Centro Politécnico Superior
Universidad de Zaragoza

Trabajo Fin de Máster
Curso 2010-2011

Máster en Ingeniería de Sistemas e Informática

Análisis filogenético mediante lógica temporal y model checking

José Ignacio Requeno

Director:
José Manuel Colom
Departamento de Informática de Ingeniería de Sistemas, CPS

20 de Noviembre del 2010

Análisis filogenético mediante lógica temporal y model checking

Resumen

El *análisis filogenético* es la rama de la bioinformática que se encarga de estudiar la clasificación de un conjunto de especies distintas en función de su relación de proximidad evolutiva. La metodología de trabajo que sigue actualmente es muy "ad-hoc"; es decir, cuando especificamos una nueva propiedad biológica o añadimos otra especie al modelo necesitamos reimplementar el algoritmo de verificación para que considere los nuevos datos. Para solucionarlo, nuestra aproximación trata de sistematizar el método de trabajo en el análisis filogenético, además de reaprovechar conceptos y métodos consolidados en otras áreas de análisis de sistemas. Para ello proponemos el uso de "model checking", una técnica de verificación automática madura en el mundo de la verificación.

Nuestra aproximación se sustenta sobre tres pilares principales. Primero, el modelado de la dinámica evolutiva es análogamente equivalente al modelado de un sistema de transiciones, donde las características biológicas de las especies (ADN, fenotipo, etc) definen las variables de los *estados*; y la mutación o salto entre especies se corresponde con la ejecución de una *transición*. Segundo, el uso de lógicas temporales sistematiza el proceso de formulación y aporta formalismo a la especificación de propiedades filogenéticas. Por último, la verificación de estas fórmulas se realiza de manera automática por cualquiera de las herramientas informáticas existentes, devolviendo un contraejemplo en el caso de resultar falsas.

Además de la aportación teórica, hemos evaluado empíricamente las prestaciones del sistema sobre una herramienta software de model checking real: SMV. Esta tecnología ha demostrado ser viable cuando trabaja aisladamente con genes o fragmentos de ADNmt, ya que su tamaño no es excesivamente grande. Sin embargo, su coste temporal escala cuadráticamente con la longitud de las secuencias, mientras que el consumo de memoria crece linealmente hasta alcanzar más de 2,5 GBytes en el caso peor. Por estos motivos, en estudios filogenéticos futuros sobre el ADNmt completo (o el ADN nuclear, que es más grande) será necesario paralelizar y componer la solución a partir de los resultados parciales. También se ha propuesto una estructura de datos basada en los *diagramas de decisión* para manejar cadenas de ADNmt de forma más eficiente en memoria a cómo se realiza en la actualidad.

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Es de bien nacido ser agradecido. En primer lugar me gustaría agradecer a Elvira y José Manuel por su sabiduría y consejos, sin los cuales este trabajo nunca hubiera llegado a buen puerto. También, cómo no, a mi fiel compañero de armas Roberto, que me apadrinó desde el primer momento en mi retorno a la universidad por el sendero de la bioinformática. Recordaré además siempre en esta etapa a Goyo, por su visión numérica de la vida; y a Jorge, que con sus cafés, erizos y conversaciones distendidas consiguió amenizar mis momentos de descanso. Querría hacer una mención especial a mis padres, que me apoyaron en todo momento y sin los cuales no estaría aquí ahora.

Por último, mencionar explícitamente al I3A por su beca de iniciación a la investigación y a la DGA por su beca doctoral [17030 G/5423/480072/91002], que con su financiación me permitieron continuar con mi la labor investigadora.

Capítulo 1

Introducción

1.1. Contexto

La *patología mitocondrial*, provocada por la presencia de mutaciones en el ADN mitocondrial (en adelante, ADNmt), es una de las causas de enfermedades genéticas raras más prevalente. El ADNmt tiene importantes características diferenciales respecto al ADN nuclear.

En primer lugar, su dimensión más reducida y una mayor abundancia de copias por célula lo hacen más manejable. Además, su herencia es estrictamente matrilineal. Esto implica que no hay recombinación genética, simplificando considerablemente la trazabilidad de las mutaciones y los problemas derivados de la reordenación de genes por la interacción cromosómica. Por último, su alta tasa de mutación, mucho más elevada que la del ADN nuclear, está debida principalmente a su localización en un entorno reactivo como es el de la mitocondria, orgánulo encargado de la respiración aeróbica celular.

Gracias a estas tres peculiaridades, hacen al ADNmt objeto habitual de estudio en el ámbito bioquímico. En especial, este trabajo ha tomado el ADNmt como punto de partida dado el interés que suscita, por todos los comentarios anteriores, entre miembros de la Facultad de Veterinaria (con quienes se ha colaborado en el desarrollo de este trabajo).

Una de las herramientas con las que cuentan los científicos para estudiar el ADNmt es el análisis filogenético. El *análisis filogenético* es una rama de la bioinformática que se ocupa de la aplicación de modelos y métodos computacionales para la construcción de un árbol (o grafo) que plasme las relaciones evolutivas de un conjunto de genes, individuos o especies (también llamados taxones).

La sucesiva incorporación de métodos formales para caracterizar conceptos claves, como la distancia evolutiva, ha proporcionado resultados valiosos.

La representación numérica de relaciones entre conjuntos de poblaciones o individuos ha sido explorada por métodos arborescentes basados en distancias (neigbour joining), y por métodos basados en minimizar el número de cambios de características biológicas (parsimonia, verosimilitud, inferencia Bayesiana) [15].

Desde un punto de vista biológico, el árbol (o potenciales árboles) evolutivo que obtenemos como resultado tiene una aplicación directa para inferir propiedades biológicas interesantes: identificar mutaciones patógenas causantes de graves enfermedades (pues estarán en hojas terminales que no se han propagado hasta el presente), deducir el histórico de las migraciones poblacionales, estimar el momento de divergencia entre dos especies o incluso encontrar regiones codificantes de gran importancia.

1.2. Motivación

Para averiguar propiedades biológicas como las comentadas anteriormente es importante detectar patrones característicos en las secuencias de ADN (propiedades como la conservación o covariación de nucleótidos y aminoácidos [27]), o analizar la organización en grupos cladísticos (familias de especies).

Sin embargo, actualmente el análisis filogenético carece de cierta flexibilidad en el sentido de que el modelo evolutivo, la especificación de propiedades y su proceso de verificación no están completamente desacoplados. Esto conduce a una forma rígida de trabajo, donde la comprobación de una nueva propiedad, la modificación del modelo o la optimización del proceso de verificación deriva en el rediseño desde cero de todo el sistema software.

Además, la inherente naturaleza temporal de los datos filogenéticos sugiere la posibilidad de introducir nuevos métodos formales novedosos, capaces de mejorar esta falta de flexibilidad en los modelos convencionales, para incorporar características como: predicción futura, embeber información del pasado y combinación de reglas evolutivas a varios niveles de abstracción.

1.3. Objetivos

La búsqueda de un entorno de trabajo flexible, que desacople aún mejor el modelado y especificación de la tarea de verificación, ha sido nuestro principal objetivo y el eje central del trabajo realizado. De esta forma, los investigadores podrán enfocar sus esfuerzos exclusivamente al modelado de la evolución y la especificación de propiedades, mientras que los algoritmos genéricos de verificación se encargarán del resto.

Con este propósito sugerimos el uso de model checking para el análisis filogenético. El *model checking* es una técnica madura de verificación automática que, dado el modelo de estados finitos de un sistema y una propiedad formal (descrita en lógica temporal), sistemáticamente comprueba si dicha propiedad se cumple para algún estado particular del modelo. El proceso de model checking consta de tres fases: modelado con un lenguaje de descripción (formalizando tanto el sistema como las propiedades), ejecución (comprobación de la validez de la propiedad con un paquete informático de model checking) y análisis de los resultados (interpretación de los contraejemplos si la propiedad no se cumple).

Además de haberse aplicado con éxito en el mundo de la informática industrial para el modelado y verificación de sistemas [16], también se ha utilizado anteriormente en el contexto de biología computacional. Mayoritariamente, este paradigma se ha aplicado en campos donde necesitaban analizar propiedades cuantitativas a lo largo del tiempo, como por ejemplo: redes de interacción celular y molecular [26], predicción del plegado dinámico de proteínas [19] o la cinética del ciclo celular [30].

Nuestra aproximación es novedosa en el ámbito del análisis filogenético al unificarlo con el formalismo de model checking. Este trabajo se sustenta sobre tres pilares principales. En primer lugar, el modelado de la dinámica evolutiva es análogamente equivalente al modelado de un sistema de transiciones o máquina de estados. En segundo lugar, el uso de fórmulas de lógica temporal aporta formalismo a la especificación de propiedades filogenéticas. Por último, la verificación de estas fórmulas se realiza de manera automática por cualquiera de las herramientas informáticas existentes.

Hay que recordar que la calidad de la verificación usando estas técnicas depende únicamente de lo refinado que sea nuestro modelo inicial del sistema, lo cual es una de sus grandes virtudes. Adicionalmente, el proceso de verificación, al estar automatizado, no requiere de una gran interacción con el usuario y es una tecnología de fácil uso [8] para la cual existe una amplia amalgama de herramientas software de verificación automática [16].

