelo químico correspondiente para la simulación de este sistema (sin considerar las reacciones de fuga (*leakage*), así como las constantes de reacción y una simplificación para reducir la cantidad de reacciones no productivas fueron tomadas según lo sugerido en el material suplementario de Qian y Winfree (2011).

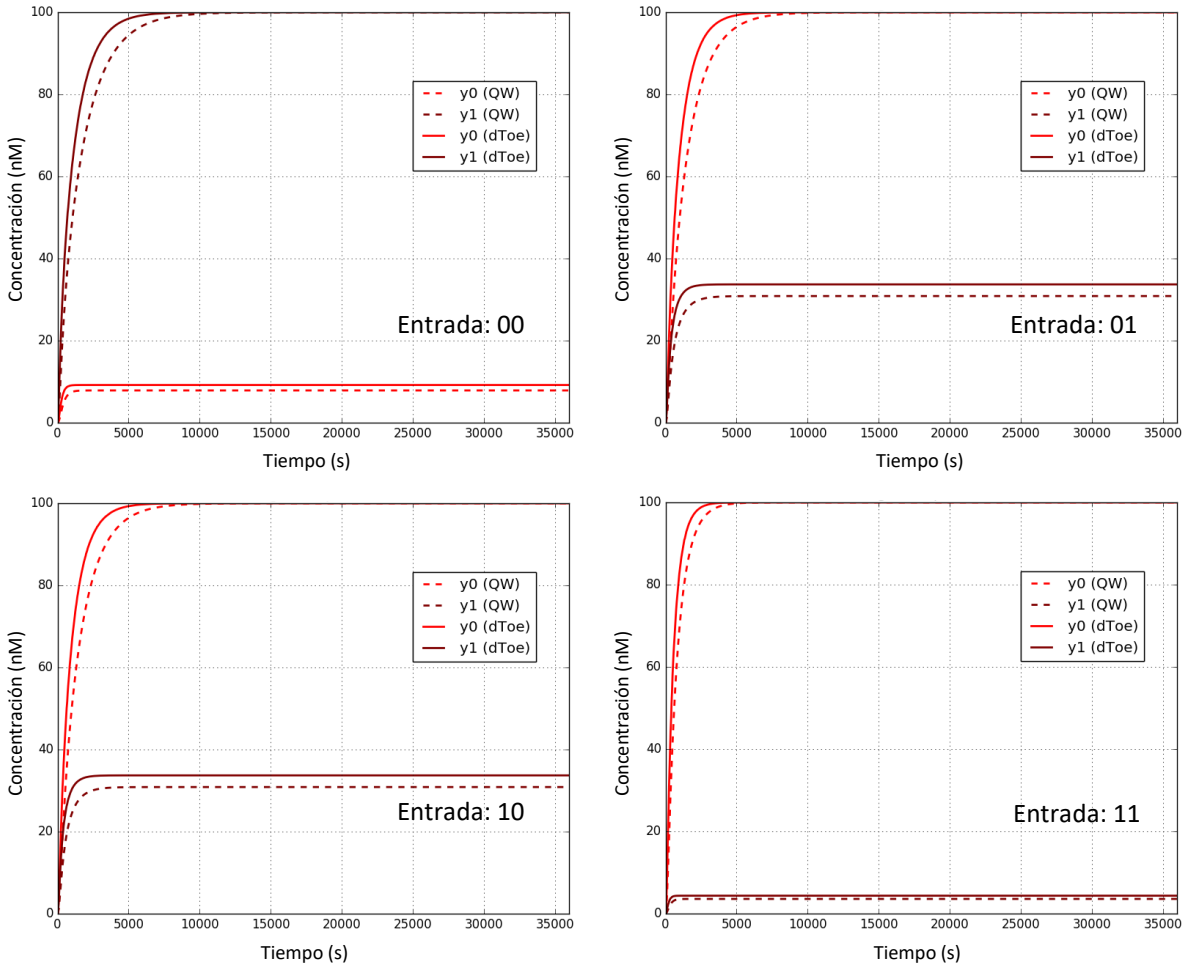

Figura 43. Simulación de compuerta NOR en Visual DSD. Escenario de simulación con constantes normales.

Tal simplificación fue indispensable puesto que la cantidad de reacciones improductivas que se generan en el circuito es muy grande, lo que genera un alto costo computacional para los simuladores estocásticos y aún para los determinísticos.

Las Figuras 45 a 48 muestran los resultados de la simulación determinística de los modelos originales (QW) y doble toehold (dToe) con los parámetros especificados, incluyendo las 16 entradas posibles ($x_4x_3x_2x_1 = 0000 - 1111$). Todas las simulaciones se realizaron utilizando una concentración base de $Z = 50$ nM, por lo que idealmente el estado lógico “1” debe estar en el intervalo $45 - 50$ nM y el estado lógico “0” en $0 - 5$ nM de concentración. En primera instancia, el comportamiento cinético de estas simulaciones está en concordancia con los resultados experimentales y simulados del artículo original (Qian y Winfree, 2011). Es notable como en todos los casos, la res-

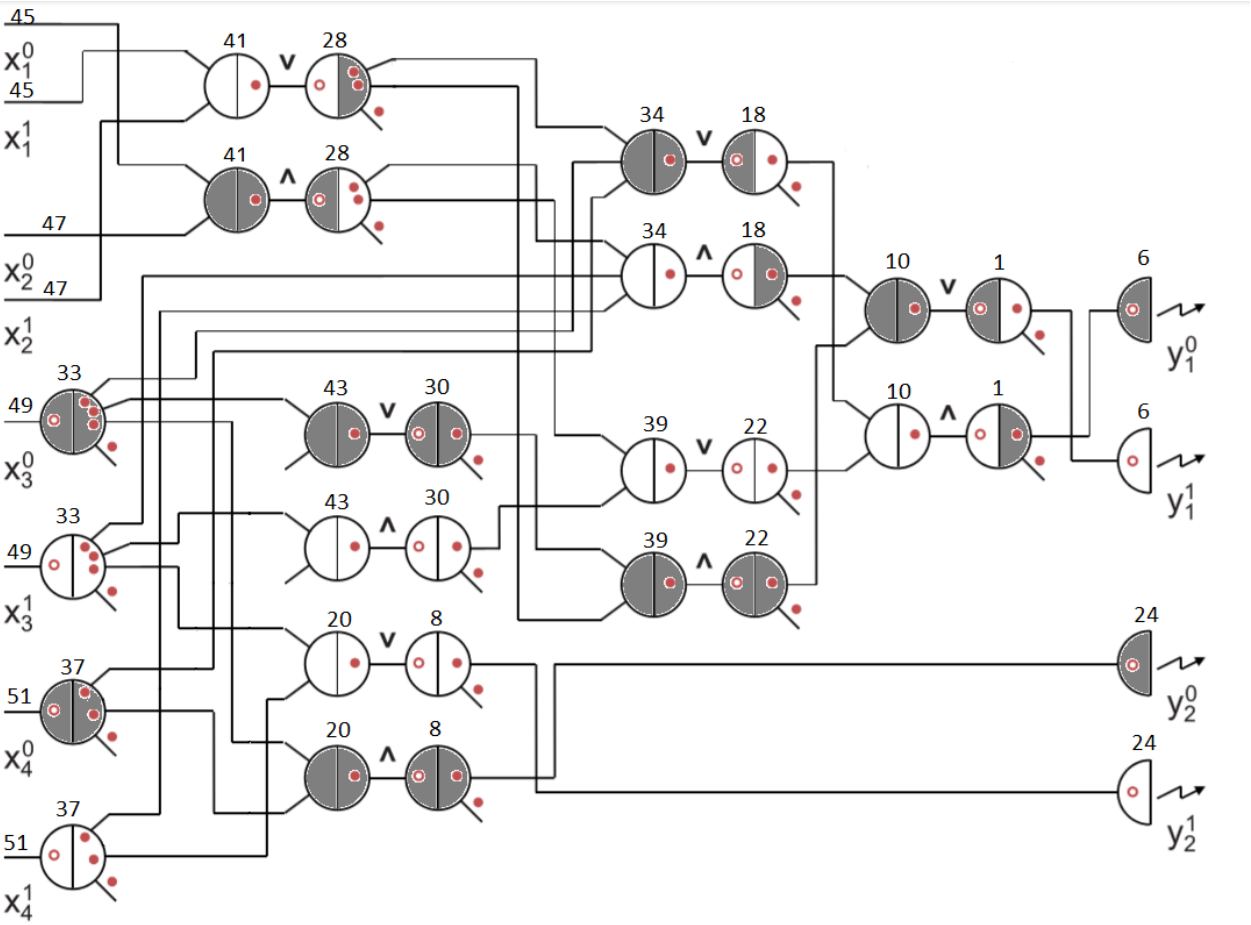

Figura 44. Representación abstracta equivalente del circuito para calcular la raíz cuadrada de un número de cuatro bits en el esquema modificado doble toehold.

puesta en nivel alto para la salida y_1 tiene siempre un retraso con respecto a y_2. Por ejemplo, para la entrada 0001, observe las salidas en alto y_1^1 y y_2^0. Esto es consecuencia de la naturaleza del circuito, ya que las salidas para y_1 (y_1^0 y y_1^1) se calculan a partir de varias compuertas en cascada, mientras que y_2^0 y y_2^1 solamente requieren una (ver Figura 23).

En los 16 casos, las salidas que corresponden a los estados altos, alcanzaron la concentración máxima (ideal) en el tiempo de simulación (10 horas). De manera general, se puede distinguir como las salidas para estos estados resultaron mejor para el modelo doble toehold. Observe también que las salidas en estado bajo presentan un comportamiento que se sale del intervalo ideal para ese estado lógico en todos los casos (con excepción de la entrada 0000). En estas salidas, es notable como el error es considerablemente menor para el modelo doble toehold. Por ejemplo, en la Figura

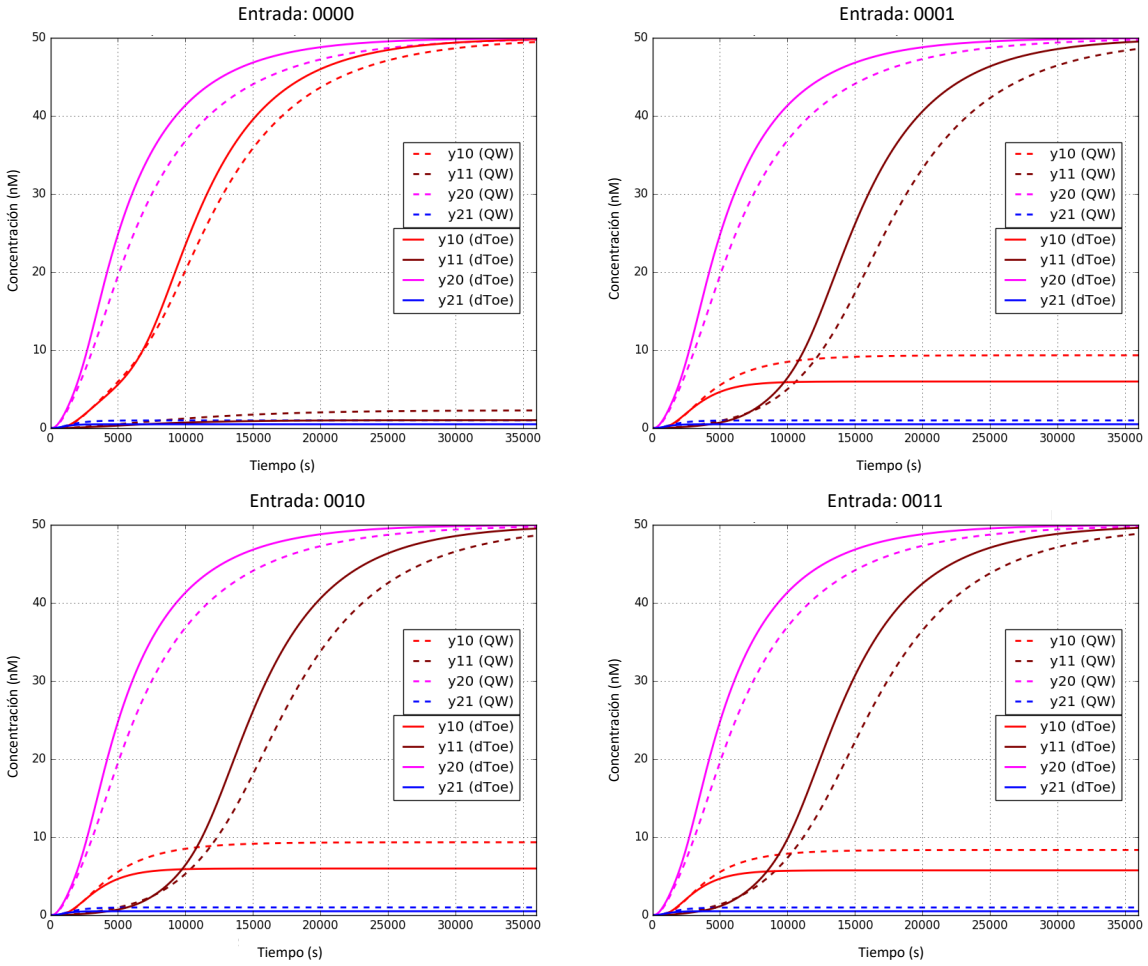

Figura 45. Resultados de simulación del circuito para calcular la raíz cuadrada de un número de cuatro bits. Las cuatro gráficas corresponden a las entradas 0000 a 0011.

46 donde las salidas y_2^0 son cero en los cuatro casos, la concentración final para esta señal resultó en aproximadamente el 60 % de la del modelo original QW.

Este tipo de fugas en los estados lógicos bajos, son consecuencia de las reacciones improductivas, principalmente sobre las moléculas umbral. Este efecto es mayor en las compuertas simples AND, cuando las entradas son 01 o 10. Observe que ese es el caso en las entradas de la Figura 46, dado que la salida y_2^0 se calcula a partir de la función x_3^0 AND x_3^0 (ver circuito de doble carril en Figura 23). Así, a pesar de que esta salida se calcula directamente, sin cascadas de compuertas, se ve afectada por el efecto de estas reacciones espurias. Otros casos críticos en este sentido, son para las entradas 1001, 1010 y 1011 (Figura 47). En todos ellos, además de y_2^0, la salida y_1^0 debe ser baja. Analizando el circuito doble carril de la Figura 23 junto con las entradas,

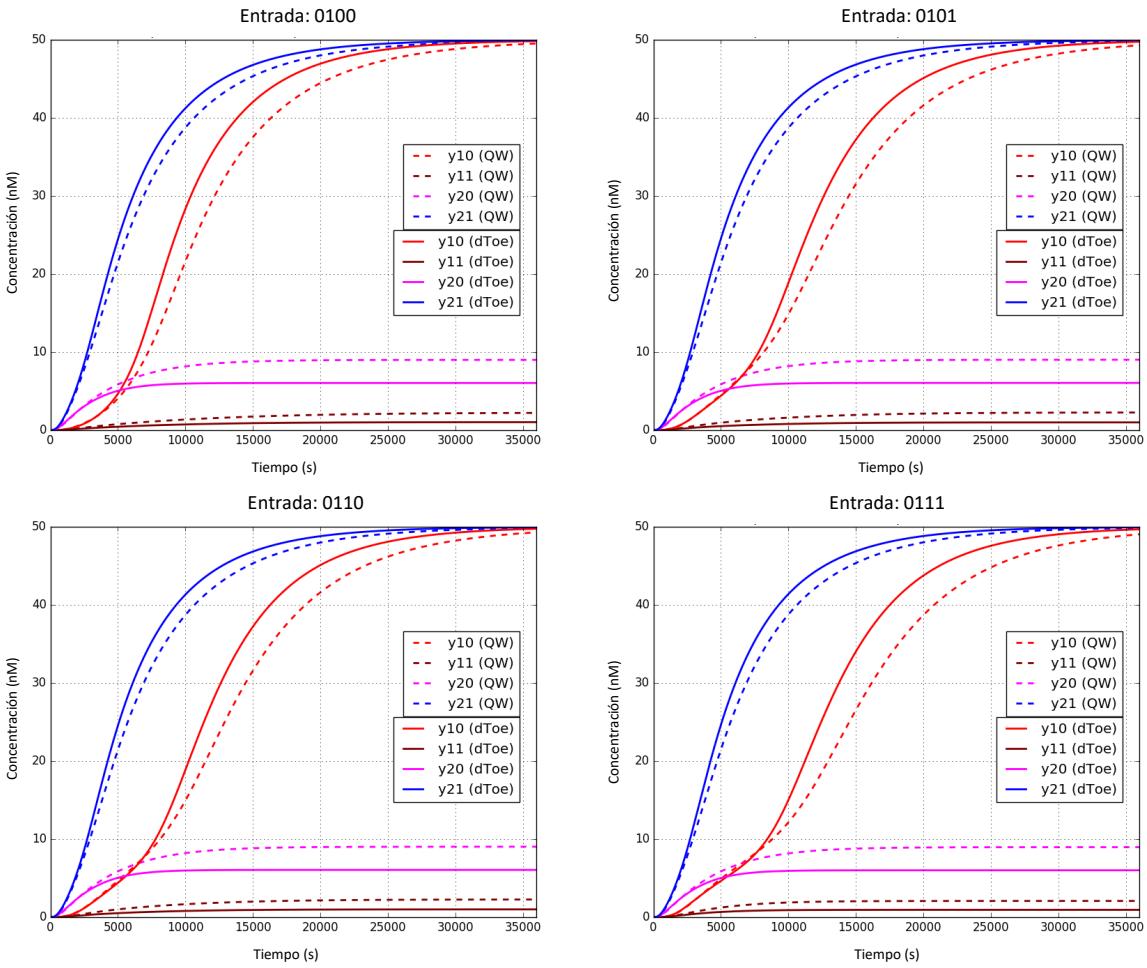

Figura 46. Resultados de simulación del circuito para calcular la raíz cuadrada de un número de cuatro bits. Las cuatro gráficas corresponden a las entradas 0100 a 0111.

uno puede comprobar que la compuerta AND que está justo antes de la salida, también recibe entradas opuestas 01 en las tres entradas mencionadas.