Como resumen de las ventajas principales que emergen del estudio de propiedades filogenéticas con esta aproximación, destacamos que pueden considerarse diferentes modelos de evolución (tanto estructuras arborescentes como redes o grafos). También, gracias al uso de un lenguaje de consulta común, propiedades complejas podrán componerse incrementalmente a partir de otras más sencillas, lo que mejora la modularidad, legibilidad y eficiencia al reaprovechar resultados anteriores. Finalmente, en los casos en los que la verificación devuelva como resultado que una propiedad es incorrecta, incluye además un contraejemplo que la invalida y que permitirá realimentar a los filogenetistas y refinar iterativamente sus hipótesis iniciales (o incluso

diseñar otras nuevas).

Por tanto, el trabajo a seguir está estructurado en el siguiente esquema:

- El estudio de distintas estrategias de representación informática del árbol filogenético como un sistema de transiciones.
- El estudio, y en su caso adaptación, de los algoritmos de model checking y las estructuras de datos con las que trabajan para tratar eficientemente las cadenas de ADN presentes en los nodos del sistema de transiciones.
- El estudio de las herramientas de model checking existentes con el fin de su adaptación para el análisis filogenético.
- El estudio experimental de las prestaciones de la aproximación propuesta y su comparación con los métodos existentes en el ámbito del análisis filogenético tradicional.

1.4. Organización del documento

Este texto constituye el trabajo de fin de máster en "Ingeniería de Sistemas e Informática" de José Ignacio Requeno. Se enmarca dentro del campo de la bioinformática. El cuerpo principal (capítulos 2 y 3) está basado en un artículo de investigación publicado en el *IEEE International Workshop on Mining and Management of Biological and Health Data* dentro de las jornadas del *IEEE International Conference on Bioinformatics & Biomedicine 2010* [10]. Dicho artículo ha sido desarrollado por el grupo ZARAMIT de la Universidad de Zaragoza, dentro del cual se incluye el presente autor. En esta primera parte se recogen las principales aportaciones conceptuales e ideas sobre el uso de lógica temporal y model checking para el análisis filogenético.

Más detalladamente, en el capítulo 2 se realiza una panorámica general. Ahí se desarrolla el model checking y se expone su relación con la filogenética: las filogenias como modelo dinámico de evolución y su interpretación como sistema de transiciones, las especificaciones filogenéticas en términos de lógica computacional (CTL), y la verificación del sistema mediante model checking. Después, el capítulo 3 ejemplifica el modelado en lógica temporal de dos de las clases de propiedades filogenéticas estructurales más importantes, como es el caso de la clasificación cladística y la relación entre subsegmentos de las secuencias.

Posteriormente, en el capítulo 4 extendemos la propuesta inicial, explicando cómo se implementa el sistema sobre una herramienta software real

de model checking, junto con un pequeño estudio de viabilidad, las limitaciones existentes y una serie de consejos y alternativas para escalar el entorno. El conjunto de datos escogido para las pruebas de escalabilidad consiste en genes del ADNmt, tal y como se comentaba en el contexto, dado el interés que suscita entre miembros de la facultad de veterinaria.

Finalmente, antes de las conclusiones y trabajo futuro (capítulo 6), el capítulo 5 sugiere la modificación de una estructura de datos presente en la mayoría de los model checkers con el fin de adaptarla al caso particular que nos atañe: la representación eficiente de largas cadenas. La aplicación de esa estructura de datos (los *diagramas de decisión binarios* [11]) a las cadenas de texto es innovadora, y se planteó inicialmente como trabajo de evaluación para el curso de "Verificación asistida por computador (CAV)", asignatura perteneciente al máster que he cursado. Aunque perfectamente podría haberse relegado a los apéndices por motivos de espacio en el documento, hemos preferido incluirlo dentro del cuerpo principal de la memoria para darle un aspecto autocontenido.

Capítulo 2

Uniendo mundos: Filogenética y Model Checking

2.1. Contexto

Los *sistemas de transiciones* [8, Def. 2.1] son potentes modelos formales para el estudio de sistemas concurrentes. Dado un modelo del sistema y una lógica adecuada para guiar el razonamiento, es posible conseguir una verificación automática y exhaustiva de nuestras propiedades a comprobar. En este capítulo, expondremos cómo las filogenias pueden interpretarse intuitivamente como dichos modelos y representarse fácilmente como tales. Comenzaremos por identificar las claves de la evolución que permiten asimilarlo a un sistema de transiciones. Después, continuaremos con el estudio de su naturaleza temporal y la formulación lógica, para concluir con una visión general del proceso de verificación bajo el paradigma del model checking.

2.2. Evolución como un Sistema de Transiciones

Desde un punto de vista abstracto, las filogenias representan modelos parciales de evolución para conjuntos de seres vivos. Generalmente, pueden modelarse como grafos dirigidos, aunque una estructura en forma de árbol filogenético normalmente será suficiente para la mayoría de nuestras necesidades, y será por tanto la que adoptemos a lo largo del documento.

Los árboles ofrecen un modelo realista de evolución agregada, en la que cada vértice representa una población de individuos relacionados que procrean entre sí y comparten una herencia genética compatible común (ADN). Los

procesos de transformación que modifican la información hereditaria dan lugar a la aparición de nuevas especies, que quedan reflejadas en el nuevo grafo mediante relaciones paterno-filiares con arcos dirigidos que parten desde los padres hacia los nuevos hijos. Recalcamos que ni el tiempo ni la ordenación de las poblaciones descendientes forman parte explícita del modelo (aunque el tiempo está implícito si lo consideramos como una sucesión secuencial de eventos), lo que es consistente con la semántica del árbol que se encuentra a continuación.

Definición 1 (Árbol etiquetado enraizado) Sea Σ un alfabeto finito y l un número natural. Un árbol filogenético sobre Σ^l puede representarse como una tupla $P = (T, r, D)$, donde:

- $T = (V, E)$ es un árbol (grafo),
- $r \in V$ es la raíz, y
- $D : V \to \Sigma^l$ es una función de diccionario que etiqueta cada vértice del árbol con la secuencia asociada a su taxón.

Los árboles se construyen habitualmente a partir de conjuntos finitos de palabras con longitud uniforme, generados como resultado de alguno de los algoritmos de (multi)alineamiento existentes. Esas palabras, que representan los datos de los taxones actuales, necesitan una correspondencia biyectiva con el conjunto de hojas del árbol. No obstante, también pueden considerarse otras codificaciones más generales como redes evolutivas (grafos).

Además, pueden incorporarse o eliminarse restricciones al modelo filogenético en cualquier momento según las necesidades. En cualquier caso, debería quedar clara de antemano la naturaleza de los árboles filogenéticos como *sistemas reactivos*: están compuestos por procesos independientes (individuales o poblacionales) que interaccionan profundamente con su entorno y se extienden indefinidamente a lo largo del tiempo. Su característica más importante es la noción de *estado*, es decir, la posibilidad de heredar parcial o totalmente información genética (aunque no siempre es invariante debido a las mutaciones).

En consecuencia, es posible modelar y verificar sistemas biológicos evolutivos. Para ello, contamos con estructuras de datos bastante interesantes para representar los sistemas de transiciones.

Definición 2 (Estructura de Kripke) Sea AP un conjunto de *proposiciones atómicas*, por ejemplo, predicados booleanos que describan propiedades observables de un estado. Una estructura de Kripke sobre

AP es un sistema de transiciones finito representado por una tupla $M = (S, S_0, R, L)$, donde:

- S es un conjunto finito de estados,
- $S_0 \subseteq S$ es el conjunto de estados iniciales,
- $R \subseteq S \times S$ es la relación de transición total entre estados, es decir, para cada estado $s \in S$, existe un $t \in S$ tal que $(s, t) \in R$, y
- $L : S \to 2^{AP}$ es la función de etiquetado que asocia a cada estado con un subconjunto de proposiciones atómicas que son ciertas en él.

Una estructura de Kripke modela un sistema que es capaz de ejecutar un número infinito de comportamientos o *caminos*, los cuales son secuencias infinitas de estados sucesivos $\pi = s_0 s_1 s_2 \ldots$ tales que $s_0 \in S_0$ y $(s_i, s_{i+1}) \in S, i \in \mathbb{N}$. El conjunto de todas las posibles ejecuciones (caminos) de la estructura puede desplegarse en su *árbol de computación*.

Aquí nos centramos especialmente en las estructuras de Kripke que representan un cierto árbol como (idealmente la verdadera) *computación* del proceso evolutivo; o más generalmente, el conjunto de posibles computaciones que resultan de las hipótesis iniciales de evolución. Las relaciones entre estados y proposiciones atómicas, así como entre las ramas del árbol y las transiciones entre estados, son una pieza esencial. Por ello, debemos aclarar algunas nociones interesantes:

- Idealmente, el estado de un proceso o población queda inequívocamente identificado por su secuencia de ADN. Dichas secuencias también determinan las proposiciones atómicas que conforman la base de las proposiciones lógicas: por ejemplo, la presencia o ausencia de un determinado carácter del alfabeto en una posición dada. En determinados casos especiales, donde distintos vértices del árbol comparten una misma secuencia, es posible enriquecer la información de sus estados con información extra (como un identificador numérico) para preservar su identidad única.
- Una vez hemos establecido una correspondencia uno a uno entre los vértices del árbol y sus estados, todavía queda por resolver la semántica de los caminos infinitos en las estructuras de Kripke: un árbol no deja de ser una fotografía instantánea local de la potencial ejecución infinita del sistema en el presente. Para solucionarlo y fijar determinados vértices como terminales, es suficiente con añadir autobucles a sus estados (aunque también existen relajaciones en las estructuras de Kripke para que devuelvan un camino finito [7]).

En este punto, podemos definir una *estructura de branching-time* para los árboles filogenéticos, y que servirá de base para interpretar las fórmulas de lógica temporal que emplearemos para expresar propiedades del árbol. La fórmula de estado más sencilla determina si el estado actual está asociado con la secuencia $s = \sigma_1\sigma_2 \ldots \sigma_l \in \Sigma^l$ (o con una subcadena de la secuencia, o con un conjunto de secuencias). Las secuencias pueden manipularse simbólicamente (como si fueran conjuntos de datos) mediante una agregación de sus partes. En términos lógicos, quedaría representado como:

$$s \equiv \bigwedge_{i=1}^{l} s\left[i\right] = \sigma_i \tag{2.1}$$

Definición 3 (Filogenia Branching-time) Un árbol (por Def. 1) $P = (T, r, D)$ está inequívocamente definido por una estructura de Kripke análoga $M_P = (V, \{r\}, R_P, L_P)$, donde:

- R_P es la relación de transición compuesta por el conjunto de arcos del árbol (dirigidos desde la raíz r) más los autobucles en las hojas: $R_P = E \cup \{(v, v) : \nexists\,(v, w) \in E \wedge v, w \in V\}$, y
- L_P es una función estándar de etiquetado definida por AP_P, donde un estado v mapeado como $D\left(v\right) = \sigma_1\sigma_2 \ldots \sigma_l$ satisface la familia de propiedades $s\left[i\right] = \sigma_i, 1 \leq i \leq l$, además de otras necesarias para mantener la unicidad lógica de cada estado (por ejemplo, un identificador numérico propio).

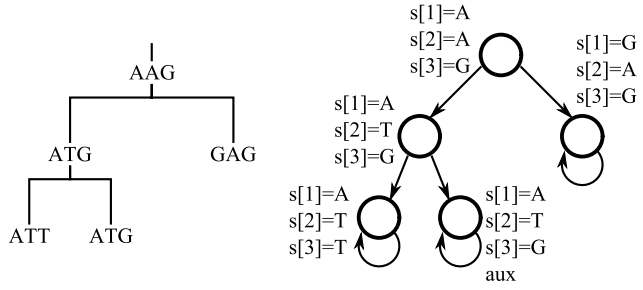

Figura 2.1: Traducción de un árbol a una estructura de Kripke

La figura 2.1 ilustra este proceso de traducción desde un árbol (filogenético) etiquetado (Def. 1) a una estructura de Kripke (Def. 2). Recalcamos que pueden considerarse fácilmente otros modelos de evolución (redes evolutivas, etc) ajustando las definiciones Defs. 1–3 según se requiera.

2.3. Lógica Temporal como un Lenguaje de Especificación

La lógica temporal es un sistema formal que permite la representación y manipulación de proposiciones lógicas cualificadas en términos de tiempo [21]. En el contexto de los sistemas de transición, se usan para definir propiedades en secuencias de transiciones entre estados de un sistema previamente definido según la abstracción correspondiente (en nuestro caso, un tipo específico de estructura de Kripke). Por ejemplo, las propiedades pueden indicar si desde un determinado punto es posible alcanzar un estado en particular, o si una propiedad siempre se cumple.

Las lógicas temporales se clasifican en dos grandes familias dependiendo de cómo tratan las secuencias de eventos: mientras que las *lógicas linear-time* engloban las lógicas que trabajan únicamente con caminos individuales $\pi = s_0 s_1 s_2 \ldots$, las *lógicas branching-time* tienen en cuenta el conjunto de posibles opciones desde un estado (y por tanto, razonando sobre el total del árbol de computación). *Computational Tree Logic* (CTL) es el máximo exponente de este último tipo de familias y el que ha sido ampliamente adoptado por la comunidad de model checking [13]. Dado que las filogenias representan procesos evolutivos que por naturaleza son esencialmente bifurcaciones, las lógicas de tipo branching-time en general, y CTL en particular, son adecuadas para su correcta descripción.

CTL reinterpreta los cuantificadores de la lógica de primer orden como *cuantificadores de camino*. Esos cuantificadores expresan el cumplimiento de una propiedad a lo largo de todos los caminos de computación (**A**), o al menos a lo largo de uno de sus caminos (**E**). Estos dos cuantificadores deben estar inmediatamente cualificados por alguno de los cinco *operadores temporales*, de los cuales tres expresan la satisfacción de una propiedad eventualmente en cualquier momento del futuro (**F**), para todos los instantes (**G**), o en el estado siguiente (**X**). Los otros dos, son construcciones condicionales en donde la propiedad precedente se cumple hasta que el consecuente se haga verdad (**U**); o la consecuente se satisface hasta el momento en el que la precedente lo haga (si lo hace, y además ambas se cumplen a la vez en el estado de transición) (**R**). Una gramática y semántica completa de las fórmulas CTL puede definirse como un subconjunto representativo de operadores lógicos de la manera siguiente; (Fig. 2.2 ilustra la semántica de todas ellas como un conjunto.)

Definición 4 (Árbol Filogenético Lógico) Una fórmula arbitraria de lógica temporal ϕ se define a partir de la siguiente gramática mínima, donde

$p \in AP$:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{EX}\,(\phi) \mid \mathbf{EG}\,(\phi) \mid \mathbf{E}\,[\phi \mathbf{U} \phi] \tag{2.2}$$

Las fórmulas se chequean sobre una estructura M considerando todos los caminos π que se originan desde un determinado estado inicial s_0. $M, s_0 \vDash \phi$ significa que s_0 satisface la propiedad ϕ. La semántica para la verificación de fórmulas bien formadas es la siguiente (con $\pi = s_0 s_1 s_2 \ldots$):

- $M, s_0 \vDash p \Leftrightarrow p \in L\,(s_0)$,
- $M, s_0 \vDash \neg\phi \Leftrightarrow M, s_0 \nvDash \phi$,
- $M, s_0 \vDash \phi \vee \psi \Leftrightarrow M, s_0 \vDash \phi$ o $M, s_0 \vDash \psi$,
- $M, s_0 \vDash \mathbf{EX}\,(\phi) \Leftrightarrow \exists \pi : M, s_1 \vDash \phi$,
- $M, s_0 \vDash \mathbf{EG}\,(\phi) \Leftrightarrow \exists \pi : M, s_i \vDash \phi, \forall i \in \mathbb{N}$, y
- $M, s_0 \vDash \mathbf{E}\,[\phi \mathbf{U} \psi] \Leftrightarrow \exists \pi, i \in \mathbb{N} : M, s_i \vDash \psi$ y $M, s_j \vDash \phi, 0 \leq j < i$.

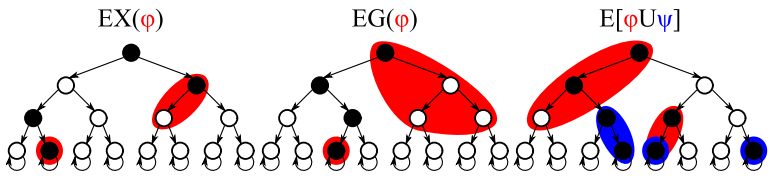

Figura 2.2: Evaluación de operadores lógicos temporales

Una fórmula CTL ϕ representa una propiedad que debe verificarse en determinados estados del árbol de computación. En este contexto, un sistema M satisface ϕ si y sólo si todos y cada uno de los estados iniciales cumplen: $\bigwedge_{s_0 \in S_0} M, s_0 \vDash \phi$. Como licencia particular y por temas de legibilidad, adaptaremos la notación de los operadores condicionales para reescribirlos como $\mathbf{E}\,[\phi \mathbf{U} \psi] \equiv \{\phi\}\,\mathbf{EU}\,\{\psi\}$.

Una lógica así definida permite la expresión formal de propiedades genéricas relacionadas con la evolución biológica de secuencias, además de su eventual verificación automática.

2.4. Model Checking como un Entorno de Inferencia

El principio que opera bajo el model checking es simple: la ejecución de software de verificación en un ordenador para comprobar la corrección de

un sistema. Para lograrlo, como entrada el software necesita un modelo del sistema junto con la especificación de sus requerimientos, ambos provistos por el usuario. En el supuesto de que alguna de las especificaciones propuestas no se cumpla, el software devuelve mediante contraejemplos los escenarios en los que se infringe la propiedad. Sin embargo, los algoritmos básicos no pueden gestionar por sí solos la explosión del espacio de estados, por lo que se necesita una manipulación simbólica de los sistemas y las fórmulas lógicas [12].

Y lo que es aún más importante, el uso de las técnicas de model checking es completamente transparente respecto del sistema que estamos verificando, dado que son independientes del dominio de aplicación y de su interpretación. Esto significa que los filogenetistas pueden redirigir los esfuerzos que antes dedicaban a temas de implementación para enfocarlos al modelado (antes de ejecutar el model checking, estableciendo las propiedades deseables u obligatorias del sistema) y el análisis de los resultados (después del model checking, observando el éxito o fracaso del proceso y estudiando los parámetros devueltos).

Capítulo 3

Modelado de propiedades estructurales

3.1. Contexto

Este capítulo está dedicado a ilustrar la metodología a seguir para modelizar propiedades biológicas filogenéticas empleando el entorno lógico aquí presentado. Generalmente, una propiedad no trivial puede descomponerse en otras más sencillas (pero igualmente expresivas) y después sintetizar el resultado a partir de estas. Los beneficios que aporta semejante descomposición lógica se resumen en dos puntos. En primer lugar, se simplifica la formulación y favorece la legibilidad. Y en segundo lugar, esas propiedades pueden reutilizarse modularmente para construir otras más complejas (bastan pequeños ajustes locales para variar la semántica de una fórmula, o incluso producir otras propiedades completamente distintas).