De esta manera, resulta intuitivo pensar que en circuitos a mayor escala aumentaría el efecto de las reacciones no productivas. Para probar este efecto, se simuló un circuito que consiste en cinco copias del circuito para calcular la raíz cuadrada, operando cada uno de forma independiente y paralela. En efecto, al aumentar la concentración de moléculas cable y de las demás compuertas, se aumenta la tasa efectiva de las reacciones espurias. La Figura 49 muestra los resultados de simulación para este circuito escalado con la entrada 1111, a una concentración base $Z = 50$ nM. El tiempo de simulación se duplicó para poder observar la evolución cinética de las salidas hasta el final. Las salidas mostradas corresponden a uno de los cinco circuitos (que

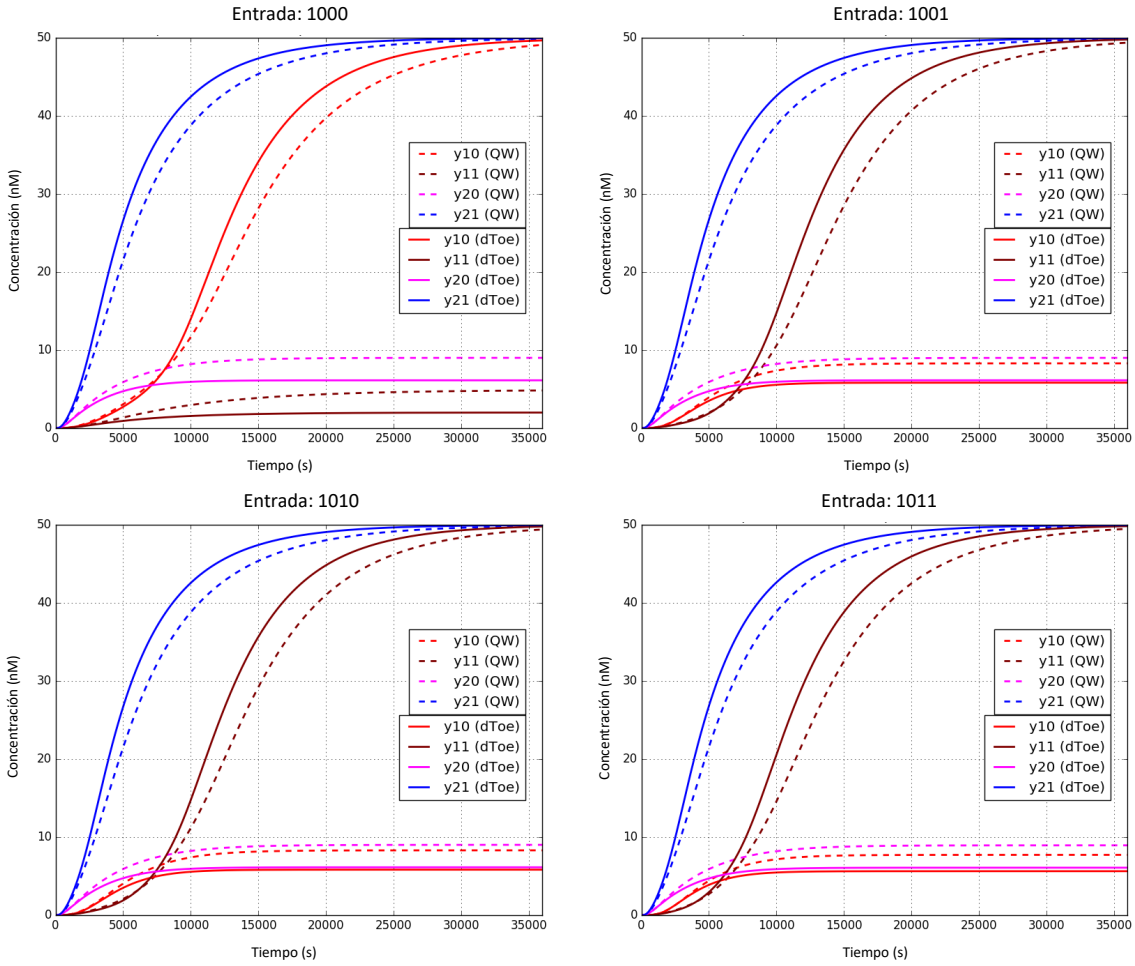

Figura 47. Resultados de simulación del circuito para calcular la raíz cuadrada de un número de cuatro bits. Las cuatro gráficas corresponden a las entradas 1000 a 1011.

mantuvieron un comportamiento casi idéntico).

Notablemente, la evolución de las salidas en estado alto y_2^1 y y_1^1 presenta una mayor separación en los dos modelos, con respecto a la que se obtenía en un solo circuito (Figura 48), lo cual sugiere que el efecto positivo de la segmentación del modelo de doble toehold se acentúa con circuitos de mayor complejidad. Aún más importante, el error en las señales en estado bajo se contiene en mayor medida. En este ejemplo, la salida y_1^0 se eleva hasta cerca de los 30 nM de concentración (casi 60 % de la máxima concentración) en el tiempo simulado (y mantiene una pendiente de crecimiento). En cambio, la misma salida en el modelo de doble toehold, asciende más lentamente hasta aproximadamente los 18 nM (36 % de la máxima concentración), para mantenerse en un estado casi estable al final de la simulación.

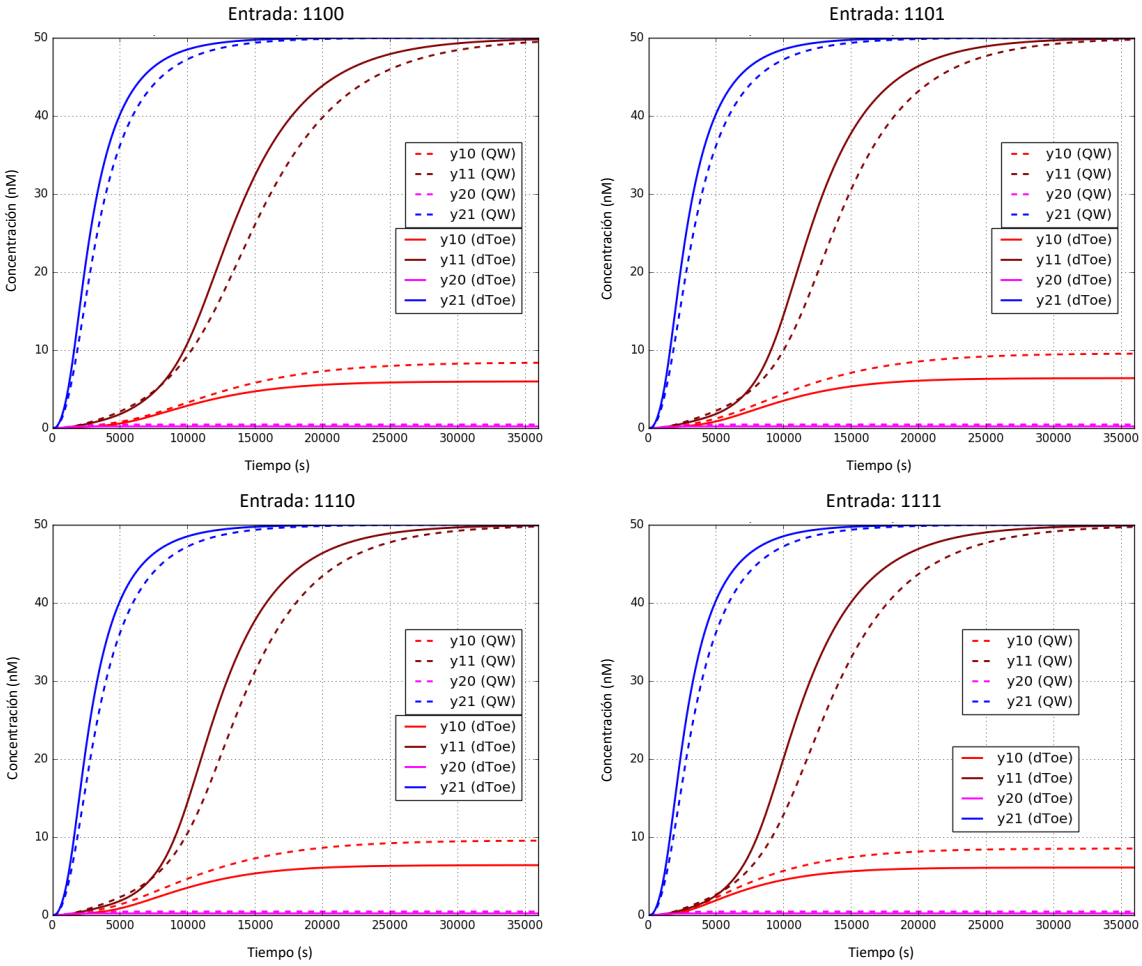

Figura 48. Resultados de simulación del circuito para calcular la raíz cuadrada de un número de cuatro bits. Las cuatro gráficas corresponden a las entradas 1100 a 1111.

La tasa efectiva de las reacciones espurias se puede controlar reduciendo las concentraciones en el sistema (Qian y Winfree, 2011). Este impacto se puede verificar en la Figura 50, que muestra los resultados del circuito quintuplicado a una concentración base de 10 nM. En este escenario, la concentración de la salida problemática y_1^0 en el modelo original QW, se contiene en aproximadamente 1.8 nM, lo cual representa el 18 % de la concentración base; y se mantiene con una pendiente de ascenso casi plana. En el modelo doble toehold, esta salida se mantiene al final cercana a los 1.2 nM (12 % de la concentración base), para estar cerca de la frontera del nivel lógico que le corresponde (10 %). A pesar de esta corrección considerable en el error, es evidente que a medida que el circuito crece en complejidad, tiene que reducirse la concentración base en la misma proporción, por lo que existe un límite físico en esta medida.

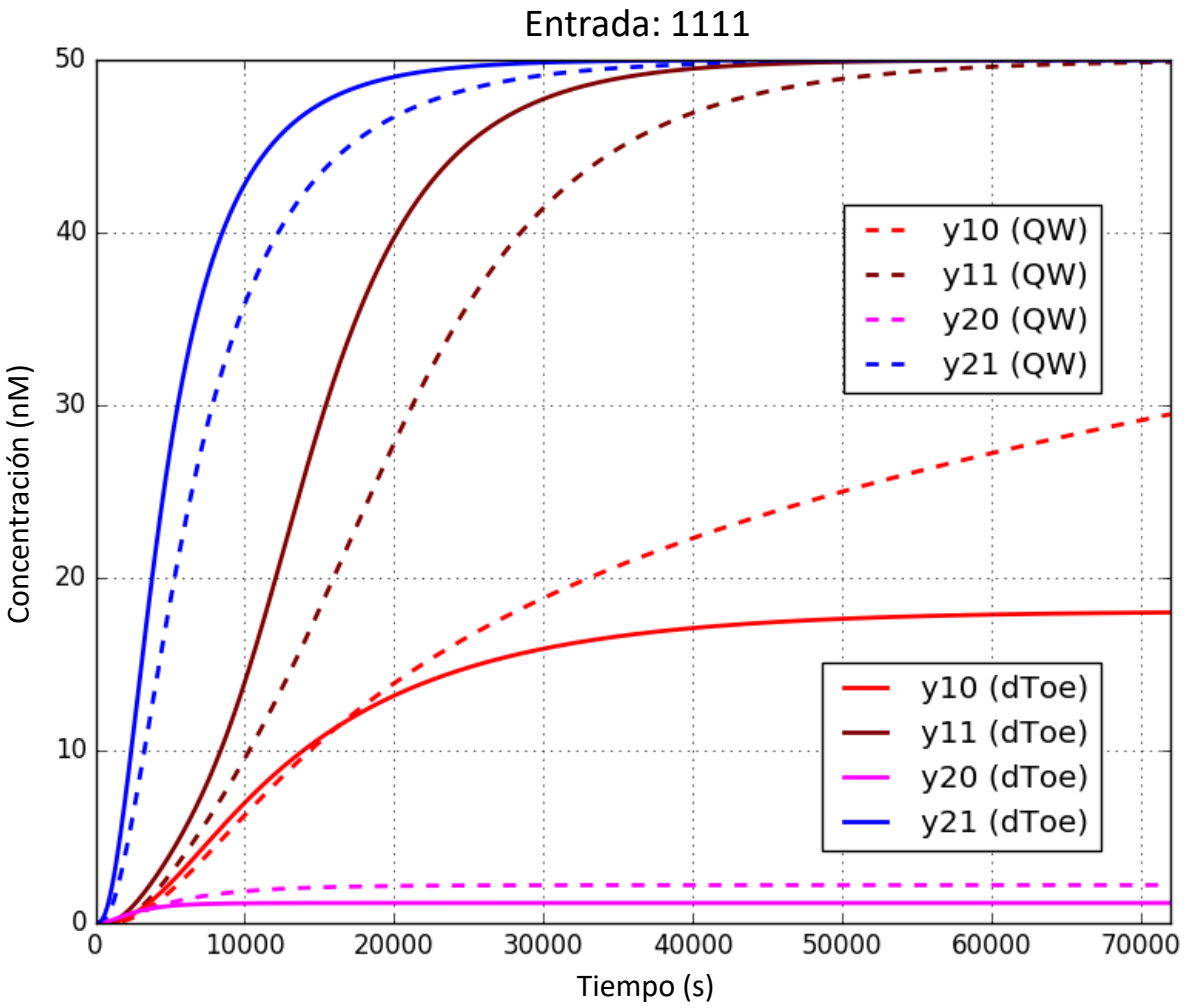

Figura 49. Resultados de simulación del circuito que contiene cinco copias del circuito de la Figura 23) operando de forma paralela. La concentración base es de $Z = 50$ nM.

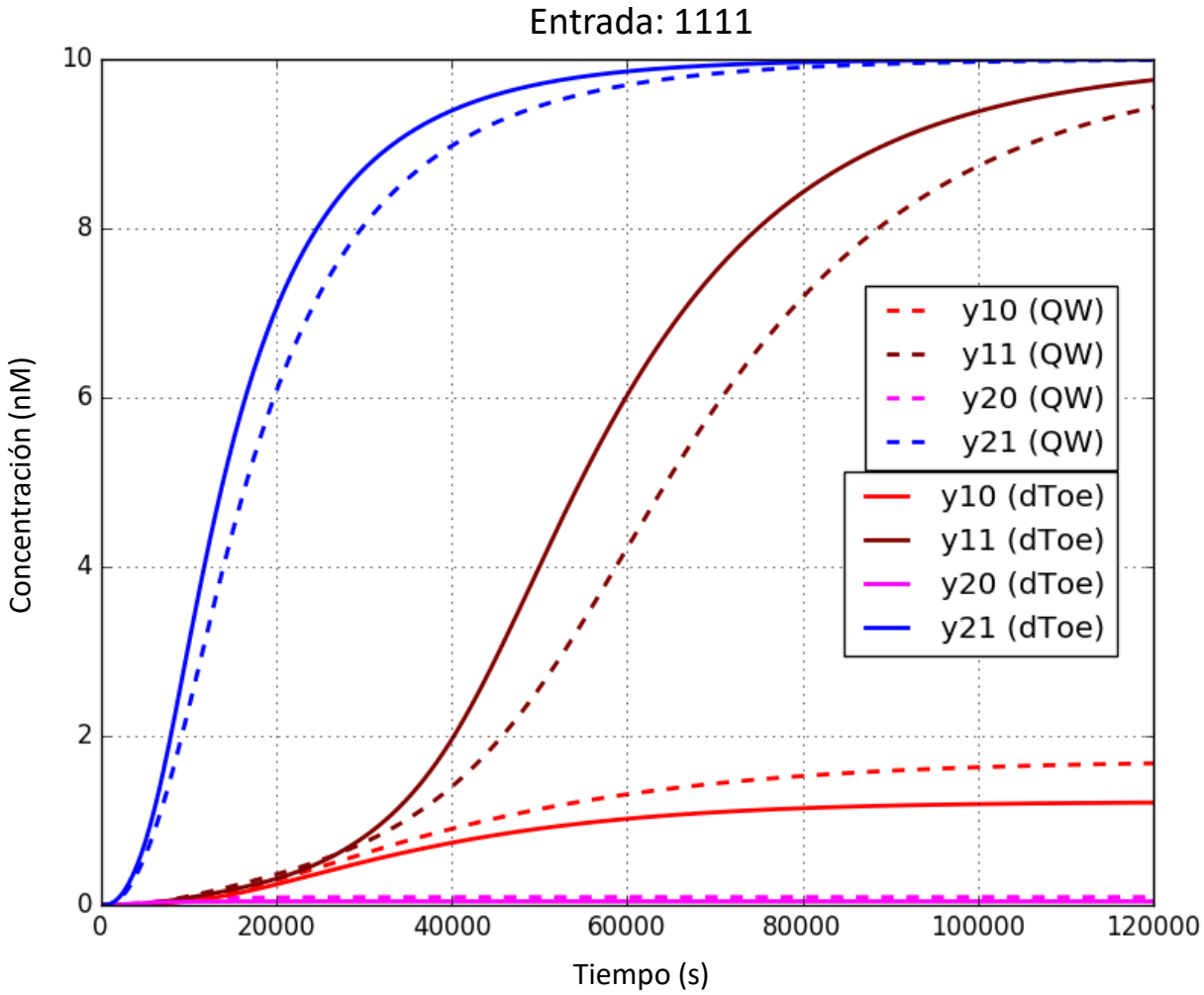

Figura 50. Resultados de simulación del circuito que contiene cinco copias del circuito de la Figura 23) operando de forma paralela. La concentración base es de $Z = 10$ nM.