Existen dos clases generales de consultas: sobre propiedades *globales* o *asociadas al árbol*, en donde lo que se inspecciona es la propia estructura de la filogenia; o sobre propiedades *locales* o *asociadas a la secuencia*, donde la composición de características de las secuencias es el eje central (y que probablemente estarán complementadas por restricciones auxiliares). En resumen, en este capítulo nos dedicaremos a modelar formalmente ejemplos de ambos tipos de propiedades.

3.2. Propiedades del Árbol

Muchas de las propiedades típicas que se estudian sobre el árbol son de naturaleza cladística. Por ejemplo, una de las consultas más frecuentes pregunta si, dado un conjunto determinado de organismos S, constituyen un

grupo monofilético o *clado* bajo la filogenia particular de estudio. En otras palabras, ¿existe un subárbol que contiene exactamente esos organismos como sus hojas?. Formalmente, buscamos un nodo en alguna parte del árbol (**EF**) que funcione como raíz del grupo y, entonces: a) todo lo que debe estar dentro del subárbol, lo está; b) todo lo que debe estar fuera, también lo está fuera. Resumiendo, no hay ningún intruso o espurio que contamine el subárbol, y sólo los miembros de S están en él.

$$clado\,(S) \equiv \mathbf{EF}\,(in\,(S) \wedge out\,(S)) \tag{3.1}$$

La regla de inclusión ($in(S)$) afirma que para cada secuencia del conjunto S (aplicando un and lógico global $\bigwedge$) hay un camino que lo encuentra como hoja. Por su parte, la regla de exclusión ($out(S)$) demanda que todos los caminos finalicen en una hoja del mismo conjunto. En casos de computaciones infinitas como este, las hojas finales se localizan mediante los patrones $\mathbf{F} \circ \mathbf{AG}\,(s)$ que definen los autobucles de los estados terminales: por construcción es el único lugar a partir de donde la computación de un subárbol permanece uniforme (es siempre igual).

$$in\,(S) \equiv \bigwedge_{s \in S} \mathbf{EF} \circ \mathbf{AG}\,(s) \tag{3.2}$$

$$out\,(S) \equiv \mathbf{AF} \circ \mathbf{AG} \left(\bigvee_{s \in S} s \right) \tag{3.3}$$

Tal y como queda patente, las propiedades individuales por separado tienen la misma carga semántica. Aquí, $in\,(S)$ se satisface por todos los clados que contienen a s, y $out\,(S)$ por todos los subclados estrictos. Es más, si la propiedad cladística se extiende para englobar tanto a las secuencias ancestrales como a las hojas, la estructura de la fórmula principal permanece inalterada: sólo las reglas de inclusión y exclusión necesitarán un refinamiento extra, que comentamos a continuación, para que las secuencias objetivo puedan localizarse en cualquier lugar del subárbol y sin contaminación de intrusos externos.

$$in'\,(S) \equiv \bigwedge_{s \in S} \mathbf{EF}\,(s) \tag{3.4}$$

$$out'\,(S) \equiv \mathbf{AG} \left(\bigvee_{s \in S} s \right) \tag{3.5}$$

Una propiedad más excitante todavía es averiguar, dado un árbol filogenético y una partición de sus hojas (que nace de una partición según la

clasificación de las secuencias), si esa partición constituye una *clasificación en haplogrupos* del árbol. Los haplogrupos son agregaciones de haplotipos relacionados que se identifican con polimorfismos característicos comunes. Por tanto, definen poblaciones genéticas, que además pueden marcarse geográficamente también. Este estudio está enfocado a las regiones no recombinables del genoma, en especial del ADN mitocondrial (de donde surgió la notación cladística original para haplogrupos [29]); y la mayor parte del cromosoma Y (para el que se utiliza esta notación [32]).

Esencialmente, un haplogrupo junto con un conjunto de poblaciones (haplogrupos hijo) que han nacido de él con el paso del tiempo, deben formar un clado. En otras palabras, un haplogrupo es un *clado anidado*: todos sus miembros ocupan las hojas del árbol, excepto posiblemente un número de sub-subárboles que carecen completamente de miembros (y tienen a su vez una estructura anidada de clado). Una filogenia tiene una clasificación válida si cada parte tiene una estructura de haplogrupo.

$$clasificador\,(S_1, S_2, \ldots, S_h) \equiv \bigwedge_{i=1}^{h} haplogrupo\,(S_i) \tag{3.6}$$

La comprobación es trivial si se conocen las relaciones entre padres e hijos del haplogrupo, simbólicamente definidas por una función $hijos\,(S)$ (que no una fórmula).

$$haplogrupo'\,(S) \equiv clado\,(S \cup hijos\,(S)) \tag{3.7}$$

Sin embargo, normalmente nos interesa permitir flexibilidad en la ubicación de los haplogrupos de estudio, el refinamiento de haplogrupos de grano grueso y la exploración de hipótesis alternativas. La ecuación (3.7) puede extenderse para determinar si un conjunto de haplogrupos hijo existen, pero a efectos prácticos es suficiente con comprobar la calidad local de los haplogrupos para cada parte por separado, sin recurrir a ninguna información adicional (más allá de la composición de las subfórmulas).

Formalmente, el haplogrupo es una relajación de $clado\,(S)$ a lo largo de su estructura local. Mientras que la regla de inclusión se preserva, al igual que en la búsqueda de la raíz del haplogrupo, la regla de exclusión se reemplaza por una propiedad de clado anidada. Bajo esta situación, todos los caminos eventualmente alcanzan un punto (**AU**) donde, o bien encuentran un miembro del subclado, o bien la raíz de un clado distinto, pero siempre siguiendo una traza a lo largo de nodos pertenecientes al haplogrupo en cuestión (para ello, se debe proporcionar de antemano una función de pertenencia h_i para cada S_i).

$$haplogrupo\,(S, h) \equiv \mathbf{EF}\,(in\,(S) \wedge nested\,(S, h)) \tag{3.8}$$

$$nested\,(S,h) \equiv \{h\}\ \mathbf{AU}\ \{out\,(S) \vee nesting\,(S)\} \tag{3.9}$$

$$nesting\,(S) \equiv \mathbf{AF} \circ \mathbf{AG} \left(\neg \bigvee_{s \in S} s \right) \tag{3.10}$$

Fijémonos bien que $nesting\,(S)$ es lo opuesto de $out\,(S)$. Obviamente, los haplogrupos terminales (esto es, clados o monofilias) son válidos para esta fórmula. En términos de cladística, la estructura de haplogrupos locales se corresponde con el concepto de *grupo polifilético*. Es precisamente por esto último que los hace indistinguibles de los *grupos parafiléticos* (basados solamente en el contenido de las hojas) por lo que la información de miembros ancestrales se torna necesaria (mirar el capítulo 3.3 e imagen 3.1).

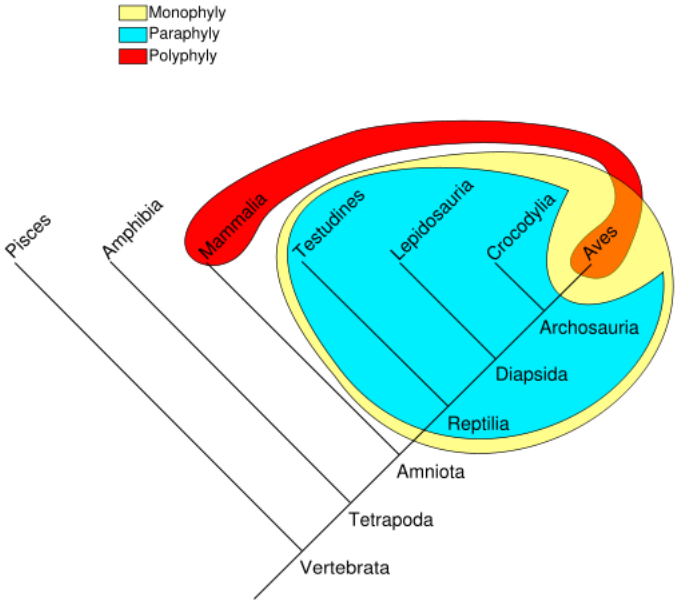

Figura 3.1: Distintos tipos de grupos filogenéticos

Por último, la asunción de que miembros ancestrales del haplogrupo satisfacen las propiedades definidas a priori es razonable (es decir, no aparecen casos especiales). Al igual que antes, la incorporación de datos ancestrales deriva en una familia relacionada de propiedades que permiten una evaluación más comprensiva del proceso de evolución.

3.3. Propiedades de Secuencia

En general, las propiedades asociadas a secuencias están basadas en fórmulas de estado (es decir, aquellas que se evalúan dentro de un nodo sin necesidad de recurrir a operadores temporales) compuestas junto con patrones temporales simples, principalmente para extender su estudio aplicándolo a la filogenia completa o para explorar sus alrededores.

A ese estilo de fórmulas de estado las llamaremos *patrones* (p). Ofrecen un potente formalismo de descripción para formular restricciones generales sin las limitaciones de una aproximación "ad-hoc". A menudo, esas propiedades se usan no necesariamente para prohibir patrones de mutación, sino como parámetros de calidad (consulta y alerta de señales inusuales posiblemente debidas a un comportamiento anómalo) y marcarlas para analizar en detalle posteriormente.

Un primer grupo de patrones lo constituyen las restricciones que estudian la corrección a nivel global, esto es, que supuestamente deben cumplirse a lo largo de toda la filogenia. Pueden clasificarse de la siguiente manera:

- La *conservación*, que se modela como una restricción en los símbolos que pueden aparecer en una posición dada de la secuencia. Comúnmente, el patrón se codifica como un vector unidimensional de booleanos que clasifica cada símbolo como permisible o no permisible. Sin embargo, entre otras extensiones, es posible definir familias de elementos compatibles (no necesariamente ligados a una determinada posición), y que puedan conmutarse entre sí para obtener secuencias "equivalentes".

- La *covariación*, que impone una relación de dependencia entre dos (o más) posiciones en una secuencia. De forma general, se representa como una matriz bidimensional de booleanos en la que, para cada símbolo de la primera columna, se marca el conjunto de símbolos admisibles en la segunda columna. Típicamente, para que la propiedad sea significativa, las asociaciones deben producirse entre símbolos dispersos.

- Una combinación de ambas.

Un patrón global así definido es fácilmente verificable con sólo extenderlo a todo el árbol de computación.

$$global\,(p) \equiv \mathbf{AG}\,(p) \tag{3.11}$$

Las excepciones que surjan a las propiedades mencionadas anteriormente pueden indicar que esas mutaciones son potencialmente peligrosas (o al menos

sospechosas), lo que es de gran interés en los estudios aplicados a filogenética [27]. Además, las (potenciales) mutaciones que se conocen a priori pueden modelarse explícitamente como patrones y evaluarse en las posiciones adecuadas de la filogenia. Especialmente si esas mutaciones afectan a funciones metabólicas importantes, es previsible que sean patógenas y limiten o impidan la reproducción del organismo, confinándolo a las hojas terminales (o cerca de ellas).

Aunque para algunas mutaciones está sistemáticamente prohibido que aparezcan como patrones globales, para otras mutaciones patógenas se les está permitido aparecer bajo a ciertas condiciones. Especialmente, es necesario que si aparece un patrón de mutación peligroso, éste no tenga sucesores (es decir, sea una hoja en la filogenia); o para proveerlo de cierta flexibilidad, desaparezca al cabo de un máximo de k pasos o sucesores ($\mathbf{AX}^k$).

$$terminal\,(p) \equiv \mathbf{AG}\,(p \rightarrow hoja) \tag{3.12}$$

$$terminal\,(p,k) \equiv \mathbf{AG}\left(p \rightarrow \mathbf{AX}^k\,(hoja)\right) \tag{3.13}$$

En este caso, las hojas (autobucles en la estructura de Kripke) deben detectarse sin ninguna referencia a una secuencia particular. Para conseguirlo fácilmente es suficiente con comparar los *vectores de estado* (los valores de AP) propios del estado objetivo con los de todos sus sucesores.

$$hoja \equiv \bigwedge_{p \in AP} p \leftrightarrow \mathbf{AX}\,(p) \tag{3.14}$$

Este último ejemplo representa las propiedades que desarrollan *exploraciones condicionales* de la filogenia. La verificación de una línea evolutiva específica sería el paso siguiente, donde los patrones servirían para definir conjuntos de estados relevantes para una fórmula, aunque su estudio está fuera del ámbito de alcance de este trabajo. Merece la pena reseñar que el chequeo de patrones en la clasificación de haplogrupos cae dentro de esta categoría.

Capítulo 4

Implementación sobre un paquete real

4.1. Traducción a SMV

En este capítulo apoyamos nuestra propuesta anterior con resultados experimentales sobre un model checker real. Para este propósito hemos elegido SMV [22], que es una herramienta popular de model checking con una larga trayectoria. La hemos escogido de entre todas las opciones existentes principalmente por su potencia. En especial, en su versión de Cadence SMV [4], posee una sintaxis más completa (incluye bucles, mejora la definición de variables temporales, etc) y permite un mayor número de operaciones sobre vectores de datos simbólicos, lo que facilita el manejo de cadenas de ADN o proteínas como vectores de caracteres. Recordamos que el ADN y las proteínas son datos importantes sobre los que normalmente se estudian propiedades del análisis filogenético. Para más detalles técnicos, los manuales y el software se encuentran disponibles en sus respectivas web [5, 4].

Antes de comenzar la implementación, fue necesario diseñar cómo se estructuraría el código del model checker. El esquema escogido en SMV para representar el árbol filogenético se observa en el siguiente ejemplo simplificado:

Algoritmo 4.1 Árbol evolutivo en notación SMV

```
MODULE main
VAR
  taxon: {taxon1,taxon2,taxon3,taxon4,taxon5};
  adn: process adn_taxon(taxon);
ASSIGN
  init(taxon) := taxon1;
  next(taxon) :=
      case
         taxon=taxon1: {taxon2, taxon3};
         taxon=taxon2: {taxon4, taxon5};
         1: taxon;
      esac;
  SPEC AG(adn.secuencia[1..3]=[G1,A1,T1])
MODULE adn_taxon(id){
  INPUT  id: {taxon1,taxon2,taxon3,taxon4,taxon5};
  VAR    secuencia: array 1..3 of {A1,C1,G1,T1};
  secuencia:= switch(id){
taxon1 | taxon2 | taxon3: [A1,G1,T1];
taxon4: [A1,A1,T1];
taxon5: [A1,G1,T1];
default: [A1,G1,T1];
};
}
```

El módulo principal que aparece en la primera parte corresponde a la definición del árbol filogenético en sí, donde la variable "taxon" indica el conjunto de valores que puede almacenar (en este caso, "taxon1", "taxon2", ..., representan las hojas y nodos intermedios). A continuación, la variable "adn" toma su valor como cadena de texto de manera asíncrona, mediante la tarea "adn_secuencia" que se dispara cuando una transición se ha ejecutado.

Después, mediante la sentencia "init" indicamos el conjunto de estados iniciales (por ahora, con los modelos más sencillos sólo contamos con una raíz en el árbol). Las transiciones se indican mediante la instrucción "next", que marca el conjunto de estados alcanzables según el estado actual. La última guarda, escrita como "1", es la selección por defecto y sirve para generar los autobucles infinitos.

Como peculiaridad indicar que, de acuerdo al ejemplo mostrado, la elección del siguiente estado se realiza de forma indeterminista en tiempo de ejecución. Es decir, para el caso de que estemos en "taxon1", la máquina de

estados disparará cualquiera de las dos opciones posibles: "taxon2" o "taxon3". Supone una característica interesante, ya que emula la deriva evolutiva que pudo sufrir una especie desde sus comienzos.

En segundo lugar, se colocan las especificaciones que queremos verificar en lógica temporal, introducidas por la palabra clave "SPEC". Los operadores lógicos y de camino en CTL se corresponden unívocamente con su palabra clave en SMV. Además, es posible macrear expresiones temporales mediante la cláusula "DEFINE" y almacenar resultados en variables booleanas. Para más información, se remite a los manuales del lenguaje.

Por último, y de manera desacoplada a la descripción de la estructura de Kripke, está la función encargada de decorar los nodos y etiquetar con propiedades ("adn_taxon(id)"). En este ejemplo sencillo, simplemente hemos etiquetado los nodos con una cadena de tres caracteres. Recalcar como inciso que el alfabeto de los vectores se define simbólicamente mediante enumeración (el alfabeto de los aminoácidos en las proteínas es distinto a los nucleótidos del ADN). Además, se les suele concatenar un carácter extra, por ejemplo el carácter "1", para evitar colisiones de nombre con los enteros en formato hexadecimal.

Para traducir a SMV desde un formato Newick, "estándar" para la notación de árboles filogenéticos, utilizamos un script perl junto con la librería Bioperl [1] (que facilita las conversiones entre ficheros) para realizarlo de forma automática con conjuntos grandes de pruebas. Como datos de entrada, aparte del árbol, opcionalmente requiere del alineamiento de secuencias y su alfabeto. Por ahora no se considera otro tipo de etiquetado de los nodos con información extra (por ejemplo, con la caracterización en haplogrupos o familias de especies), y de necesitarse, deberá confeccionarse a medida. Esto se debe a que, habitualmente, las propiedades sobre secuencias de ADN son las más costosas en tiempo de verificación y en representación de memoria, por lo que resultan interesantes para el estudio de viabilidad del apartado 4.2.

4.2. Rendimiento

Con el fin de demostrar la viabilidad y escalabilidad del sistema, se han hecho pruebas para medir su rendimiento. Especialmente, se ha analizado el coste de generar la estructura de transiciones asociada al árbol evolutivo; y el coste de almacenar las secuencias de ADN, pues las cadenas de texto son las propiedades más exigentes tanto para guardar como para verificar.

Los genes escogidos para los test de carga son: ND1, ND2, ND3, ND4L, ND4, ND5 y ND6. Se han elegido por su diversidad de tamaño, siendo ND5 uno de los genes mas grandes que pueden encontrarse en ADNmt, tal y como

se aprecia en la imagen 4.1. Además, pertenecen el complejo respiratorio 1 de la mitocondria, una de las partes que estudiamos conjuntamente con los miembros de la Facultad de Veterinaria y que podría beneficiarse de las técnicas de model checking.