Capítulo 6. Conclusiones y trabajo futuro

La presente investigación introduce algunos aportes en el área del cómputo biomolecular en dos paradigmas de cómputo: la inferencia lógica y los circuitos lógicos moleculares.

Por un lado, se presenta un nuevo sistema para la resolución de programas lógicos a nivel molecular basados en el mecanismo de desplazamiento de hebras de ADN. El sistema diseñado codifica reglas y hechos en ADN para realizar rutas de inferencia hacia adelante o hacia atrás, para responder una consulta inicial. La codificación permite que las entradas actúen como agentes catalizadores sobre diferentes compuertas de implicación, liberando las hebras de salida sin agotarse. Esta característica constituye la principal brecha en investigación abordada en el presente estudio, dado que con este aporte se podría potencialmente superar al estado del arte considerando aplicaciones en regímenes de bajas concentraciones, donde la sensibilidad de los dispositivos es un tema relevante (Shi *et al.*, 2016; Graybill y Bailey, 2016).

Adicionalmente, se diseñó un procedimiento simple para verificar la viabilidad termodinámica de los motivos del modelo. Tal procedimiento produjo secuencias adecuadas para la construcción de moléculas termodinámicamente estables. También se verificó el comportamiento cinético con reacciones de interferencia inherentes al modelo. Tales reacciones contienen un paso de migración de ramas de 4 vías, la cual parece introducir una barrera cinética que se deriva en una baja propensión de las interferencias con respecto a las reacciones diseñadas. Como consecuencia, los resultados de simulación sugirieron que aún con estas reacciones, las señales de salida ofrecen una clara distinguibilidad entre inferencias positivas y negativas.

Comparando los resultados de simulación aquí obtenidos contra los experimentales de Ran *et al.* (2009) en los mismos casos de prueba, se pudo observar que el esquema propuesto es competitivo. Además, dos ventajas directas sobre tal trabajo son la capacidad de representar una mayor cantidad de variables (dominios largos de 15 o más nt contra extremos pegajosos de 4 nt) y estar basado en un mecanismo libre de enzimas. Esto último le confiere una mayor posibilidad de ser aplicado en ambientes *in vivo*, dado que las enzimas requieren condiciones más controladas de operación. Con respecto a la propuesta de Sainz de Murieta y Rodríguez-Patón (2012), quienes también

se basan en desplazamiento de hebras, una ventaja es la ya mencionada capacidad catalítica del modelo aquí propuesto. Además, dado que su codificación de variables consiste en dos toeholds concatenados — cuya longitud es crucial en el desempeño cinético de los dispositivos basados en desplazamiento de hebras — existe un límite menor en la representación de variables.

Por otro lado, se incluyó una propuesta para mejorar el estado del arte en circuitos lógicos moleculares, particularmente la compuerta sube-baja (Qian y Winfree, 2011), que ha sido ya objeto de estudio en otras investigaciones en las que se han propuesto algunas modificaciones para la reusabilidad de las compuertas (Song *et al.*, 2017; Eshra *et al.*, 2019) y la minimización de las reacciones de fuga (Song *et al.*, 2018). En el presente trabajo se planteó el uso de dos toeholds para manejar las señales en estado bajo y alto en la lógica de doble carril. La ventaja inmediata de dicha medida repercute en la escalabilidad, dado que requiere la mitad de dominios representativos, con respecto al esquema original. Aunque para los circuitos considerados a la fecha, esta ventaja no resulta de gran trascendencia (al requerir solamente algunas decenas de dominios), en el futuro sí podría resultar crucial para la implementación de circuitos de mayor complejidad. Además, bajo este esquema, la dualidad de los toeholds produce una segmentación que divide las reacciones no productivas (causantes principales de fugas en las señales en estado bajo), minimizando así su impacto. Los resultados de simulaciones mostraron mejores comportamientos del esquema modificado y sugieren que el impacto podría ser mayor en circuitos a mayor escala que el contemplado.

A continuación, se expone un número de perspectivas para la mejora de lo obtenido en el presente documento.

- Aunque los resultados de simulación son ampliamente aceptados como una buena aproximación al comportamiento de los sistemas en la realidad, se sugiere se realice una validación experimental en condiciones reales.
- En el diseño de las moléculas implicación del Capítulo 4, el detalle de las reacciones de interferencia se debe al segmento colgante que contiene un toehold. Algunas posibilidades de eliminar esas reacciones serían secuestrar el toehold en el segmento, ya sea en una horquilla, uniéndolo a la hebra base o cubriéndolo con una hebra complementaria para liberarlo en otra reacción de intercambio de

toeholds con una molécula especialmente diseñada para ello. Todas estas posibles soluciones requieren el diseño de nuevos métodos para probarlas, dado que su implementación conlleva a que la molécula implicación tenga una estructura con pseudo-nudos, lo que no es manejado por los simuladores actuales.

- Una característica que está cobrando auge en el área de investigación, es la reutilización de los dispositivos. Así, un buen punto de mejora consiste en diseñar mecanismos que, una vez realizado el cómputo, recuperen las moléculas originales (esto es, una especie de “reinicio” del sistema).

- Al igual que el diseño del que se tomó inspiración (Ran *et al.*, 2009), el modelo de programación lógica aquí presentado se limita a predicados unarios. Un siguiente paso para dotar de mayor poder expresivo a estos esquemas es considerar predicados binarios o incluso n-arios. Además, para superar esta limitación, se debe diseñar una forma de implementar la sustitución de variables (instanciación). A la fecha solamente ha habido propuestas teóricas que abordan esta cuestión (Mihalache, 1997; Kobayashi, 1999; Uejima *et al.*, 2002).

Literatura citada

Adleman, L. M. (1994). Molecular Computation of Solutions to Combinatorial Problems. *Science*, **266**(11): 1021–1024.

Amir, Y., Ben-Ishay, E., Levner, D., Ittah, S., Abu-Horowitz, A., y Bachelet, I. (2014). Universal computing by DNA origami robots in a living animal. *Nature Nanotechnology*, **9**: 353–357.

Andersen, E. S., Dong, M., Nielsen, M. M., Jahn, K., Subramani, R., Mamdouh, W., Golas, M. M., Sander, B., Stark, H., Oliveira, C. L. P., Pedersen, J. S., Birkedal, V., Besenbacher, F., Gothelf, K. V., y Kjems, J. (2009). Self-assembly of a nanoscale DNA box with a controllable lid. *Nature*, **459**: 73–76.

Benenson, Y. (2012). Biomolecular computing systems: principles, progress and potential. *Nat. Rev. Genet.*, **13**: 455–468.

Benenson, Y., Paz-Elizur, T., Adar, R., Keinan, E., Livneh, Z., y Shapiro, E. (2001). Programmable and autonomous computing machine made of biomolecules. *Nature*, **414**: 430–434.

Bonnet, J., Yin, P., Ortiz, M. E., Subsoontorn, P., y Endy, D. (2013). Amplifying genetic logic gates. *Science*, **340**(6132): 599–603.

Chatterjee, G., Dalchau, N., Muscat, R. A., Phillips, A., y Seelig, G. (2017). A spatially localized architecture for fast and modular DNA computing. *Nature Nanotechnology*, **12**: 920–927.

Chen, Y.-J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., y Seelig, G. (2013). Programmable chemical controllers made from DNA. *Nature Nanotechnology*, **8**: 755–762.

Chen, Y.-J., Groves, B., Muscat, R. A., y Seelig, G. (2015). DNA nanotechnology from the test tube to the cell. *Nature Nanotechnology*, **10**: 748–760.

Cherry, K. M. y Qian, L. (2018). Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, **559**: 370–376.

Cook, M., Soloveichik, D., Winfree, E., y Bruck, J. (2009). *Programmability of Chemical Reaction Networks*, pp. 543–584. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dabby, N. L. (2013). *Synthetic Molecular Machines for Active Self-Assembly: Prototype Algorithms, Designs, and Experimental Study*. Tesis de doctorado, California Institute of Technology.

Dirks, R., Bois, J., Schaeffer, J., Winfree, E., y Pierce, N. (2007). Thermodynamic Analysis of Interacting Nucleic Acid Strands. *SIAM Review*, **49**(1): 65–88.

Du, Y. y Dong, S. (2017). Nucleic acid biosensors: Recent advances and perspectives. *Anal. Chem.*, **89**(1): 189–215.

Eshra, A., Shah, S., Song, T., y Reif, J. (2019). Renewable DNA hairpin-based logic circuits. *IEEE Transactions on Nanotechnology*, **18**: 252–259.

Evans, A., Thadani, N., y Suh, J. (2016). Biocomputing nanoplatforms as therapeutics and diagnostics. *Journal of Controlled Release*, **240**: 387–393.

Feynman, R. P. (1960). There's plenty of room at the bottom. *Engineering and Science*, **23**(5): 22–36.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, **81**(25): 2340–2361.

Graybill, R. M. y Bailey, R. C. (2016). Emerging biosensing approaches for microRNA analysis. *Anal. Chem.*, **88**(1): 431–450.

Grun, C., Werfel, J., Zhang, D. Y., y Yin, P. (2015). DyNAMiC Workbench: an integrated development environment for dynamic DNA nanotechnology. *J. R. Soc. Interface*, **12**(111): 20150580.

Guo, Y., Wei, B., Xiao, S., Yao, D., Li, H., Xu, H., Song, T., Li, X., y Liang, H. (2017). Recent advances in molecular machines based on toehold-mediated strand displacement reaction. *Quantitative Biology*, **5**(1): 25–41.

Hagiya, M., Arita, M., Kiga, D., Sakamoto, K., y Yokoyama, S. (1999). Towards parallel evaluation and learning of boolean μ-formulas with molecules. En: H. Rubin y D. Harlan Wood (eds.), *DNA Based Computers III*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 48, pp. 57–72.

Hurley, P. J. (2000). *A concise introduction to logic. 10th edition*. Wadsworth Pub. Belmont, CA.

Ignatova, Z., Martínez-Pérez, I. M., y Zimmermann, K.-H. (2008). *DNA Computing Models*. Springer.

Jung, C. y Ellington, A. D. (2014). Diagnostic applications of nucleic acid circuits. *Acc. Chem. Res.*, **47**(6): 1825–1835.

Kahan, M., Gil, B., Adar, R., y Shapiro, E. (2008). Towards molecular computers that operate in a biological environment. *Physica D: Nonlinear Phenomena*, **237**(9): 1165–1172.

Kobayashi, S. (1999). Horn clause computation with DNA molecules. *Journal of Combinatorial Optimization*, **3**(2): 277–299.

Lai, Y.-H., Sun, S.-C., y Chuang, M.-C. (2014). Biosensors with built-in biomolecular logic gates for practical applications. *Biosensors*, **4**(3): 273–300.

Lakin, M. R., Youssef, S., Polo, F., Emmott, S., y Phillips, A. (2011). Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*, **27**(22): 3211–3213.

Lakin, M. R., Youssef, S., Cardelli, L., y Phillips, A. (2012). Abstractions for DNA circuit design. *J. R. Soc. Interface*, **9**(68): 470–486.

Lee, I.-H., Park, J.-Y., Jang, H.-M., Chai, Y.-G., y Zhang, B.-T. (2003). DNA Implementation of Theorem Proving with Resolution Refutation in Propositional Logic. En: M. Hagiya y A. Ohuchi (eds.), *DNA 8*, Vol. 2568 de *Lecture Notes in Computer Science*. Springer, pp. 156–167.

Lee, S. H., van Noort, D., Yang, K.-A., Lee, I.-H., Zhang, B.-T., y Park, T. H. (2012). Biomolecular theorem proving on a chip: a novel microfluidic solution to a classical logic problem. *Lab Chip*, **12**: 1841–1848.

Li, S., Jiang, Q., Liu, S., Zhang, Y., Tian, Y., Song, C., Wang, J., Zou, Y., Anderson, G. J., Han, J.-Y., Chang, Y., Liu, Y., Zhang, C., Chen, L., Zhou, G., Nie, G., Yan, H., Ding, B., y Zhao, Y. (2018). A DNA nanorobot functions as a cancer therapeutic in response to a molecular trigger in vivo. *Nature Biotechnology*, **36**(3): 258–264.

Lipton, R. (1995). DNA Solution of Hard Computational Problems. *Science*, **268**(5210): 542–545.

Lloyd, J. W. (1984). *Foundations of Logic Programming*. Springer-Verlag. Berlin, Heidelberg.

Lu, C.-H., Willner, B., y Willner, I. (2013). Dna nanotechnology: From sensing and dna machines to drug-delivery systems. *ACS Nano*, **7**(10): 8320–8332. PMID: 24070191.

Macdonald, J., Li, Y., Sutovic, M., Lederman, H., Pendri, K., Lu, W., Andrews, B. L., Stefanovic, D., y Stojanovic, M. N. (2006). Medium scale integration of molecular logic gates in an automaton. *Nano Lett.*, **6**(11): 2598–2603.

McCulloch, W. y Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**(115): 115–133.

Mihalache, V. (1997). Prolog approach to DNA computing. En: *Proc. IEEE*. Int. Conf. Evolutionary Computation, pp. 249–254.

Mulawka, J. J., Piotr, W., y Borsuk, P. (1998). Implementation of the inference engine based on molecular computing technique. En: *Proc. IEEE*. Int. Conf. on Evolutionary Computation, pp. 493–498.

Nunes de Castro, L. (2007). Fundamentals of natural computing: an overview. *Physics of Life Reviews*, **4**(1): 1–36.

Omabegho, T., Sha, R., y Seeman, N. C. (2009). A bipedal DNA brownian motor with coordinated legs. *Science*, **324**(5923): 67–71.

Ordóñez-Guillén, N. E. y Martínez-Pérez, I. M. (2019). Catalytic DNA strand displacement cascades applied to logic programming. *IEEE Access*, **7**(1): 100428–100441.

Ouyang, Q., Kaplan, P. D., Liu, S., y Libchaber, A. (1997). DNA Solution of the Maximal Clique Problem. *Science*, **278**(5337): 446–449.

Panyutin, I. y Hsieh, P. (1994). The kinetics of spontaneous DNA branch migration. *Proc. Natl. Acad. Sci. USA*, **91**(6): 2021–2025.

Phillips, A. y Cardelli, L. (2009). A programming language for composable DNA circuits. *J. R. Soc. Interface*, **6**(suppl_4): S419–S436.

Qian, L. y Winfree, E. (2009). A simple DNA gate motif for synthesizing large-scale circuits. *Journal of the Royal Society Interface*, **8**(62): 1281–1297.

Qian, L. y Winfree, E. (2011). Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, **332**: 1196–1201.

Qian, L., Soloveichik, D., y Winfree, E. (2011a). Efficient turing-universal computation with DNA polymers. En: Y. Sakakibara y Y. Mi (eds.), *DNA Computing and Molecular Programming*, Berlin, Heidelberg. Springer Berlin Heidelberg, pp. 123–140.

Qian, L., Winfree, E., y Bruck, J. (2011b). Neural network computation with DNA strand displacement cascades. *Nature*, **332**: 368–372.

Ramsey, S., Orrell, D., y Bolouri, H. (2005). DIZZY: Stochastic simulation of large-scale genetic regulatory networks. *Journal of Bioinformatics and Computational Biology*, **03**(02): 415–436.

Ran, T., Kaplan, S., y Shapiro, E. (2009). Molecular implementation of simple logic programs. *Nature nanotechnology*, **4**: 642–648.

Rodríguez-Patón, A., Larrea, J. M., y Sainz de Murieta, I. (2010). Inference with DNA molecules. En: C. Calude, M. Hagiya, K. Morita, G. Rozenberg, y J. Timmis (eds.), *Unconventional computation*, Vol. 6079 de *Lecture Notes in Computer Science*. Springer, p. 192.

Rodríguez-Patón, A., Sainz de Murieta, I., y Sosík, P. (2014). DNA strand displacement system running logic programs. *Biosystems*, **115**: 5–12.

Rogowski, L. y Sosík, P. (2014). The laws of natural deduction in inference by DNA computer. *The Scientific World Journal*, **2014**.

Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N. V., Goodman, M. F., Rothemund, P. W. K., y Adleman, L. M. (1998). A sticker based model for DNA computation. *Journal of Computational Biology*, **5**(4): 615–629.

Russell, S. y Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, tercera edición. Upper Saddle River, NJ, USA.

Sainz de Murieta, I. y Rodríguez-Patón, A. (2012). DNA biosensors that reason. *Biosystems*, **109**: 91–104.

SantaLucia, J. y Hicks, D. (2004). The thermodynamics of DNA structural motifs. *Annual Review of Biophysics and Biomolecular Structure*, **33**(1): 415–440.

SantaLucia, J. J. (1998). A unified view of polymer, dumbbell, and oligonucleotide dna nearest-neighbor thermodynamics. *Proc. Natl. Acad. Sci. U.S.A.*, **95**(4): 737–738.

Schaeffer, J. M. (2013). *Stochastic simulation of the kinetics of multiple interacting nucleic acid strands*. Tesis de doctorado, California Institute of Technology.

Schaeffer, J. M., Thachuk, C., y Winfree, E. (2015). Stochastic simulation of the kinetics of multiple interacting nucleic acid strands. En: *DNA Computing and Molecular Programming (DNA21), Lecture Notes in Computer Science (LNCS)*. Springer, Vol. 9211, pp. 194–211.