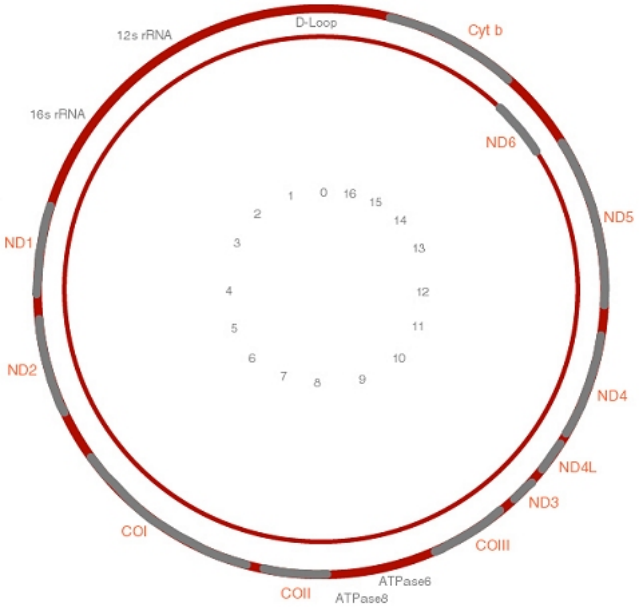

Figura 4.1: ADN mitocondrial

En particular, se han utilizado muestras de ADNmt humano extraídas de la base de datos GenBank [3]. Para estudiar la evolución, se han generado árboles que tienen entre 500 y 2000 secuencias en las hojas (a ellas hay que unirles otros tantos nodos internos, cuya cantidad depende de la estructura final de cada árbol). Además, como simplificación, se han utilizado las proteínas que codifican esos genes en vez de directamente su secuencia de nucleótidos del ADNmt: la secuencia de nucleótidos de un gen en el ADNmt es el triple de larga que la cadena de aminoácidos de su proteína asociada. Los árboles filogenéticos (según el método de neigbour joining) en formato Newick, junto con los alineamientos, han sido generados por la herramienta ClustalW2 [2].

El estudio del rendimiento está dividido en dos partes. En primer lugar, la tabla 4.1 calcula los segundos necesarios en generar una estructura de Kripke y almacenar la secuencia asociada.

	Nº Muestras			
Longitud Secuencias	500	1000	1500	2000
98 (ND4L)	6,59	13,64	20,07	29,23
115 (ND3)	9,53	18,72	27,20	39,53
174 (ND6)	20,24	41,02	59,61	86,04
318 (ND1)	69,99	129,29	191,82	261,54
347 (ND2)	76,88	152,9	227,13	326,11
459 (ND4)	133,32	271,05	393,49	543,87
603 (ND5)	224,58	450,07	673,50	970,39

Tabla 4.1: Coste temporal de creación de la estructura de Kripke y almacenamiento de las secuencias

Los tiempos obtenidos están en segundos y son razonablemente aceptables para el número de datos utilizados: apenas unos 15 minutos para el caso más extremo de nuestras pruebas. Sin embargo, escalan cuasi cuadráticamente conforme aumenta la longitud de las cadenas de texto: aumentar por seis el tamaño de las secuencias significa multiplicar el tiempo por casi treintaiseis.

Mientras tanto, si incrementamos el número de muestras por cuatro, el tiempo escala linealmente aproximadamente en el mismo orden de magnitud. Esto se debe potencialmente a que, puesto que nos movemos dentro de un mismo gen (o especie), las muestras no difieren demasiado entre sí (hay poca variabilidad) y es fácil fusionar partes comunes gracias a la estructura de datos que el SMV utiliza. Dicha estructura de datos se denomina *reduced ordered binary decision diagrams* [11], y le propondremos una extensión en el capítulo 5 para mejorar el manejo de cadenas de caracteres. De hecho, la tabla 4.2, que hace referencia a los megabytes necesarios para guardar las secuencias y la estructura de Kripke, motiva aún más su optimización (en gran parte de los casos, ocupa más de un giga). Y todo ello a pesar de que el coste en memoria es lineal para todos los casos, tanto en función del tamaño del conjunto como de la longitud de la cadena de ADN.

	Nº Muestras			
Longitud Secuencias	500	1000	1500	2000
98 (ND4L)	116	228	340	451
115 (ND3)	135	266	397	528
174 (ND6)	202	401	599	797
318 (ND1)	366	728	1089	1451
347 (ND2)	399	794	1188	1582
459 (ND4)	527	1048	1570	2091
603 (ND5)	691	1376	2061	2746

Tabla 4.2: Coste en memoria para la creación de la estructura de Kripke y almacenamiento de las secuencias

Recordamos que cada gen, aunque están extraídos del mismo ADNmt fuente, evolucionan "independientemente" y por eso puede haber ligeras oscilaciones sutiles en la pendiente de sus respectivamente rectas, pero todas siguen la misma tendencia.

Hemos escogido la primera columna (500 muestras) y la primera fila (gen ND4L) de la tabla 4.1 para demostrar gráficamente cómo crece el coste. El color negro de las imágenes representa la función que mejor aproxima a nuestros datos, que aparecen en rojo. Además, se incluye la ecuación de dicha función y su coeficiente de determinación R^2 (cuanto más cercano a uno, menor error entre nuestros datos y la función que los aproxima). La imagen 4.2 muestra el crecimiento lineal del coste temporal cuando aumentamos el número de muestras, mientras que en la imagen 4.3 se observa el crecimiento cuadrático con la longitud de la secuencia. Las gráficas que muestran el crecimiento lineal del consumo de memoria no se han incluido porque esta tendencia se puede apreciar fácilmente observando los valores de la tabla 4.2.

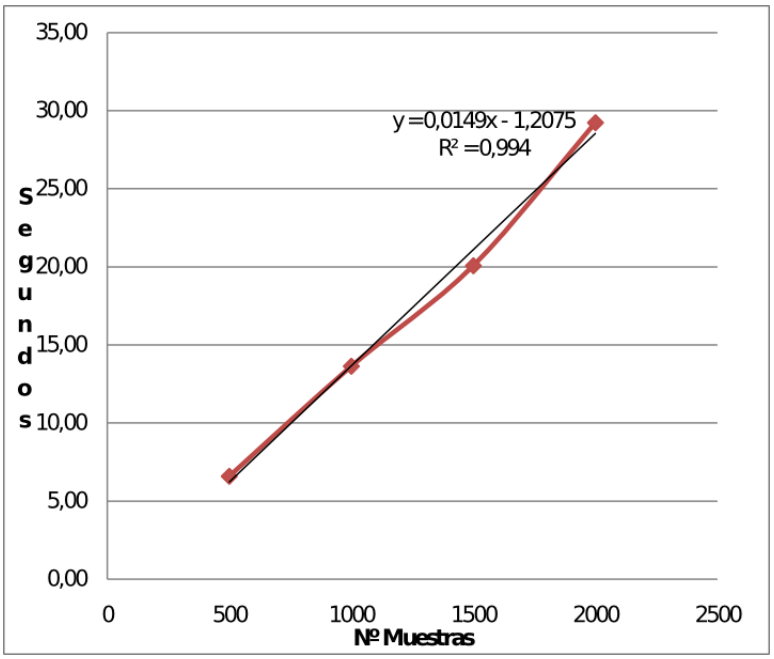

Figura 4.2: Gráfica de crecimiento temporal lineal respecto del número de muestras

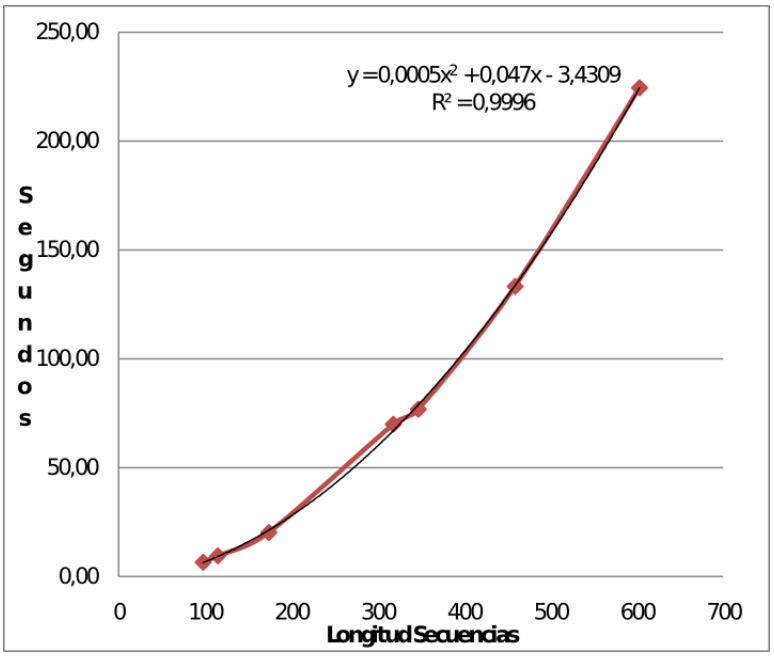

Figura 4.3: Gráfica de crecimiento temporal cuadrático respecto de la longitud de las secuencias

A continuación, la siguiente prueba (cuyos resultados se pueden ver en la tabla 4.3) difiere de la anterior en que aquí se considera el tiempo consumido para la verificación de propiedades. Los resultados representan los segundos extra que hay que añadir a la estructura inicial para validar 190 ecuaciones sencillas en lógica CTL.

La verificación de propiedades es realmente dependiente del método de exploración escogido por el model checker (profundidad o anchura) y del primer estado donde la fórmula CTL se torna falsa (esto es, rompiendo el proceso de verificación y saliendo con un contraejemplo bajo el brazo). Por tanto, las ecuaciones de lógica temporal han sido generadas automáticamente, con el fin de que el resultado de su evaluación sea lo más diverso posible. De esta forma, podremos considerar los resultados de la tabla 4.3 como una estimación del coste medio introducido por el proceso de verificación.