Seelig, G., Soloveichik, D., Zhang, D. Y., y Winfree, E. (2006). Enzyme-free nucleic acid logic circuits. *Science*, **314**(5805): 1585–1588.

Shi, K., Dou, B., Yang, J., Yuan, R., y Xiang, Y. (2016). Cascaded strand displacement for non-enzymatic target recycling amplification and label-free electronic detection of microRNA from tumor cells. *Analytica Chimica Acta*, **916**: 1–7.

Simmel, F. C. (2007). Towards biomedical applications for nucleic acid nanodevices. *Nanomedicine*, **2**(6): 817–830. PMID: 18095848.

Siuti, P., Yazbek, J., y Lu, T. K. (2013). Synthetic circuits integrating logic and memory in living cells. *Nature Biotechnology*, **31**(5): 448–452.

Siuti, P., Yazbek, J., y Lu, T. K. (2014). Engineering genetic circuits that compute and remember. *Nat. Protoc.*, **9**(6): 1292–1300.

Smith, D., Schüller, V., Engst, C., Rädler, J., y Liedl, T. (2013). Nucleic acid nanostructures for biomedical applications. *Nanomedicine*, **8**(1): 105–121.

Song, T., Gopalkrishnan, N., Eshra, A., Garg, S., Mokhtar, R., Bui, H., Chandran, H., y Reif, J. (2018). Improving the performance of DNA strand displacement circuits by shadow cancellation. *ACS Nano*, **12**(11): 11689–11697. PMID: 30372034.

Song, X. y Reif, J. (2019). Nucleic acid databases and molecular-scale computing. *ACS Nano*, **13**(6): 6256–6268. PMID: 31117381.

Song, X., Eshra, A., Dwyer, C., y Reif, J. (2017). Renewable DNA seesaw logic circuits enabled by photoregulation of toehold-mediated strand displacement. *RSC Adv.*, **7**(45): 28130–28144.

Sterling, L. y Shapiro, E. (1994). *The Art of Prolog (2Nd Ed.): Advanced Programming Techniques*. MIT Press. Cambridge, MA, USA.

Stojanovic, M. N. y Stefanovic, D. (2003). A deoxyribozyme-based molecular automaton. *Nature Biotechnology*, **21**(9): 1069–1074.

Thubagere, A. J., Thachuk, C., Berleant, J., Johnson, R. F., Ardelean, D. A., Cherry, K. M., y Qian, L. (2017). Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. *Nature Communications*, **8**(1): 14373.

Turing, A. M. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, **237**(641): 37–72.

Uejima, H., Hagiya, M., y Kobayashi, S. (2002). Horn Clause Computation by Self-assembly of DNA Molecules. En: N. Jonoska y N. C. Seeman (eds.), *DNA 7*, Vol. 2340 de *Lecture Notes in Computer Science*. Springer, pp. 308–320.

Velázquez Sánchez, A. K. (2014). *Diseño de compuertas lógicas mediante desplazamiento de cadenas de ADN*. Tesis de maestría, Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California. xiv, 147 hojas.

Venkataraman, S., Dirks, R. M., Rothemund, P. W. K., Winfree, E., y Pierce, N. A. (2007). An autonomous polymerization motor powered by DNA hybridization. *Nature Nanotechnology*, **2**: 490–494.

von Neumann, J. (1966). *Theory of Self-Reproducing Automata*. University of Illinois Press.

Wasiewicz, P., Janczak, T., J, M. J., y Plucienniczak, A. (1999). The inference via DNA computing. En: *Proc. of the 1999 Congress on Evolutionary Computation*. Vol. 2, pp. 988–993.

Wasiewicz, P., Janczak, T., J, M. J., y Plucienniczak, A. (2000). The inference based on molecular computing. *Cybernetics and Systems*, **31**(3): 283–315.

Waterman, M. S. (1995). *Introduction to Computational Biology: Maps, Sequences and Genomes*. Chapman Hall. London.

Watson, J. D. y Crick, F. H. (1953). Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, **171**(4356): 737–738.

Win, M. N. y Smolke, C. D. (2008). Higher-order cellular information processing with synthetic rna devices. *Science*, **322**(5900): 456–460.

Winfree, E., Eng, T., y Rozenberg, G. (2001). String tile models for dna computing by self-assembly. En: *Revised Papers from the 6th International Workshop on DNA-Based Computers: DNA Computing*, Berlin, Heidelberg. Springer-Verlag, DNA '00, pp. 63–88.

Yan, F., Gen, L., Yin, L., y Dahzi, M. (2007). Autonomous inference via DNA computing. *Progress in Natural Science*, **17**(6): 725–732.

Yurke, B., Turberfield, A. J., Mills Jr, A. P., Simmel, F. C., y Neumann, J. L. (2000). A DNA-fuelled molecular machine made of DNA. *Nature*, **406**: 605–608.

Zadeh, J. N., Steenberg, C. D., Bois, J. S., Wolfe, B. R., Pierce, M. B., Khan, A. R., Dirks, R. M., y Pierce, N. A. (2010). NUPACK: Analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, **32**(1): 170–173.

Zhang, D. Y. y Seelig, G. (2011). Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry*, **3**: 103–113.

Zhang, D. Y. y Winfree, E. (2009). Control of DNA strand displacement kinetics using toehold exchange. *J. Am. Chem. Soc.*, **131**(47): 17303–17314.

Zhang, D. Y., Turberfield, A. J., Yurke, B., y Winfree, E. (2007). Engineering entropy-driven reactions and networks catalyzed by DNA. *Science*, **318**(5853): 1121–1125.

Zhang, J. X., Zhang, J. X., Fang, J. Z., Duan, W., Wu, L. R., Zhang, A. W., Dalchau, N., Yordanov, B., Petersen, R., Phillips, A., y Zhang, D. Y. (2018). Predicting DNA hybridization kinetics from sequence. *Nature Chemistry*, **10**: 91–98.

Anexo A

Implementaciones en Visual DSD.

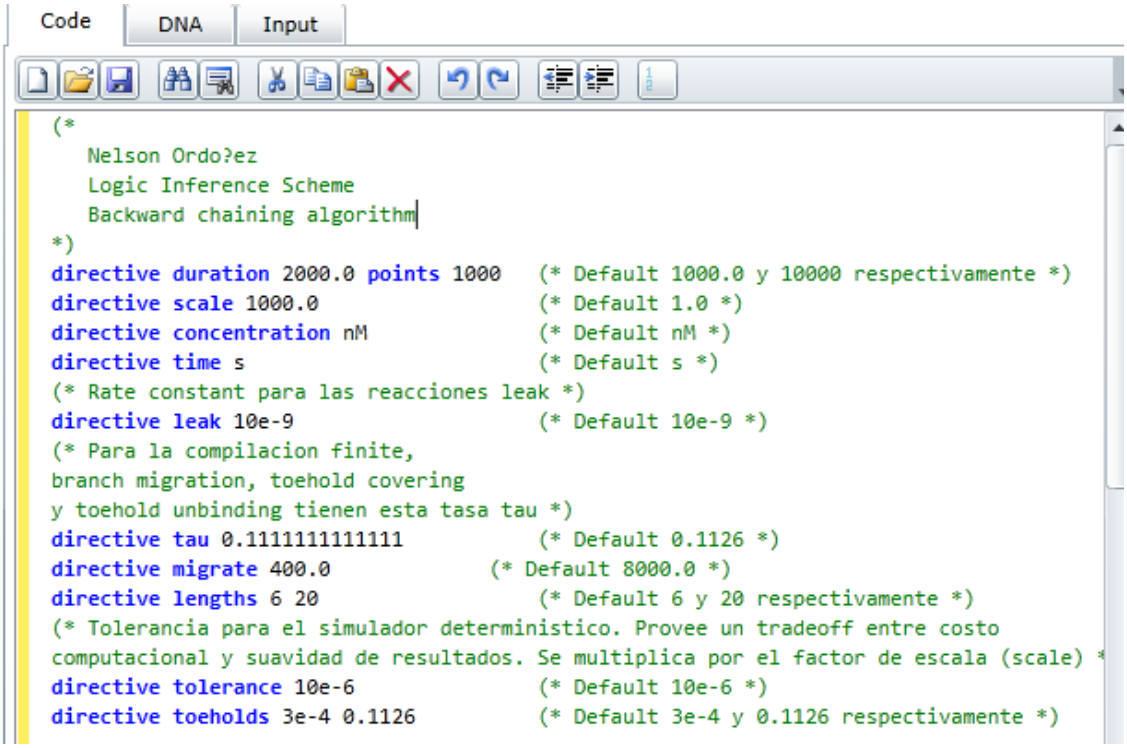

Figura 51. La sección de encabezado para todos los programas en Visual DSD

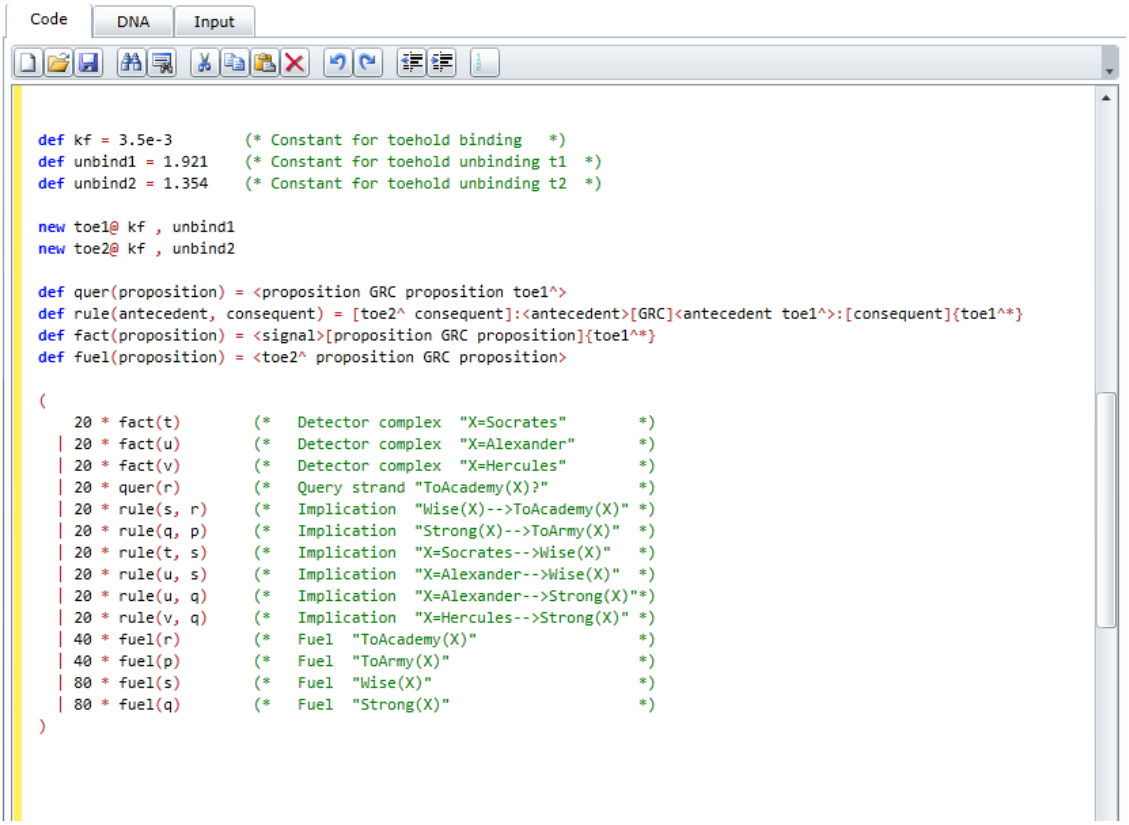

Figura 52. Implementación del programa E1 con la consulta inicial "*Academia*(X)". La versión NCAT no incluye moléculas combustible.

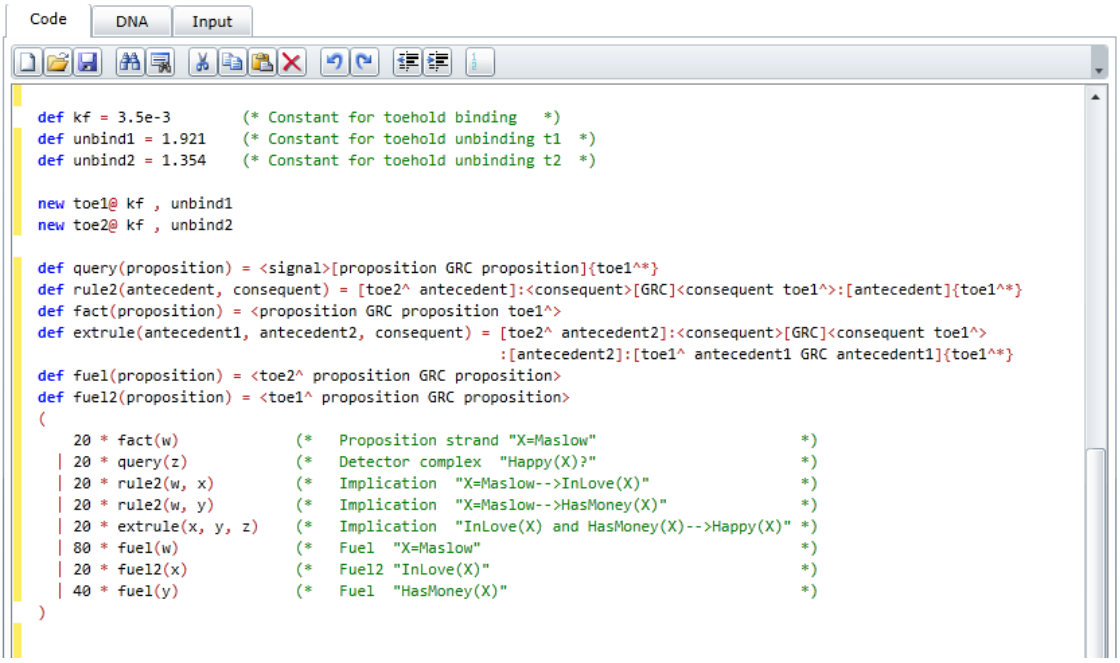

Figura 53. Implementación del programa E2 con la consulta inicial "*Feliz*(Maslow)". La versión NCAT no incluye moléculas combustible.

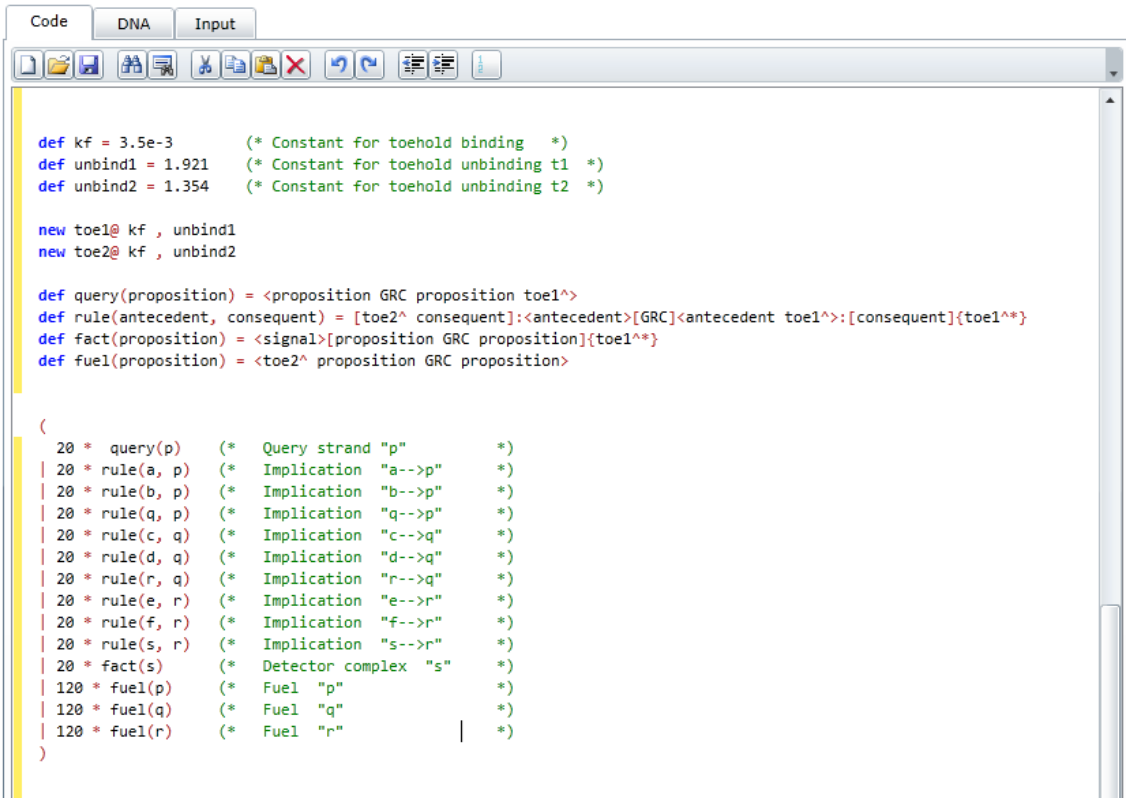

Figura 54. Implementación del programa E3. La versión NCAT no incluye moléculas combustible.