	Nº Muestras			
Longitud Secuencias	500	1000	1500	2000
98 (ND4L)	8,75	15,04	26,08	38,57
115 (ND3)	4,48	14,11	25,25	36,89
174 (ND6)	8,92	14,65	25,91	35,47
318 (ND1)	6,33	23,14	34,65	46,58
347 (ND2)	8,89	14,88	29,13	28,74
459 (ND4)	9,28	8,66	27,81	50,05
603 (ND5)	13,28	20,37	29,05	38,88

Tabla 4.3: Coste temporal de verificación de propiedades

De esas 190 ecuaciones creadas, la mitad corresponden a fórmulas del estilo:

```
SPEC EF AG ((adn.secuencia[i] = A1) || (adn.secuencia[i] =-1));
```

La fórmula es una variante de la definición de conservación: busca un subgrupo de secuencias para el cual la posición i está completamente conservada en todos ellos y se corresponde con un aminoácido "A". En particular, el valor "-1", que en nuestro caso es el valor comodín por defecto de las secuencias, hace de guarda para forzar una búsqueda más profunda y así conseguir una estimación más realista.

El resto de fórmulas sirven para ejemplificar la covariación y tienen como patrón:

```
SPEC AF ((adn.secuencia[i] = B1)-> (EX (adn.secuencia[i+1] = V1)));
```

La ecuación equivale a decir: para todo nodo (especie) que en el futuro tenga un aminoácido "B" en la posición i de la cadena, algún descendiente directo suyo tendrá un aminoácido en la posición $i + 1$ que será "V". Ambos tipos de ecuaciones se comprueban para todos los valores de secuencia desde la posición $i = 1$ hasta la $i = 95$.

Los benchmark han sido ejecutados sobre una máquina con procesador Intel Core 2 Duo E6750 a 2,66 GHz, 8 GB de RAM, Linux Debian con kernel 2.6.32 como sistema operativo, Cadence SMV versión 10.11.02p1 y el monitor de memoria memmon (del paquete UTILIB 4.1 [6]) para medir automáticamente el consumo de memoria máximo. La implementación del software SMV es secuencial, así que únicamente se ha utilizado uno de los dos cores del procesador.

Para concluir este punto, desearíamos resumir dos aspectos importantes. En primer lugar, el sistema propuesto funciona aceptablemente rápido para analizar genes aislados y con una carga notable de muestras, lo cual demuestra la viabilidad del entorno para estudios concretos.

Sin embargo, el entorno escala mal con la longitud de las cadenas. De hecho, se ha comprobado que el sistema es incapaz de manejar cadenas que se mueven entorno a las decenas de miles de caracteres y unos pocos cientos de muestras (u órdenes de magnitud similares). Eso significa que, en el caso de desear verificar propiedades complejas y que engloben a la totalidad del ADNmt, será necesario descomponerlas en varias proposiciones lógicas separadas más sencillas que afecten solamente a segmentos de tamaño más reducido. Posteriormente habrá que reconstruir la solución final a partir de resultados parciales.

Esta idea de descomposición automática de proposiciones lógicas es innovadora porque hasta ahora los artículos de investigación publicados se enfocaban en reducir el tamaño de la estructura de Kripke o la longitud de la computación: por ejemplo, en nuestro contexto, dividir el total de muestras en subconjuntos más pequeños y verificar las propiedades sobre cada subconjunto por separado [9]. El principal problema latente es que al verificar cada subpropiedad independientemente, en la reconstrucción de la solución final debemos comprobar que los caminos escogidos sean iguales.

Capítulo 5

Representación compacta de secuencias de ADN mediante diagramas de decisión

5.1. Contexto

La representación plana clásica de una secuencia de ADN, entendida como una sucesión lineal de caracteres, es inviable: incluso para cadenas de tamaño relativamente pequeño como el ADN mitocondrial (en comparación con el ADN nuclear), el tratamiento de sus $16K$ caracteres se complica a la hora de procesar y almacenar conjuntos elevados de muestras. Esto queda especialmente patente en el coste de almacenamiento en memoria y en el coste de ejecución de las herramientas de *model checking* (capítulo 4.2).

Como solución, proponemos el uso de *diagramas de decisión* para representar secuencias de texto. Los diagramas de decisión son una estructura de datos equivalente a los grafos acíclicos dirigidos. Normalmente, los model checkers trabajan con una versión binaria, los *diagramas de decisión binarios* (BDD), para manejar grandes conjuntos de datos. En particular, utilizan los *reduced ordered binary decision diagrams* (ROBDD) [11], que son un caso especial donde se impone una relación de orden entre las variables, para representar las estructuras de Kripke.

Como principal ventaja, los ROBDD destacan por obtener la representación canónica mínima para un conjunto de elementos de acuerdo con una relación de orden predefinida: los diagramas de decisión permiten solapar caminos comunes del grafo para consumir menos espacio, lo cual es bastante ventajoso para representar cadenas de texto en estudios poblacionales, donde más del 90 % del ADN de una especie está completamente conserva-

do en todos sus individuos. Además, facilita la comparación entre conjuntos aparentemente distintos gracias a la reducción a su forma canónica. Sin embargo, encontrar la función de ordenación óptima para los datos de entrada es un problema *NP-completo*.

Por último, los diagramas de decisión también colaboran implícitamente en el análisis de conservación y covariación de las secuencias. Por ejemplo, un análisis topológico de los caminos críticos del grafo nos proporciona los nodos (posiciones de la cadena) más conservados. A su vez, gracias a la reducción del espacio de estados, los algoritmos actuales de covariación se verán beneficiados: mejorarán el rendimiento y potencialmente reducirán el ruido de fondo que interfiere en los falsos positivos.

Salvo que se indique lo contrario, a lo largo del presente capítulo usaremos indistintamente "secuencia", "cadena (de texto)" o "palabra" como sinónimos. Aunque el estudio está enfocado hacia el ADN, también es extensible a otro tipo de cadenas, como por ejemplo proteínas, palabras de diccionario, etc.

5.2. Estado del arte

La aplicación de diagramas de decisión a conjuntos de secuencias todavía no ha sido ampliamente explorado, dado que existen otras estructuras de datos basadas en diccionario (como los *prefix tree* o *radix tree* [17]) que funcionan bastante bien y no necesitan buscar la función de ordenación óptima para los datos de entrada. De hecho, el uso de diagramas de decisión binarios para representar conjuntos de secuencias es relativamente moderno [20].

Inicialmente, la propuesta más sencilla es definir una variable booleana por posición de la secuencia y carácter del alfabeto, para indicar la presencia o ausencia de dicho carácter en esa posición determinada de la cadena. En particular, como sus variables son binarias, se necesita un máximo de $\lceil log_2 |\Sigma| \rceil$ "bits" por posición para alfabetos con una cardinalidad superior a dos elementos, siendo Σ nuestro alfabeto.

En [20] a esta aproximación la llaman *seqBDD*. Tenemos un ejemplo de ella en la imagen 5.1. En ese artículo, después de aplicar ciertas reducciones, demuestran que el tamaño de su diagrama de decisión es menor que el de un prefix tree equivalente, debido principalmente a la fusión de caminos comunes a lo largo del grafo y no únicamente de los prefijos compartidos.

Sequence	Itemset Encoded
$p_1 = aabac$	a_1, a_2, b_3, a_4, c_5
$p_2 = baba$	b_1, a_2, b_3, a_4
$p_3 = aaca$	a_1, a_2, c_3, a_4
$p_4 = bbac$	b_1, b_2, a_3, c_4

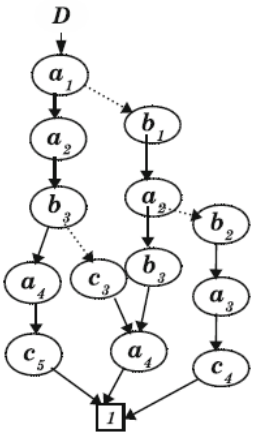

Figura 5.1: Representación en ROBDD de una secuencia (SeqBDD)

Sin embargo, su representación naif es cuestionable y aún podría mejorarse más porque no aprovecha toda la compactación disponible. Los nodos, al ser binarios, sobrecargan el sistema cuando representan el alfabeto del ADN: el alfabeto del ADN (y en general, cualquier otro) tiene más de dos caracteres. Aumentar el rango de valores que admiten las variables de los diagramas de decisión, para pasar de binarias a multivaluadas, sería un primer paso suficiente. Para ello, el punto 5.3 estudia las distintas opciones de diagramas de decisión diseñados para valores multivaluados, con el fin de analizar cómo de bien encajan en nuestro contexto. Junto con una serie de heurísticas expuestas en el punto 5.4, el apartado 5.5 demuestra que el coste de almacenamiento en memoria se ve reducido.

5.3. Herramientas y métodos

Los *data decision diagrams* [14] y los *multiple-valued decision diagrams* [23, 24] son una ampliación de los ROBDD para variables multivaluadas. Los *set decision diagrams* (SDD) [33, 18] son un caso especial de los anteriores, donde se admite etiquetar las aristas del diagrama con conjuntos de datos en vez de con una variable simple (la imagen 5.2 es un ejemplo de SDD). Esto sirve, sobretodo, para simplificar la notación en determinados casos cuando los arcos de salida de todas las variables asociadas a la posición i de la cadena van a parar al mismo nodo siguiente (imagen 5.4). Por tanto, los SDD suponen el caso más general y la mejor opción para conseguir adaptar la estructura de datos a las cadenas de texto que proponemos. Como breve inciso, en la notación a lo largo de este texto se han mantenido los arcos al

terminal cero (no pertenencia al conjunto) por tradición con la representación clásica de los diagramas de decisión, aunque es razonable omitirlos en futuras revisiones para aumentar la legibilidad, tal y como hacen los *zero binary decision diagrams* [25].