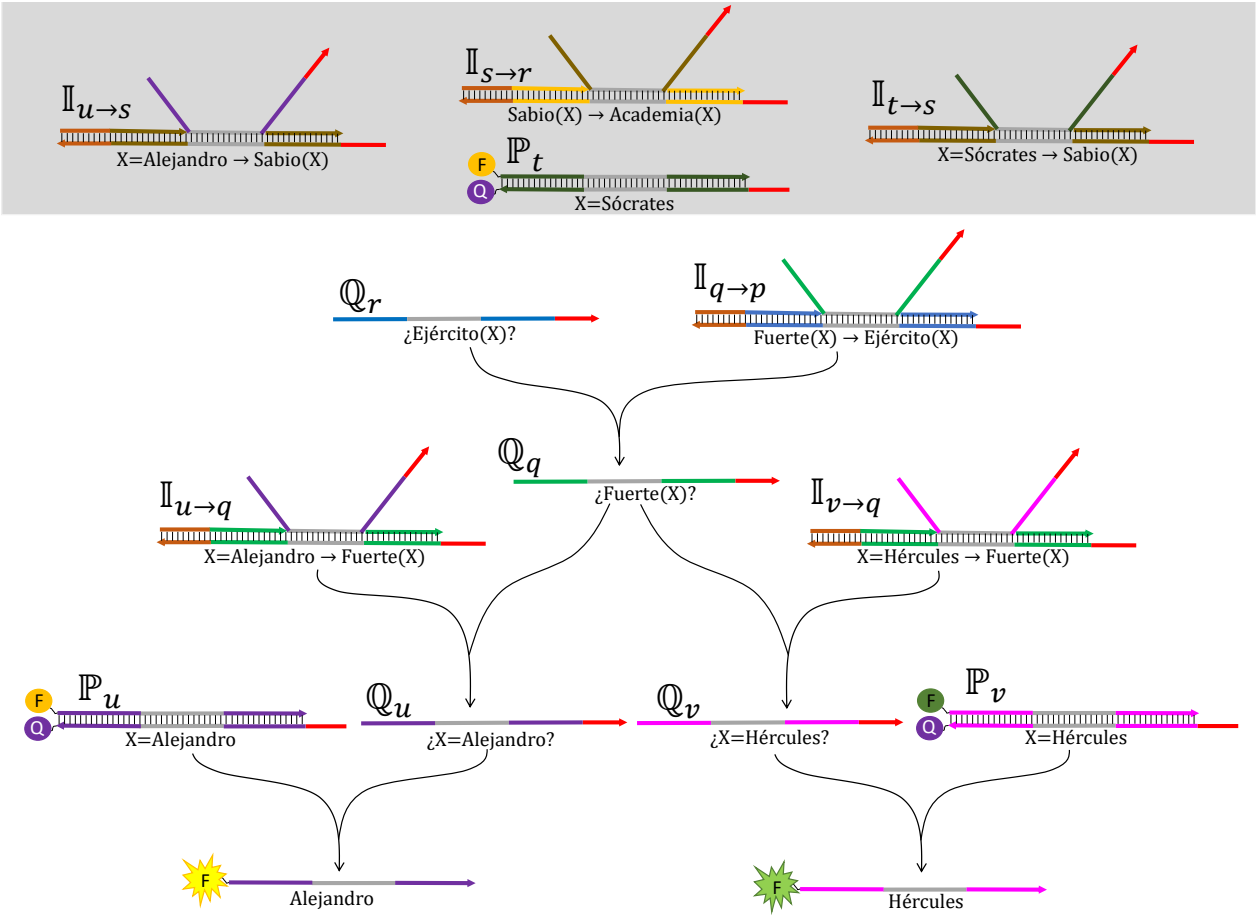

Figura 55. Proceso de encadenamiento hacia atrás para el Ejemplo E1 con la consulta inicial "¿Quién es apto para el ejército?". Al final del proceso, el sistema responde "Alejandro" y "Hércules". Las moléculas no reactivas aparecen en el rectángulo sombreado. Las reacciones de repostaje se omiten.

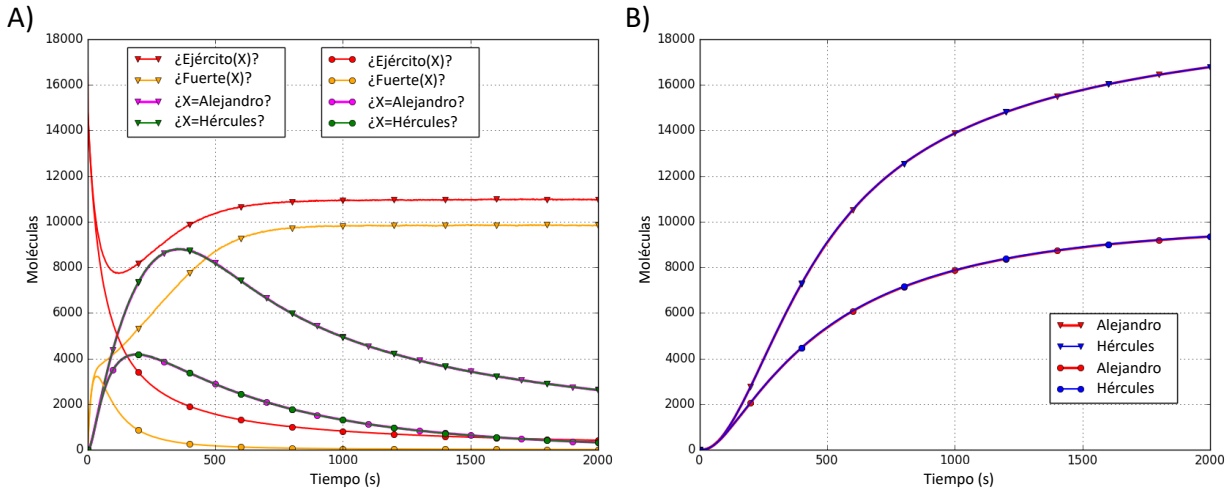

Figura 56. Resultados de simulación para el ejemplo E1 con la consulta inicial "¿Quién es apto para el ejército?". A) Comportamiento cinético de las consultas. B) Comportamiento cinético de las señales de salida. Los marcadores triangulares y circulares indican el modo CAT y NCAT, respectivamente.

Anexo B

Resultados de simulaciones termodinámicas a nivel de tubos.

Todas las imágenes fueron generadas con la versión NUPACK online (www.nupack.org)

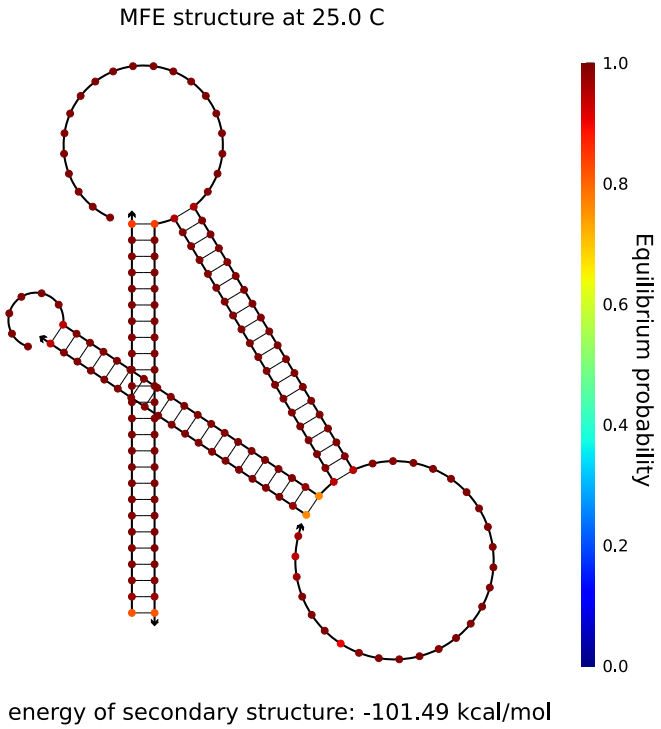

Figura 57. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{t \to s}$ (que corresponde al hecho *Sabio*(Sócrates)) del programa E1.

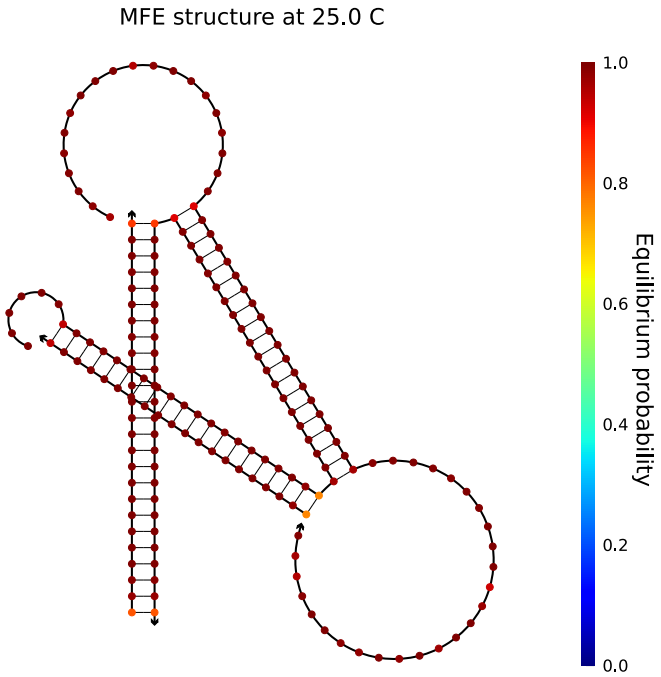

Figura 58. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{u \to s}$ (que corresponde al hecho *Sabio*(Alejandro)) del programa E1.

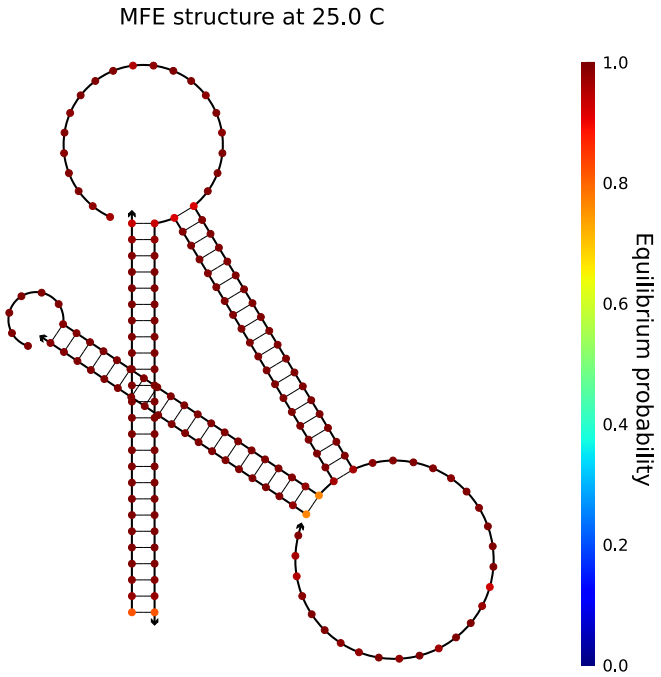

Figura 59. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{u \to q}$ (que corresponde al hecho *Fuerte*(Alejandro)) del programa E1.

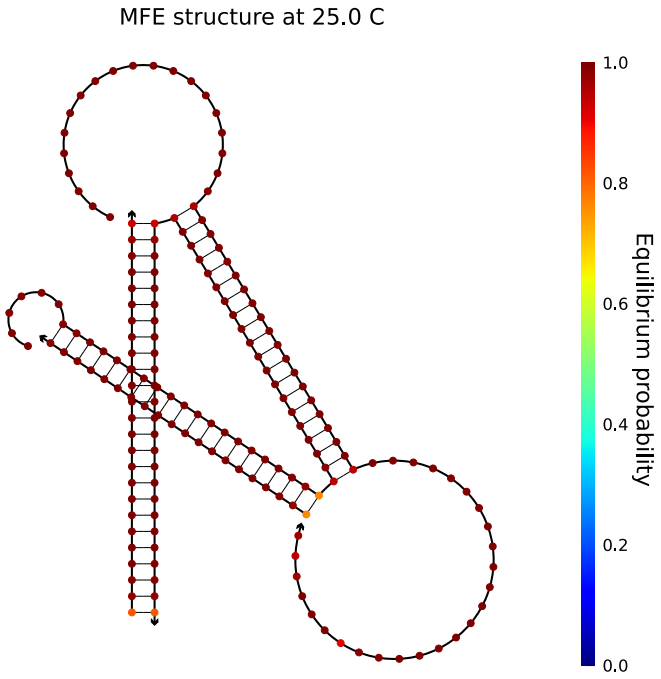

Figura 60. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{v \to q}$ (que corresponde al hecho *Fuerte*(Hércules)) del programa E1.

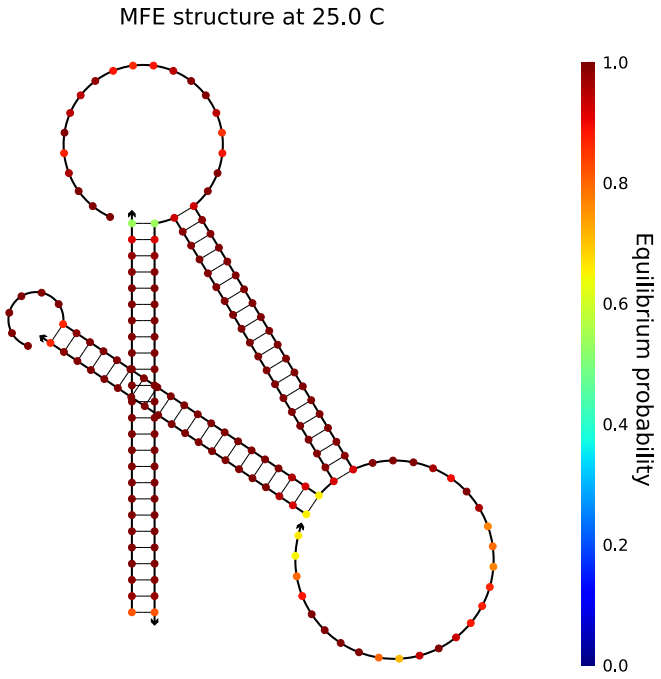

Figura 61. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{s \to r}$ (que corresponde a la regla *Sabio*(X)→*Academia*(X)) of program E1.

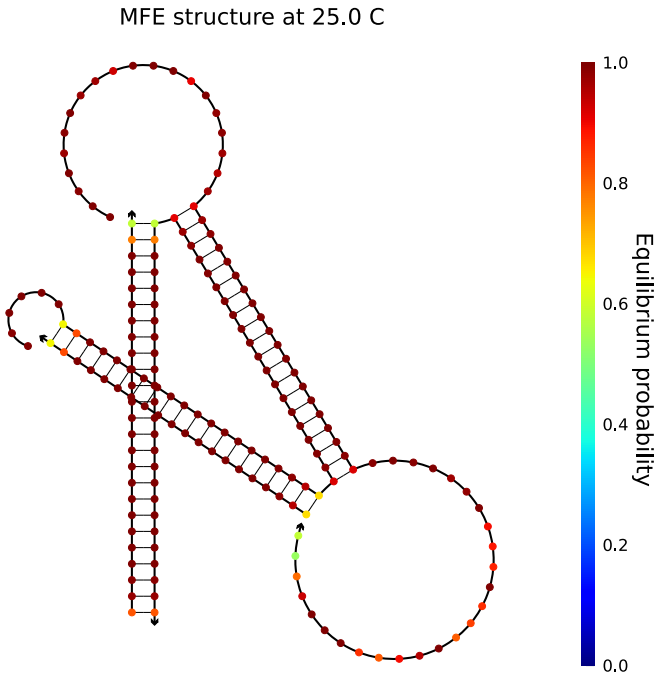

Figura 62. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{q \to p}$ (que corresponde a la regla *Fuerte*(X)→*Ejército*(X)) del programa E1.

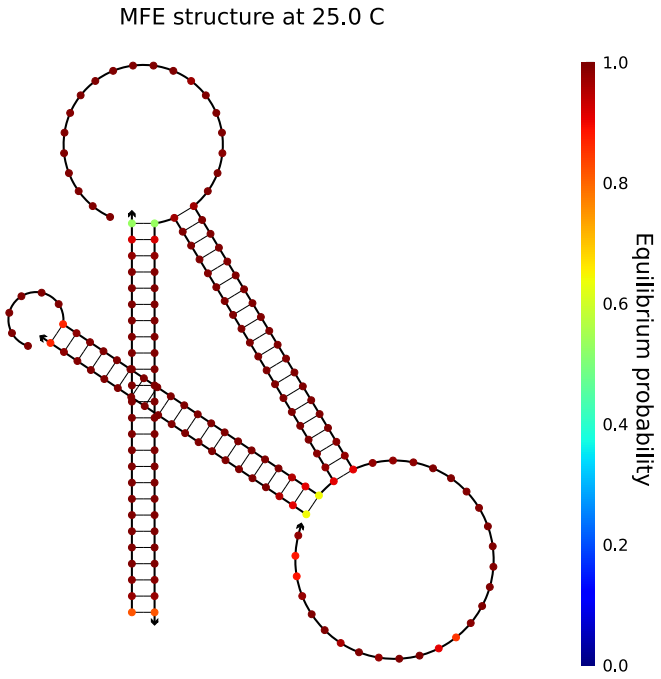

Figura 63. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{w \to x}$ (que corresponde al hecho *enamorado*(Maslow)) del programa E2.

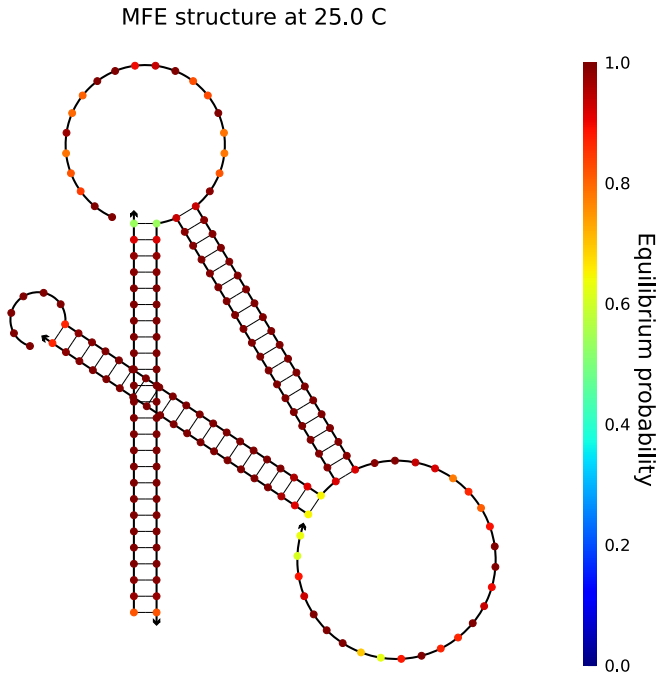

Figura 64. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{w \to y}$ (que corresponde al hecho *Adinerado*(Maslow)) del programa E2.

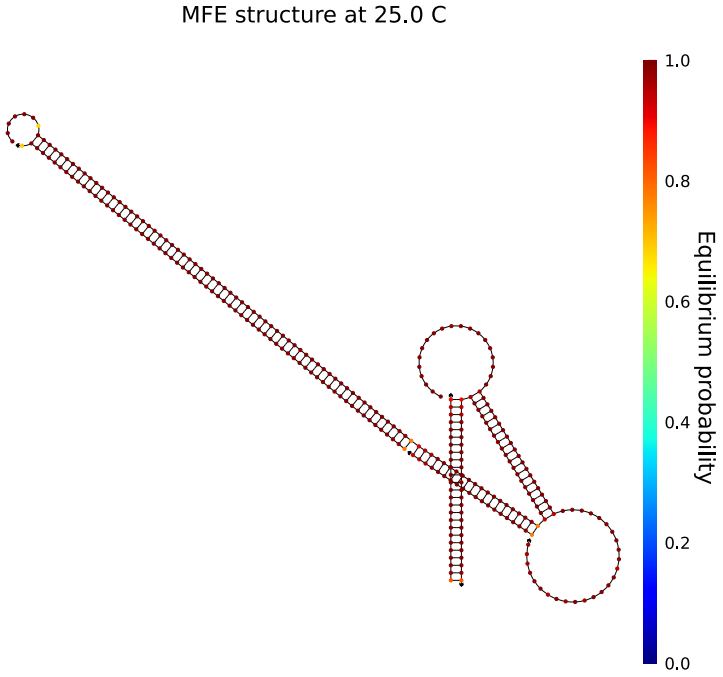

Figura 65. Estructura secundaria MFE con las probabilidades por base para la implicación $\mathbb{I}_{x \wedge y \to z}$ (que corresponde a la regla *Enamorado*(X)∧*Adinerado*(X)→ *Feliz*(X)) del programa E2.

Figura 66. Concentraciones en equilibrio arrojadas por NUPACK a 37°C. Las concentraciones de entrada fueron 10 nM para todas las hebras. El análisis produjo concentraciones marginales de moléculas basura para las seis implicaciones del programa E1. A 25°C, todas las hebras se consumieron en la molécula implicación deseada. Resultados similares se obtuvieron para el programa E2 (no incluidos).

Anexo C

Simulaciones en MULTISTRAND.

Tabla 8. Configuración de parámetros de MULTISTRAND. (*) Todas las reacciones en el escenario de "un paso" y las trayectorias para el paso de unión del escenario "por pasos" se simularon en el modo de simulación (simulation_mode) *First Step*. Para el resto de los pasos en el escenario "por pasos" se utilizó el modo *First Passage*.

Parámetro	Valor
simulation_mode	"First Step"/"First Passage"(*)
parameter_type	"Nupack"
substrate_type	"DNA"
rate_method	"Metropolis"
simulation_time	10.0 s
dangles	"Some"
temperature	25°C
rate_scaling	"Calibrated"

Tabla 9. Secuencias de los dominios utilizadas en las simulaciones MULTISTRAND.

Dominio	Secuencia
t_1	CCTATT
t_2	CATCG
GRC	CCCTTACCCAACCCAATCCC
p	CTAAAATCTACCATACTAAA
q	TTTCACTATAATCTTCATCC
x	CCCTTACCCAACCCAATCCC

```
s1 = Strand(name="F1_incumbent", domains=[p, Gen, p])
s2 = Strand(name="F1_substrate", domains=[t1.C, p.C, Gen.C, p.C])
s3 = Strand(name="Q1_queryP", domains=[p, Gen, p, t1])
initial_complex1 = Complex(strands=[s3], structure="....")
initial_complex2 = Complex(strands=[s1, s2], structure="((+.))")
complete_complex = Complex(strands=[s1], structure="****")
success_sc = StopCondition("SUCCESS", [(complete_complex, Dissoc_Macrostate, 0)])
failed_sc = StopCondition("FAILURE", [(initial_complex1, Dissoc_Macrostate, 0)])
o.start_state = [initial_complex1, initial_complex2]
o.stop_conditions = [success_sc, failed_sc]
```

Figura 67. Implementación de trayectorias para la reacción de un solo paso $\mathbb{QP}$.

```
s1 = Strand(name="R1_queryP_out", domains=[p, Gen, p, t1])
s2 = Strand(name="R1_right_cover", domains=[q])
s3 = Strand(name="R1_substrate", domains=[t1.C, q.C, Gen.C])
s4 = Strand(name="Q1_queryQ", domains=[q, Gen, q, t1])
initial_complex1 = Complex(strands=[s1, s2, s3], structure=".(..+(+.))")
initial_complex2 = Complex(strands=[s4], structure="....")
complete_complex = Complex(strands=[s1], structure="****")
success_sc = StopCondition("SUCCESS", [(complete_complex, Dissoc_Macrostate, 0)])
failed_sc = StopCondition("FAILURE", [(initial_complex2, Dissoc_Macrostate, 0)])
o.start_state = [initial_complex1, initial_complex2]
o.stop_conditions = [success_sc, failed_sc]
```

Figura 68. Implementación de trayectorias para la reacción de un solo paso $\mathbb{QI}$.

```
s1 = Strand(name="R1_queryP_out", domains=[p, Gen, p, t1])
s2 = Strand(name="R1_right_cover", domains=[q])
s3 = Strand(name="R1_substrate", domains=[t1.C, q.C, Gen.C])
s4 = Strand(name="F1_incumbent", domains=[p, Gen, p])
s5 = Strand(name="F1_substrate", domains=[t1.C, p.C, Gen.C, p.C])
initial_complex1 = Complex(strands=[s1, s2, s3], structure=".(..+(+.))")
initial_complex2 = Complex(strands=[s4, s5], structure="(((+.)))")
complete_complex1 = Complex(strands=[s4, s2, s3], structure="***+*+***")
complete_complex2 = Complex(strands=[s1, s5], structure="****+****")
success_sc1 = StopCondition("SUCCESS", [(complete_complex1, Dissoc_Macrostate, 0)])
success_sc2 = StopCondition("SUCCESS", [(complete_complex2, Dissoc_Macrostate, 0)])
failed_sc = StopCondition("FAILURE", [(initial_complex1, Dissoc_Macrostate, 0)])
o.start_state = [initial_complex1, initial_complex2]
o.stop_conditions = [success_sc1, success_sc2, failed_sc]
```

Figura 69. Implementación de trayectorias para la reacción de interferencia de un solo paso ℙ.

```
s1 = Strand(name="R1_queryP_out",  domains=[p, Gen, p, T1])
s2 = Strand(name="R1_right_cover", domains=[q])
s3 = Strand(name="R1_substrate",   domains=[t1.C, q.C, Gen.C])
s4 = Strand(name="R2_left_cover",  domains=[t2, p])
s5 = Strand(name="R2_queryX_out",  domains=[x, Gen, x, t1])
s6 = Strand(name="R2_right_cover", domains=[p])
s7 = Strand(name="R2_substrate",   domains=[t1.C, p.C, Gen.C, p.C, t2.C])
initial_complex1 = Complex(strands=[s1, s2, s3],     structure=".(..+(+.))")
initial_complex2 = Complex(strands=[s4, s5, s6, s7], structure="((+.(..+(+.))))")
complete_complex1 = Complex(strands=[s5, s2, s3],    structure="****+*+***")
complete_complex2 = Complex(strands=[s4, s1, s7],   structure="**+****+*****")
success_sc1 = StopCondition("SUCCESS", [(complete_complex1, Dissoc_Macrostate, 0)])
success_sc2 = StopCondition("SUCCESS", [(complete_complex2, Dissoc_Macrostate, 0)])
failed_sc = StopCondition("FAILURE", [(initial_complex1, Dissoc_Macrostate, 0)])
o.start_state = [initial_complex1, initial_complex2]
o.stop_conditions = [success_sc1, success_sc2, failed_sc]
```

Figura 70. Implementación de trayectorias para la reacción de interferencia de un solo paso 𝕀.

Anexo D

Modelo para simulación con reacciones de interferencia en DIZZY.

```
//           PARAMETER SETTINGS

scale\_factor   = 1000;                  // 1000 molecules per nM
NA\_en\_nM      = 6.0221415e14;\item
Volume          = scale\_factor/NA\_en\_nM;


n\_avogadro      = 6.0221415e23;
fact\_conversion = n\_avogadro * Volume;    // Deterministic to stochastic bimolecular rate constants

RepDomain       = 20;                   // Length of representative domain
GenDomain       = 20;                   // Length of generic domain
kf              = 3.5e+6;               // Toehold binding constant (1/(M x s))
SStep           = 400;                  // (Elementary step 3-way BM rate constant)
FWaySStep       = 1/2.7;                // (Elementary step 4-way BM rate constant)

kBinding        = kf/fact\_conversion;                          // Stochastic toehold binding constant
kuToe1          = 1.92;                                        // Toehold unbinding constant (t1)
kuToe2          = 1.35;                                        // Toehold unbinding constant (t2)
kBM\_RD         = SStep / (RepDomain^2);                       // 3-way BM constant (representative domain)
kBM\_GD         = SStep / (GenDomain^2);                       // 3-way BM constant (generic domain)
kBM             = SStep / ((RepDomain+GenDomain+RepDomain)^2);  // 3-way BM constant (BM segment)
FWay\_kBM\_GD       = FWaySStep / (GenDomain^2);               // 4-way BM constant (generic domain)

//          INITIAL CONCENTRATIONS

// PROPOSITIONS
// ToArmy (X)     :  A
// Strong (X)     :  B
// ToAcademy (X)  :  C
// Wise (X)       :  D
// X = Socrates   :  E
// X = Alexander  :  F
// X = Hercules   :  G

concent     = 20 * scale\_factor;

ImpBA       = concent;            //  Implication Strong(X)-->ToArmy(X)
ImpDC       = concent;            //  Implication Wise(X)-->ToAcademy(X)
ImpED       = concent;            //  Implication X=Socrates-->Wise(X)
ImpFD       = concent;            //  Implication X=Alexander-->Wise(X)
ImpFB       = concent;            //  Implication X=Socrates-->Strong(X)
ImpGB       = concent;            //  Implication X=Hercules-->Strong(X)
QryA        = concent;            //  Query ToArmy(X)?
QryC        =  0;                 //  Query ToAcademy(X)?
DetE        = concent;            //  Detector X=Socrates
DetF        = concent;            //  Detector X=Alexander
DetG        = concent;            //  Detector X=Hercules
FueA        = 2.0 * concent;      //  Fuel ToArmy(X)
FueB        = 4.0 * concent;      //  Fuel Strong(X)
FueC        = 2.0 * concent;      //  Fuel Tocademy(X)
FueD        = 4.0 * concent;      //  Fuel Wise(X)

//  ALL OTHER CONCENTRATIONS WERE INITAILIZED TO ZERO
```

Figura 71. Configuración de parámetros e inicialización en Dizzy.

```
//  ******************  KINETIC MODEL  ****************************

// ***************   VALID   REACTIONS     ************************

//                  RQ reactions

R01_step1f,     QryA + ImpBA -> IntBA1,     kBinding;
R01_step1r,     IntBA1 -> QryA + ImpBA,     kuToe1;
R01_step2,      IntBA1 -> IntBA2 + RCovA,   kBM_RD;
R01_step3,      IntBA2 -> Int1A1 + QryB,    kBM_GD;

R02_step1f,     QryC + ImpDC -> IntDC1,     kBinding;
R02_step1r,     IntDC1 -> QryC + ImpDC,     kuToe1;
R02_step2,      IntDC1 -> IntDC2 + RCovC,   kBM_RD;
R02_step3,      IntDC2 -> Int1C1 + QryD,    kBM_GD;

R03_step1f,     QryD + ImpED -> IntED1,     kBinding;
R03_step1r,     IntED1 -> QryD + ImpED,     kuToe1;
R03_step2,      IntED1 -> IntED2 + RCovD,   kBM_RD;
R03_step3,      IntED2 -> Int1D1 + QryE,    kBM_GD;

R04_step1f,     QryD + ImpFD -> IntFD1,     kBinding;
R04_step1r,     IntFD1 -> QryD + ImpFD,     kuToe1;
R04_step2,      IntFD1 -> IntFD2 + RCovD,   kBM_RD;
R04_step3,      IntFD2 -> Int1D1 + QryF,    kBM_GD;

R05_step1f,     QryB + ImpFB -> IntFB1,     kBinding;
R05_step1r,     IntFB1 -> QryB + ImpFB,     kuToe1;
R05_step2,      IntFB1 -> IntFB2 + RCovB,   kBM_RD;
R05_step3,      IntFB2 -> Int1B1 + QryF,    kBM_GD;

R06_step1f,     QryB + ImpGB -> IntGB1,     kBinding;
R06_step1r,     IntGB1 -> QryB + ImpGB,     kuToe1;
R06_step2,      IntGB1 -> IntGB2 + RCovB,   kBM_RD;
R06_step3,      IntGB2 -> Int1B1 + QryG,    kBM_GD;

//                  RQ (continued) reactions

R07_step1f,     Int1A1 -> Int1A2,           kBM_RD;
R07_step1r,     Int1A2 -> Int1A1,           kBM_RD;
R07_step2f,     Int1A2 -> Res1A + LCovA,    kuToe2;
R07_step2r,     Res1A + LCovA -> Int1A2,    kBinding;

R08_step1f,     Int1C1 -> Int1C2,           kBM_RD;
R08_step1r,     Int1C2 -> Int1C1,           kBM_RD;
R08_step2f,     Int1C2 -> Res1C + LCovC,    kuToe2;
R08_step2r,     Res1C + LCovC -> Int1C2,    kBinding;

R09_step1f,     Int1D1 -> Int1D2,           kBM_RD;
R09_step1r,     Int1D2 -> Int1D1,           kBM_RD;
R09_step2f,     Int1D2 -> Res1D + LCovD,    kuToe2;
R09_step2r,     Res1D + LCovD -> Int1D2,    kBinding;

R10_step1f,     Int1B1 -> Int1B2,           kBM_RD;
R10_step1r,     Int1B2 -> Int1B1,           kBM_RD;
R10_step2f,     Int1B2 -> Res1B + LCovB,    kuToe2;
R10_step2r,     Res1B + LCovB -> Int1B2,    kBinding;
```

Figura 72. Definición de reacciones QI en Dizzy.

```
//                  Fueling reactions

R11_step1f,     FueA + Res1A -> Int2A1,     kBinding;
R11_step1r,     Int2A1 -> FueA + Res1A,     kuToe2;
R11_step2f,     Int2A1 -> Int2A2,           kBM;
R11_step2r,     Int2A2 -> Int2A1,           kBM;
R11_step3f,     Int2A2 -> Res2A + QryA,     kuToe1;
R11_step3r,     Res2A + QryA -> Int2A2,     kBinding;

R12_step1f,     FueC + Res1C -> Int2C1,     kBinding;
R12_step1r,     Int2C1 -> FueC + Res1C,     kuToe2;
R12_step2f,     Int2C1 -> Int2C2,           kBM;
R12_step2r,     Int2C2 -> Int2C1,           kBM;
R12_step3f,     Int2C2 -> Res2C + QryC,     kuToe1;
R12_step3r,     Res2C + QryC -> Int2C2,     kBinding;

R13_step1f,     FueD + Res1D -> Int2D1,     kBinding;
R13_step1r,     Int2D1 -> FueD + Res1D,     kuToe2;
R13_step2f,     Int2D1 -> Int2D2,           kBM;
R13_step2r,     Int2D2 -> Int2D1,           kBM;
R13_step3f,     Int2D2 -> Res2D + QryD,     kuToe1;
R13_step3r,     Res2D + QryD -> Int2D2,     kBinding;

R14_step1f,     FueB + Res1B -> Int2B1,     kBinding;
R14_step1r,     Int2B1 -> FueB + Res1B,     kuToe2;
R14_step2f,     Int2B1 -> Int2B2,           kBM;
R14_step2r,     Int2B2 -> Int2B1,           kBM;
R14_step3f,     Int2B2 -> Res2B + QryB,     kuToe1;
R14_step3r,     Res2B + QryB -> Int2B2,     kBinding;

//                  QA reactions

R15_step1f,     QryE + DetE -> Int3E,       kBinding;
R15_step1r,     Int3E -> QryE + DetE,       kuToe1;
R15_step2,      Int3E -> SignalE + WDetE,   kBM;

R16_step1f,     QryF + DetF -> Int3F,       kBinding;
R16_step1r,     Int3F -> QryF + DetF,       kuToe1;
R16_step2,      Int3F -> SignalF + WDetF,   kBM;

R17_step1f,     QryG + DetG -> Int3G,       kBinding;
R17_step1r,     Int3G -> QryG + DetG,       kuToe1;
R17_step2,      Int3G -> SignalG + WDetG,   kBM;
```

Figura 73. Definición de reacciones de repostaje y ℚℙ en Dizzy.

```
//        UNPRODUCTIVE REACTIONS  (BINDING-UNBINDING)
//         --- RULES WITH QUERIES ---

U01_Bind,       QryB + ImpBA -> Int4BAB,    kBinding;
U01_Unbind,     Int4BAB -> QryB + ImpBA,    kuToe1;
U02_Bind,       QryC + ImpBA -> Int4BAC,    kBinding;
U02_Unbind,     Int4BAC -> QryC + ImpBA,    kuToe1;
U03_Bind,       QryD + ImpBA -> Int4BAD,    kBinding;
U03_Unbind,     Int4BAD -> QryD + ImpBA,    kuToe1;
U04_Bind,       QryE + ImpBA -> Int4BAE,    kBinding;
U04_Unbind,     Int4BAE -> QryE + ImpBA,    kuToe1;
U05_Bind,       QryF + ImpBA -> Int4BAF,    kBinding;
U05_Unbind,     Int4BAF -> QryF + ImpBA,    kuToe1;
U06_Bind,       QryG + ImpBA -> Int4BAG,    kBinding;
U06_Unbind,     Int4BAG -> QryG + ImpBA,    kuToe1;

U07_Bind,       QryA + ImpDC -> Int4DCA,    kBinding;
U07_Unbind,     Int4DCA -> QryA + ImpDC,    kuToe1;
U08_Bind,       QryB + ImpDC -> Int4DCB,    kBinding;
U08_Unbind,     Int4DCB -> QryB + ImpDC,    kuToe1;
U09_Bind,       QryD + ImpDC -> Int4DCD,    kBinding;
U09_Unbind,     Int4DCD -> QryD + ImpDC,    kuToe1;
U10_Bind,       QryE + ImpDC -> Int4DCE,    kBinding;
U10_Unbind,     Int4DCE -> QryE + ImpDC,    kuToe1;
U11_Bind,       QryF + ImpDC -> Int4DCF,    kBinding;
U11_Unbind,     Int4DCF -> QryF + ImpDC,    kuToe1;
U12_Bind,       QryG + ImpDC -> Int4DCG,    kBinding;
U12_Unbind,     Int4DCG -> QryG + ImpDC,    kuToe1;

U13_Bind,       QryA + ImpED -> Int4EDA,    kBinding;
U13_Unbind,     Int4EDA -> QryA + ImpED,    kuToe1;
U14_Bind,       QryB + ImpED -> Int4EDB,    kBinding;
U14_Unbind,     Int4EDB -> QryB + ImpED,    kuToe1;
U15_Bind,       QryC + ImpED -> Int4EDC,    kBinding;
U15_Unbind,     Int4EDC -> QryC + ImpED,    kuToe1;
U16_Bind,       QryE + ImpED -> Int4EDE,    kBinding;
U16_Unbind,     Int4EDE -> QryE + ImpED,    kuToe1;
U17_Bind,       QryF + ImpED -> Int4EDF,    kBinding;
U17_Unbind,     Int4EDF -> QryF + ImpED,    kuToe1;
U18_Bind,       QryG + ImpED -> Int4EDG,    kBinding;
U18_Unbind,     Int4EDG -> QryG + ImpED,    kuToe1;

U19_Bind,       QryA + ImpFD -> Int4FDA,    kBinding;
U19_Unbind,     Int4FDA -> QryA + ImpFD,    kuToe1;
U20_Bind,       QryB + ImpFD -> Int4FDB,    kBinding;
U20_Unbind,     Int4FDB -> QryB + ImpFD,    kuToe1;
U21_Bind,       QryC + ImpFD -> Int4FDC,    kBinding;
U21_Unbind,     Int4FDC -> QryC + ImpFD,    kuToe1;
U22_Bind,       QryE + ImpFD -> Int4FDE,    kBinding;
U22_Unbind,     Int4FDE -> QryE + ImpFD,    kuToe1;
U23_Bind,       QryF + ImpFD -> Int4FDF,    kBinding;
U23_Unbind,     Int4FDF -> QryF + ImpFD,    kuToe1;
U24_Bind,       QryG + ImpFD -> Int4FDG,    kBinding;
U24_Unbind,     Int4FDG -> QryG + ImpFD,    kuToe1;
```

Figura 74. Definición de reacciones no productivas en Dizzy.

```
U25_Bind,       QryA + ImpFB -> Int4FBA,    kBinding;
U25_Unbind,     Int4FBA -> QryA + ImpFB,    kuToe1;
U26_Bind,       QryC + ImpFB -> Int4FBC,    kBinding;
U26_Unbind,     Int4FBC -> QryC + ImpFB,    kuToe1;
U27_Bind,       QryD + ImpFB -> Int4FBD,    kBinding;
U27_Unbind,     Int4FBD -> QryD + ImpFB,    kuToe1;
U28_Bind,       QryE + ImpFB -> Int4FBE,    kBinding;
U28_Unbind,     Int4FBE -> QryE + ImpFB,    kuToe1;
U29_Bind,       QryF + ImpFB -> Int4FBF,    kBinding;
U29_Unbind,     Int4FBF -> QryF + ImpFB,    kuToe1;
U30_Bind,       QryG + ImpFB -> Int4FBG,    kBinding;
U30_Unbind,     Int4FBG -> QryG + ImpFB,    kuToe1;

U31_Bind,       QryA + ImpGB -> Int4GBA,    kBinding;
U31_Unbind,     Int4GBA -> QryA + ImpGB,    kuToe1;
U32_Bind,       QryC + ImpGB -> Int4GBC,    kBinding;
U32_Unbind,     Int4GBC -> QryC + ImpGB,    kuToe1;
U33_Bind,       QryD + ImpGB -> Int4GBD,    kBinding;
U33_Unbind,     Int4GBD -> QryD + ImpGB,    kuToe1;
U34_Bind,       QryE + ImpGB -> Int4GBE,    kBinding;
U34_Unbind,     Int4GBE -> QryE + ImpGB,    kuToe1;
U35_Bind,       QryF + ImpGB -> Int4GBF,    kBinding;
U35_Unbind,     Int4GBF -> QryF + ImpGB,    kuToe1;
U36_Bind,       QryG + ImpGB -> Int4GBG,    kBinding;
U36_Unbind,     Int4GBG -> QryG + ImpGB,    kuToe1;
//-------------------------------------------------------
//          --- DETECTORS WITH QUERIES ---
U37_Bind,       QryA + DetE -> Int5EA,      kBinding;
U37_Unbind,     Int5EA -> QryA + DetE,      kuToe1;
U38_Bind,       QryB + DetE -> Int5EB,      kBinding;
U38_Unbind,     Int5EB -> QryB + DetE,      kuToe1;
U39_Bind,       QryC + DetE -> Int5EC,      kBinding;
U39_Unbind,     Int5EC -> QryC + DetE,      kuToe1;
U40_Bind,       QryD + DetE -> Int5ED,      kBinding;
U40_Unbind,     Int5ED -> QryD + DetE,      kuToe1;
U41_Bind,       QryF + DetE -> Int5EF,      kBinding;
U41_Unbind,     Int5EF -> QryF + DetE,      kuToe1;
U42_Bind,       QryG + DetE -> Int5EG,      kBinding;
U42_Unbind,     Int5EG -> QryG + DetE,      kuToe1;

U43_Bind,       QryA + DetF -> Int5FA,      kBinding;
U43_Unbind,     Int5FA -> QryA + DetF,      kuToe1;
U44_Bind,       QryB + DetF -> Int5FB,      kBinding;
U44_Unbind,     Int5FB -> QryB + DetF,      kuToe1;
U45_Bind,       QryC + DetF -> Int5FC,      kBinding;
U45_Unbind,     Int5FC -> QryC + DetF,      kuToe1;
U46_Bind,       QryD + DetF -> Int5FD,      kBinding;
U46_Unbind,     Int5FD -> QryD + DetF,      kuToe1;
U47_Bind,       QryE + DetF -> Int5FE,      kBinding;
U47_Unbind,     Int5FE -> QryE + DetF,      kuToe1;
U48_Bind,       QryG + DetF -> Int5FG,      kBinding;
U48_Unbind,     Int5FG -> QryG + DetF,      kuToe1;
```

Figura 75. Definición de reacciones no productivas en Dizzy (continúa).

```
U49_Bind,       QryA + DetG -> Int5GA,      kBinding;
U49_Unbind,     Int5GA -> QryA + DetG,      kuToe1;
U50_Bind,       QryB + DetG -> Int5GB,      kBinding;
U50_Unbind,     Int5GB -> QryB + DetG,      kuToe1;
U51_Bind,       QryC + DetG -> Int5GC,      kBinding;
U51_Unbind,     Int5GC -> QryC + DetG,      kuToe1;
U52_Bind,       QryD + DetG -> Int5GD,      kBinding;
U52_Unbind,     Int5GD -> QryD + DetG,      kuToe1;
U53_Bind,       QryE + DetG -> Int5GE,      kBinding;
U53_Unbind,     Int5GE -> QryE + DetG,      kuToe1;
U54_Bind,       QryF + DetG -> Int5GF,      kBinding;
U54_Unbind,     Int5GF -> QryF + DetG,      kuToe1;
//-----------------------------------------------------
//         --- RESULTANT COMPLEXES T1 WITH QUERIES ---
U55_Bind,       QryB + Res2A -> Int6AB,     kBinding;
U55_Unbind,     Int6AB -> QryB + Res2A,     kuToe1;
U56_Bind,       QryC + Res2A -> Int6AC,     kBinding;
U56_Unbind,     Int6AC -> QryC + Res2A,     kuToe1;
U57_Bind,       QryD + Res2A -> Int6AD,     kBinding;
U57_Unbind,     Int6AD -> QryD + Res2A,     kuToe1;
U58_Bind,       QryE + Res2A -> Int6AE,     kBinding;
U58_Unbind,     Int6AE -> QryE + Res2A,     kuToe1;
U59_Bind,       QryF + Res2A -> Int6AF,     kBinding;
U59_Unbind,     Int6AF -> QryF + Res2A,     kuToe1;
U60_Bind,       QryG + Res2A -> Int6AG,     kBinding;
U60_Unbind,     Int6AG -> QryG + Res2A,     kuToe1;

U61_Bind,       QryA + Res2C -> Int6CA,     kBinding;
U61_Unbind,     Int6CA -> QryA + Res2C,     kuToe1;
U62_Bind,       QryB + Res2C -> Int6CB,     kBinding;
U62_Unbind,     Int6CB -> QryB + Res2C,     kuToe1;
U63_Bind,       QryD + Res2C -> Int6CD,     kBinding;
U63_Unbind,     Int6CD -> QryD + Res2C,     kuToe1;
U64_Bind,       QryE + Res2C -> Int6CE,     kBinding;
U64_Unbind,     Int6CE -> QryE + Res2C,     kuToe1;
U65_Bind,       QryF + Res2C -> Int6CF,     kBinding;
U65_Unbind,     Int6CF -> QryF + Res2C,     kuToe1;
U66_Bind,       QryG + Res2C -> Int6CG,     kBinding;
U66_Unbind,     Int6CG -> QryG + Res2C,     kuToe1;

U67_Bind,       QryA + Res2D -> Int6DA,     kBinding;
U67_Unbind,     Int6DA -> QryA + Res2D,     kuToe1;
U68_Bind,       QryB + Res2D -> Int6DB,     kBinding;
U68_Unbind,     Int6DB -> QryB + Res2D,     kuToe1;
U69_Bind,       QryC + Res2D -> Int6DC,     kBinding;
U69_Unbind,     Int6DC -> QryC + Res2D,     kuToe1;
U70_Bind,       QryE + Res2D -> Int6DE,     kBinding;
U70_Unbind,     Int6DE -> QryE + Res2D,     kuToe1;
U71_Bind,       QryF + Res2D -> Int6DF,     kBinding;
U71_Unbind,     Int6DF -> QryF + Res2D,     kuToe1;
U72_Bind,       QryG + Res2D -> Int6DG,     kBinding;
U72_Unbind,     Int6DG -> QryG + Res2D,     kuToe1;
```

Figura 76. Definición de reacciones no productivas en Dizzy (continúa).

```
U73_Bind,       QryA + Res2B -> Int6BA,     kBinding;
U73_Unbind,     Int6BA -> QryA + Res2B,     kuToe1;
U74_Bind,       QryC + Res2B -> Int6BC,     kBinding;
U74_Unbind,     Int6BC -> QryC + Res2B,     kuToe1;
U75_Bind,       QryD + Res2B -> Int6BD,     kBinding;
U75_Unbind,     Int6BD -> QryD + Res2B,     kuToe1;
U76_Bind,       QryE + Res2B -> Int6BE,     kBinding;
U76_Unbind,     Int6BE -> QryE + Res2B,     kuToe1;
U77_Bind,       QryF + Res2B -> Int6BF,     kBinding;
U77_Unbind,     Int6BF -> QryF + Res2B,     kuToe1;
U78_Bind,       QryG + Res2B -> Int6BG,     kBinding;
U78_Unbind,     Int6BG -> QryG + Res2B,     kuToe1;
//-------------------------------------------------------------
//         --- RESULTANT COMPLEXES T2 WITH FUELS ---
U79_Bind,       FueB + Res1A -> Int7AB,     kBinding;
U79_Unbind,     Int7AB -> FueB + Res1A,     kuToe2;
U80_Bind,       FueC + Res1A -> Int7AC,     kBinding;
U80_Unbind,     Int7AC -> FueC + Res1A,     kuToe2;
U81_Bind,       FueD + Res1A -> Int7AD,     kBinding;
U81_Unbind,     Int7AD -> FueD + Res1A,     kuToe2;

U82_Bind,       FueA + Res1C -> Int7CA,     kBinding;
U82_Unbind,     Int7CA -> FueA + Res1C,     kuToe2;
U83_Bind,       FueB + Res1C -> Int7CB,     kBinding;
U83_Unbind,     Int7CB -> FueB + Res1C,     kuToe2;
U84_Bind,       FueD + Res1C -> Int7CD,     kBinding;
U84_Unbind,     Int7CD -> FueD + Res1C,     kuToe2;

U85_Bind,       FueA + Res1D -> Int7DA,     kBinding;
U85_Unbind,     Int7DA -> FueA + Res1D,     kuToe2;
U86_Bind,       FueB + Res1D -> Int7DB,     kBinding;
U86_Unbind,     Int7DB -> FueB + Res1D,     kuToe2;
U87_Bind,       FueC + Res1D -> Int7DC,     kBinding;
U87_Unbind,     Int7DC -> FueC + Res1D,     kuToe2;

U88_Bind,       FueA + Res1B -> Int7BA,     kBinding;
U88_Unbind,     Int7BA -> FueA + Res1B,     kuToe2;
U89_Bind,       FueC + Res1B -> Int7BC,     kBinding;
U89_Unbind,     Int7BC -> FueC + Res1B,     kuToe2;
U90_Bind,       FueD + Res1B -> Int7BD,     kBinding;
U90_Unbind,     Int7BD -> FueD + Res1B,     kuToe2;
//-------------------------------------------------------------
//         --- RESULTANT COMPLEXES T2 WITH LEFT COVERS ---
U91_Bind,       LCovB + Res1A -> Int8AB,        kBinding;
U91_Unbind,     Int8AB -> LCovB + Res1A,        kuToe2;
U92_Bind,       LCovC + Res1A -> Int8AC,        kBinding;
U92_Unbind,     Int8AC -> LCovC + Res1A,        kuToe2;
U93_Bind,       LCovD + Res1A -> Int8AD,        kBinding;
U93_Unbind,     Int8AD -> LCovD + Res1A,        kuToe2;

U94_Bind,       LCovA + Res1C -> Int8CA,        kBinding;
U94_Unbind,     Int8CA -> LCovA + Res1C,        kuToe2;
U95_Bind,       LCovB + Res1C -> Int8CB,        kBinding;
U95_Unbind,     Int8CB -> LCovB + Res1C,        kuToe2;
U96_Bind,       LCovD + Res1C -> Int8CD,        kBinding;
U96_Unbind,     Int8CD -> LCovD + Res1C,        kuToe2;
```

Figura 77. Definición de reacciones no productivas en Dizzy (continúa).

```
U97_Bind,       LCovA + Res1D -> Int8DA,        kBinding;
U97_Unbind,     Int8DA -> LCovA + Res1D,        kuToe2;
U98_Bind,       LCovB + Res1D -> Int8DB,        kBinding;
U98_Unbind,     Int8DB -> LCovB + Res1D,        kuToe2;
U99_Bind,       LCovC + Res1D -> Int8DC,        kBinding;
U99_Unbind,     Int8DC -> LCovC + Res1D,        kuToe2;

U100_Bind,      LCovA + Res1B -> Int8BA,        kBinding;
U100_Unbind,    Int8BA -> LCovA + Res1B,        kuToe2;
U101_Bind,      LCovC + Res1B -> Int8BC,        kBinding;
U101_Unbind,    Int8BC -> LCovC + Res1B,        kuToe2;
U102_Bind,      LCovD + Res1B -> Int8BD,        kBinding;
U103_Unbind,    Int8BD -> LCovD + Res1B,        kuToe2;
```

Figura 78. Definición de reacciones no productivas en Dizzy (continúa).

```
// *************    CROSSTALK  REACTIONS  ********************

XTRR01_step1f,  ImpBA + ImpFB -> xtIntBAFB1,        kBinding;
XTRR01_step1r,  xtIntBAFB1 -> ImpBA + ImpFB,        kuToe1;
XTRR01_step2,   xtIntBAFB1 -> xtIntBAFB2 + RCovB,   kBM_RD;
XTRR01_step3,   xtIntBAFB2 -> xtImpFA + Int1B1,     FWay_kBM_GD;

XTRR02_step1f,  ImpBA + ImpGB -> xtIntBAGB1,        kBinding;
XTRR02_step1r,  xtIntBAGB1 -> ImpBA + ImpGB,        kuToe1;
XTRR02_step2,   xtIntBAGB1 -> xtIntBAGB2 + RCovB,   kBM_RD;
XTRR02_step3,   xtIntBAGB2 -> xtImpGA + Int1B1,     FWay_kBM_GD;

XTRR03_step1f,  ImpDC + ImpED -> xtIntDCED1,        kBinding;
XTRR03_step1r,  xtIntDCED1 -> ImpDC + ImpED,        kuToe1;
XTRR03_step2,   xtIntDCED1 -> xtIntDCED2 + RCovD,   kBM_RD;
XTRR03_step3,   xtIntDCED2 -> xtImpEC + Int1D1,     FWay_kBM_GD;

XTRR04_step1f,  ImpDC + ImpFD -> xtIntDCFD1,        kBinding;
XTRR04_step1r,  xtIntDCFD1 -> ImpDC + ImpFD,        kuToe1;
XTRR04_step2,   xtIntDCFD1 -> xtIntDCFD2 + RCovD,   kBM_RD;
XTRR04_step3,   xtIntDCFD2 -> xtImpFC + Int1D1,     FWay_kBM_GD;

XTRF01_step1f,  ImpED + DetE -> xtIntEDE1,          kBinding;
XTRF01_step1r,  xtIntEDE1 -> ImpED + DetE,          kuToe1;
XTRF01_step2f,  xtIntEDE1 -> xtIntEDE2,             kBM_RD;
XTRF01_step2r,  xtIntEDE2 -> xtIntEDE1,             kBM_RD;
XTRF01_step3f,  xtIntEDE2 -> xtIntEDE3,             FWay_kBM_GD;
XTRF01_step3r,  xtIntEDE3 -> xtIntEDE2,             FWay_kBM_GD;
XTRF01_step4,   xtIntEDE3 -> SignalE + WDetE,       kBM_RD;

XTRF02_step1f,  ImpFD + DetF -> xtIntFDF1,          kBinding;
XTRF02_step1r,  xtIntFDF1 -> ImpFD + DetF,          kuToe1;
XTRF02_step2f,  xtIntFDF1 -> xtIntFDF2,             kBM_RD;
XTRF02_step2r,  xtIntFDF2 -> xtIntFDF1,             kBM_RD;
XTRF02_step3f,  xtIntFDF2 -> xtIntFDF3,             FWay_kBM_GD;
XTRF02_step3r,  xtIntFDF3 -> xtIntFDF2,             FWay_kBM_GD;
XTRF02_step4,   xtIntFDF3 -> SignalF + WDetF,       kBM_RD;

XTRF03_step1f,  ImpFB + DetF -> xtIntFBF1,          kBinding;
XTRF03_step1r,  xtIntFBF1 -> ImpFB + DetF,          kuToe1;
XTRF03_step2f,  xtIntFBF1 -> xtIntFBF2,             kBM_RD;
XTRF03_step2r,  xtIntFBF2 -> xtIntFBF1,             kBM_RD;
XTRF03_step3f,  xtIntFBF2 -> xtIntFBF3,             FWay_kBM_GD;
XTRF03_step3r,  xtIntFBF3 -> xtIntFBF2,             FWay_kBM_GD;
XTRF03_step4,   xtIntFBF3 -> SignalF + WDetF,       kBM_RD;

XTRF04_step1f,  ImpGB + DetG -> xtIntGBG1,          kBinding;
XTRF04_step1r,  xtIntGBG1 -> ImpGB + DetG,          kuToe1;
XTRF04_step2f,  xtIntGBG1 -> xtIntGBG2,             kBM_RD;
XTRF04_step2r,  xtIntGBG2 -> xtIntGBG1,             kBM_RD;
XTRF04_step3f,  xtIntGBG2 -> xtIntGBG3,             FWay_kBM_GD;
XTRF04_step3r,  xtIntGBG3 -> xtIntGBG2,             FWay_kBM_GD;
XTRF04_step4,   xtIntGBG3 -> SignalG + WDetG,       kBM_RD;
```

Figura 79. Definición de reacciones de interferencia en Dizzy.

```
// ************  XTALK-PRODUCED  REACTIONS  *********************

X01_step1f,     QryA + xtImpFA -> IntFA1,       kBinding;
X01_step1r,     IntFA1 -> QryA + xtImpFA,       kuToe1;
X01_step2,      IntFA1 -> IntFA2 + RCovA,       kBM_RD;
X01_step3,      IntFA2 -> Int1A1 + QryF,        kBM_GD;

X02_step1f,     QryA + xtImpGA -> IntGA1,       kBinding;
X02_step1r,     IntGA1 -> QryA + xtImpGA,       kuToe1;
X02_step2,      IntGA1 -> IntGA2 + RCovA,       kBM_RD;
X02_step3,      IntGA2 -> Int1A1 + QryG,        kBM_GD;

X03_step1f,     QryC + xtImpEC -> IntEC1,       kBinding;
X03_step1r,     IntEC1 -> QryC + xtImpEC,       kuToe1;
X03_step2,      IntEC1 -> IntEC2 + RCovC,       kBM_RD;
X03_step3,      IntEC2 -> Int1C1 + QryE,        kBM_GD;

X04_step1f,     QryC + xtImpFC -> IntFC1,       kBinding;
X04_step1r,     IntFC1 -> QryC + xtImpFC,       kuToe1;
X04_step2,      IntFC1 -> IntFC2 + RCovC,       kBM_RD;
X04_step3,      IntFC2 -> Int1C1 + QryF,        kBM_GD;
```

Figura 80. Definición de reacciones ℚℙ producidas como consecuencia de las reacciones de interferencia.

```
// *********   UNPRODUCTIVE XTALK-PRODUCED REACTIONS  ********

XU01_Bind,      QryB + xtImpFA -> Int4FAB,  kBinding;
XU01_Unbind,    Int4FAB -> QryB + xtImpFA,  kuToe1;
XU02_Bind,      QryC + xtImpFA -> Int4FAC,  kBinding;
XU02_Unbind,    Int4FAC -> QryC + xtImpFA,  kuToe1;
XU03_Bind,      QryD + xtImpFA -> Int4FAD,  kBinding;
XU03_Unbind,    Int4FAD -> QryD + xtImpFA,  kuToe1;
XU04_Bind,      QryE + xtImpFA -> Int4FAE,  kBinding;
XU04_Unbind,    Int4FAE -> QryE + xtImpFA,  kuToe1;
XU05_Bind,      QryF + xtImpFA -> Int4FAF,  kBinding;
XU05_Unbind,    Int4FAF -> QryF + xtImpFA,  kuToe1;
XU06_Bind,      QryG + xtImpFA -> Int4FAG,  kBinding;
XU06_Unbind,    Int4FAG -> QryG + xtImpFA,  kuToe1;

XU07_Bind,      QryB + xtImpGA -> Int4GAB,  kBinding;
XU07_Unbind,    Int4GAB -> QryB + xtImpGA,  kuToe1;
XU08_Bind,      QryC + xtImpGA -> Int4GAC,  kBinding;
XU08_Unbind,    Int4GAC -> QryC + xtImpGA,  kuToe1;
XU09_Bind,      QryD + xtImpGA -> Int4GAD,  kBinding;
XU09_Unbind,    Int4GAD -> QryD + xtImpGA,  kuToe1;
XU10_Bind,      QryE + xtImpGA -> Int4GAE,  kBinding;
XU10_Unbind,    Int4GAE -> QryE + xtImpGA,  kuToe1;
XU11_Bind,      QryF + xtImpGA -> Int4GAF,  kBinding;
XU11_Unbind,    Int4GAF -> QryF + xtImpGA,  kuToe1;
XU12_Bind,      QryG + xtImpGA -> Int4GAG,  kBinding;
XU12_Unbind,    Int4GAG -> QryG + xtImpGA,  kuToe1;

XU13_Bind,      QryA + xtImpEC -> Int4ECA,  kBinding;
XU13_Unbind,    Int4ECA -> QryA + xtImpEC,  kuToe1;
XU14_Bind,      QryB + xtImpEC -> Int4ECB,  kBinding;
XU14_Unbind,    Int4ECB -> QryB + xtImpEC,  kuToe1;
XU15_Bind,      QryD + xtImpEC -> Int4ECD,  kBinding;
XU15_Unbind,    Int4ECD -> QryD + xtImpEC,  kuToe1;
XU16_Bind,      QryE + xtImpEC -> Int4ECE,  kBinding;
XU16_Unbind,    Int4ECE -> QryE + xtImpEC,  kuToe1;
XU17_Bind,      QryF + xtImpEC -> Int4ECF,  kBinding;
XU17_Unbind,    Int4ECF -> QryF + xtImpEC,  kuToe1;
XU18_Bind,      QryG + xtImpEC -> Int4ECG,  kBinding;
XU18_Unbind,    Int4ECG -> QryG + xtImpEC,  kuToe1;

XU19_Bind,      QryA + xtImpFC -> Int4FCA,  kBinding;
XU19_Unbind,    Int4FCA -> QryA + xtImpFC,  kuToe1;
XU20_Bind,      QryB + xtImpFC -> Int4FCB,  kBinding;
XU20_Unbind,    Int4FCB -> QryB + xtImpFC,  kuToe1;
XU21_Bind,      QryD + xtImpFC -> Int4FCD,  kBinding;
XU21_Unbind,    Int4FCD -> QryD + xtImpFC,  kuToe1;
XU22_Bind,      QryE + xtImpFC -> Int4FCE,  kBinding;
XU22_Unbind,    Int4FCE -> QryE + xtImpFC,  kuToe1;
XU23_Bind,      QryF + xtImpFC -> Int4FCF,  kBinding;
XU23_Unbind,    Int4FCF -> QryF + xtImpFC,  kuToe1;
XU24_Bind,      QryG + xtImpFC -> Int4FCG,  kBinding;
XU24_Unbind,    Int4FCG -> QryG + xtImpFC,  kuToe1;
```

Figura 81. Definición de reacciones no productivas producidas como consecuencia de las reacciones de interferencia.

Anexo E

Simulaciones a bajas concentraciones.

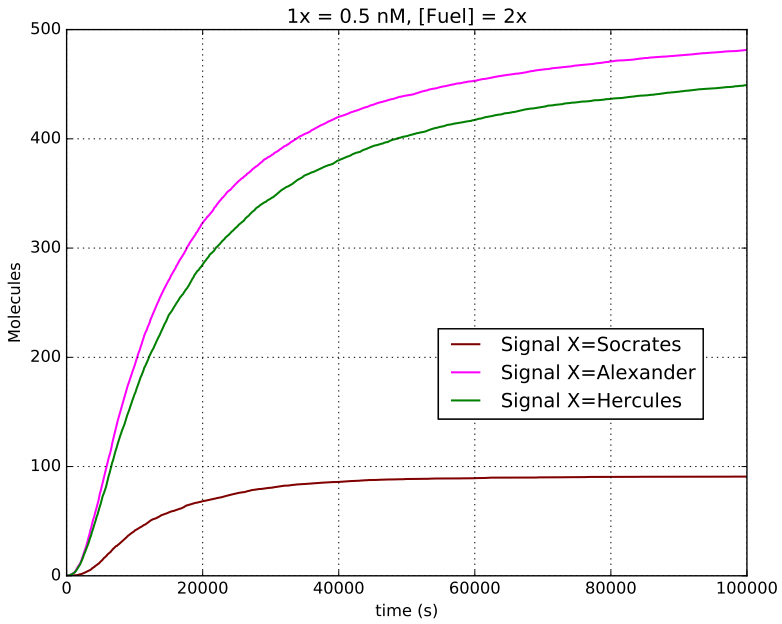

Figura 82. Cinética de las señales de salida en el programa E1 a bajas concentraciones ($1x = 0.5$ nM). Las moléculas combustible están presentes a $2x$. El tiempo considerado para la simulación está escalado en la misma proporción de la que se reduce la concentración.

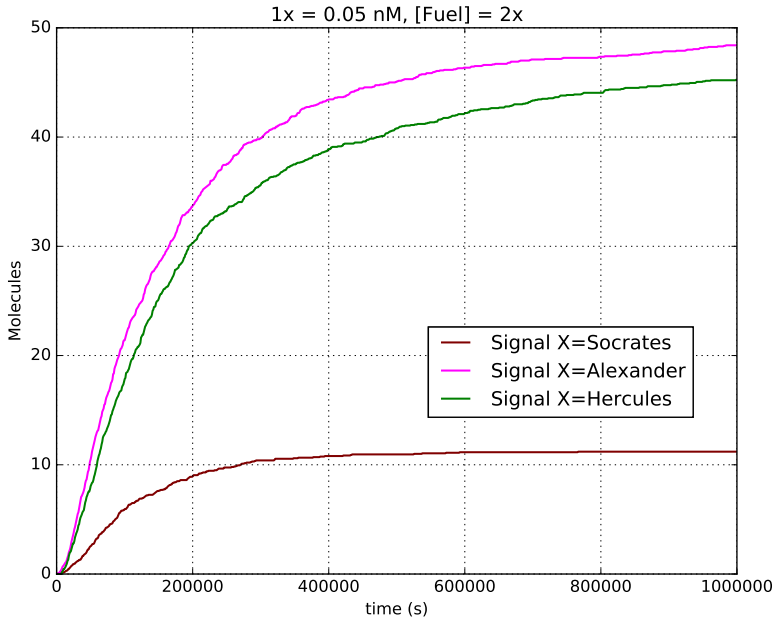

Figura 83. Cinética de las señales de salida en el programa E1 a bajas concentraciones ($1x = 0.05$ nM). Las moléculas combustible están presentes a $2x$. El tiempo considerado para la simulación está escalado en la misma proporción de la que se reduce la concentración.

Anexo F

Compuertas sube-baja.

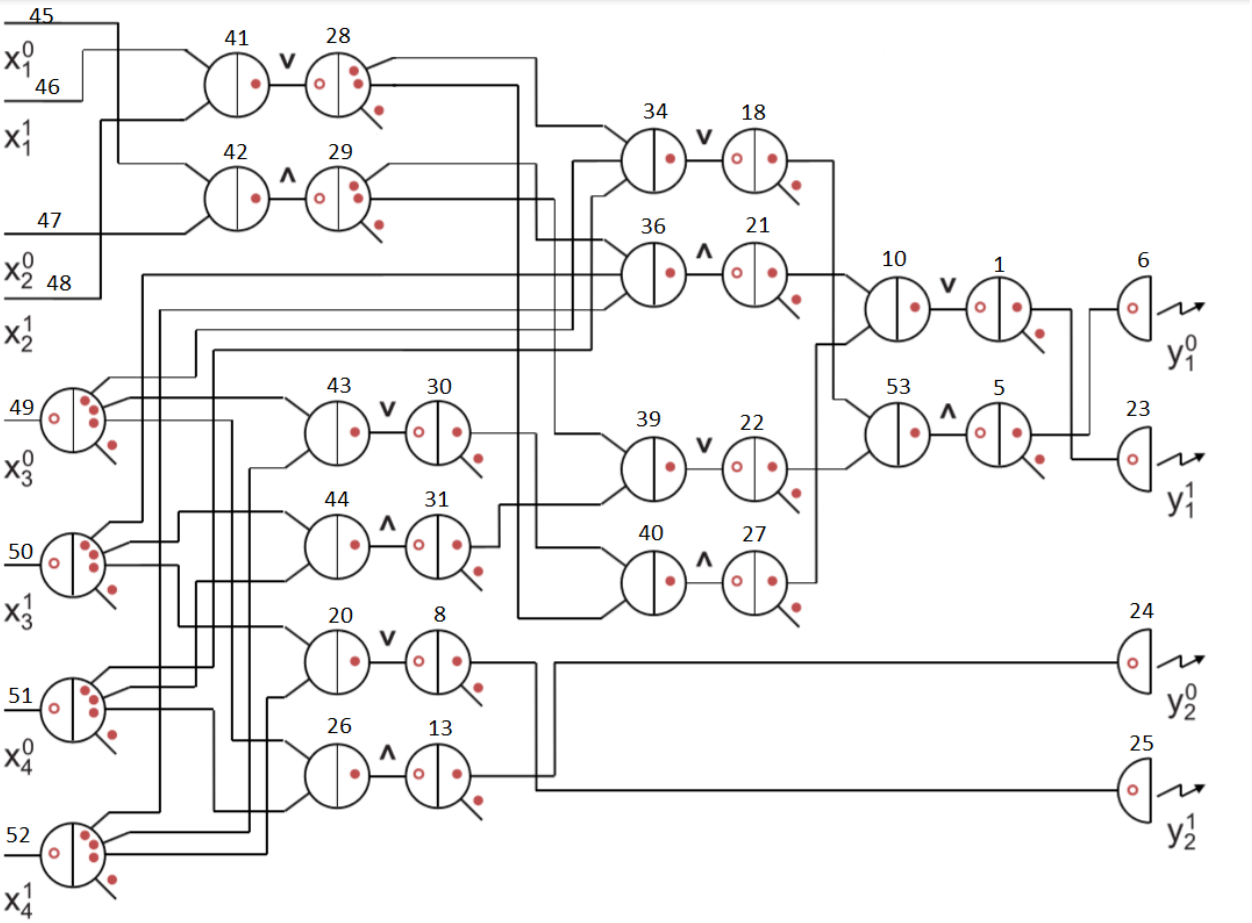

Figura 84. Representación abstracta del circuito para calcular la raíz cuadrada de un número de cuatro bits de Qian y Winfree (2011).