Las heurísticas aquí propuestas están reunidas en dos grandes grupos. Por un lado, está la parte considerada como "tratamiento previo" de las secuencias, que hace referencia a la fase de reordenación (punto 5.4.1) y cómo organizar las cadenas para sacar el máximo provecho a priori. Existen otras alternativas y consideraciones suplementarias que explicamos en el punto 5.4.3.

Por el otro, están las reglas de reducción (punto 5.4.2), que se aplican localmente una vez que hemos extraído toda la compactación inicial. Dado que aplicamos las reducciones sobre una estructura tipo seqBDD (que primeramente traducimos a un diagrama de decisión multivaluado SDD, inherentemente menor), con |seqBDD|≤|prefix tree| por [20], y como las simplificaciones son análogas a las que convierten un prefix tree en un radix tree (por ejemplo, colapsando los nodos que tienen un sólo hijo), todo esto implica que |SDD|≤|radix tree|. Es decir, conseguimos una cota superior más pequeña para los diagramas de decisión multivaluados, lo que supone una mejora en el ahorro de memoria respecto implementaciones anteriores.

5.4. Heurísticas

5.4.1. Reordenación inicial

Supongamos que tenemos un puñado de secuencias como las de la tabla 5.1. Idealmente, están alineadas correctamente de forma que las zonas comunes aparecen por bloques. En color rojo resaltan las columnas que poseen todos sus valores idénticos, y en verde aquellas con el elemento más frecuente. Si seleccionamos el mayor número posible de columnas que tengan todos sus componentes iguales, las colocamos al comienzo de la cadena y las fusionamos en un único nodo del grafo, reduciremos la dimensión total considerablemente.

	1	2	3	4	5
s_1	A	A	C	G	C
s_2	A	A	A	G	C
s_3	A	A	T	G	C
s_4	A	A	T	G	A
s_5	A	A	T	G	T

Tabla 5.1: Secuencias de ADN

Gracias a esta reordenación (figura 5.2), la implementación sobre el diagrama de decisión disminuirá potencialmente su número total de nodos respecto a un seqBDD (incluso menos que un prefix tree): las sentencias de ADN que comparten r columnas iguales almacenarán su cabecera en un mismo nodo, en vez de en r nodos independientes. Adicionalmente, retrasamos la bifurcación de las ramas, evitando que los caminos diverjan demasiado pronto. Esto nos permitirá aprovechar mejor la estructura de los SDD.

Esta filosofía de intercambio *a priori* de columnas, junto con un par de reducciones que expondremos en el capítulo 5.4.2, es un buen avance. La recuperación posterior del contenido desde la estructura de datos será sencilla porque sabemos la permutación escogida de los datos en todo momento. No obstante, encontrar la mejor reordenación global para un árbol de decisión es una tarea *NP-completa*.

La ordenación inicial se limita a resolver el problema del *conjunto de subsecuencias común más largo* [34], *LCS* por sus siglas en inglés. Existe un algoritmo de programación dinámica para dos cadenas de texto cuyo tiempo de ejecución es de orden polinómico con la longitud de los datos de entrada. La extensión del problema a un conjunto de m cadenas es *NP-completo*, pero al estar preprocesado el multialineamiento como en el ejemplo, se simplifica enormemente hasta hacerlo tratable también en tiempo polinomial: mientras recorremos las secuencias, leeremos los bloques que vayan apareciendo ($\mathcal{O}(m*k)$, siendo m el número de secuencias y k su longitud). El objetivo final es simplificar el grafo mediante reordenación, para ahorrar espacio en memoria y evitar registrar linealmente las m cadenas.

En todo momento es indispensable registrar la permutación elegida que transforma $s_i = x_{i_1}x_{i_2}\dots x_{i_k}$ en $\sigma(s_i) = s'_i = x'_{i_1}x'_{i_2}\dots x'_{i_k}$. Nos permitirá revertir posteriormente el efecto y definir la función característica de las secuencias. Es evidente que la sobrecarga introducida por guardar el patrón de permutación es tanto menor cuanto mayor número de secuencias haya.

$$\sigma = \begin{pmatrix} x_{i_1}x_{i_2}\dots x_{i_k} \\ x'_{i_1}x'_{i_2}\dots x'_{i_k} \end{pmatrix} \tag{5.1}$$

La permutación es única y se aplica a todo el conjunto total del alineamiento. Es decir, por sencillez, no admitimos de momento reordenaciones locales para subconjuntos de cadenas. El ejemplo anterior quedará intercambiado de la siguiente forma:

	1	2	4	5	3
s'_1	A	A	G	C	C
s'_2	A	A	G	C	A
s'_3	A	A	G	C	T
s'_4	A	A	G	A	T
s'_5	A	A	G	T	T

Tabla 5.2: Reordenación inicial

El conjunto de cadenas de texto $S = \{s_1, s_2, \ldots, s_m\}$ (definidas como "estados finales" o "alcanzables" de nuestro sistema) viene definido por la función característica del alineamiento. Esto es, a cada cadena le asignamos una función característica que la representa, y el conjunto S es la unión lógica de todas las funciones características de sus cadenas: $\chi_{RS} = \sum_{s_i \epsilon S} \chi(s_i)$. La función característica de una cadena $s_i = x_{i_1} x_{i_2} \ldots x_{i_k}$ es el valor resultante de la permutación $\sigma(\cdot)$:

$$\chi(s_i) = x'_{i_1} x'_{i_2} \ldots x'_{i_k} \tag{5.2}$$

Como botón de muestra, la ecuación 5.3 refleja el tratamiento de las funciones. El operador lógico *and* $(\cdot)$ simboliza la concatenación de subcadenas y el operador lógico *or* $(+)$ la elección.

$$\begin{aligned} \chi_{RS} &= \chi(s_1) + \chi(s_2) + \chi(s_3) + \chi(s_4) + \chi(s_5) \\ &= AAGCC + AAGCA + AAGCT + AAGAT + AAGTT \\ &= AAG \cdot [C \cdot (C + A + T) + (A + T)T] \end{aligned} \tag{5.3}$$

Finalmente, en el menor grafo SDD posible habrá tantas variables como subcadenas comunes. Gráficamente, el árbol resultado es el de la imagen 5.2. AAG está en la raíz. En el primer nivel de anidamiento, $C \cdot (C + A + T)$ y $(A+T) \cdot T$ son las dos subramas principales. $C \cdot (C+A+T)$ está repartido en dos nodos, siendo C el primero. Dada su condición de caracteres terminales, $(C + A + T)$ quedarán unidas en un mismo conjunto. $\{C, A, T\}$ es el arco del último nodo. Análogamente se construye $(A + T) \cdot T$.

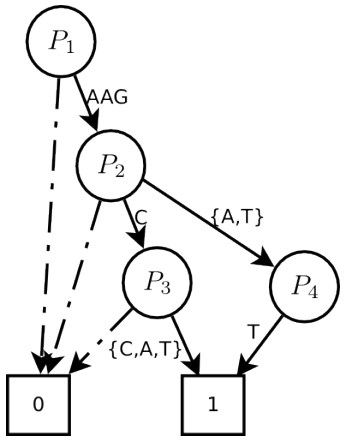

Figura 5.2: Set Decision Diagram

5.4.2. Reglas de reducción

En esta sección definimos algunas reglas de simplificación que permitirán transformar un seqBDD en una estructura más compacta. Si aplicamos dichas reglas sobre un seqBDD mínimo (previamente reducido mediante sus propias heurísticas [20]), demostraremos implícitamente que nuestra nueva estructura es más pequeña.

La representación de los seqBDD es binaria. Esto significa que cada carácter del alfabeto tiene dos posibles valores en la estructura: cero o uno (pertenencia o no pertenencia a la secuencia). Por consiguiente, habrá un máximo de n nodos por posición de la cadena, siendo $|\Sigma| = n$. El paso inicial consiste en ampliar el rango de valores de los diagramas de decisión utilizando los SDD expuestos anteriormente, ya conseguimos un gran avance. Mediante una función sobreyectiva $f_i : \mathbb{B}^n \to \mathbb{N}$ traducimos las n variables booleanas de un seqBDD en la posición i de la cadena de texto en una única variable numérica. Así, de esta manera dividimos como máximo por n la dimensión inicial.

La diferencia entre un *radix tree* y un *prefix tree* está en que, en el primero, los nodos con un solo hijo se fusionan con su descendiente para crear un nodo más grueso y así economizar memoria por la sobrecarga de la estructura (gestión de punteros, etc). Por tanto, podemos aplicar aquí mismo también la idea de fusión de nodos. Si las etiquetas a_i y a_{i+1} de la imagen 5.3 son un carácter (o una subsecuencia de caracteres), pueden colapsarse en un único nuevo arco conformado por la subcadena obtenida de su concatenación. La fusión de nodos es la reducción más prioritaria, dado que interviene en la compactación de caminos comunes del grafo.

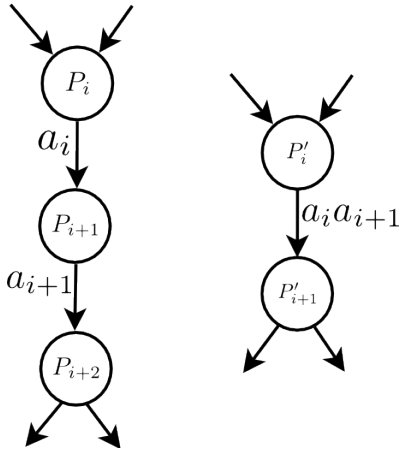

Figura 5.3: Fusión de nodos

El siguiente paso consiste en la unión de caminos. Si varios arcos apuntan a la entrada de un nodo, equivalen a un nuevo arco etiquetado como la unión de sus etiquetas (imagen 5.4).

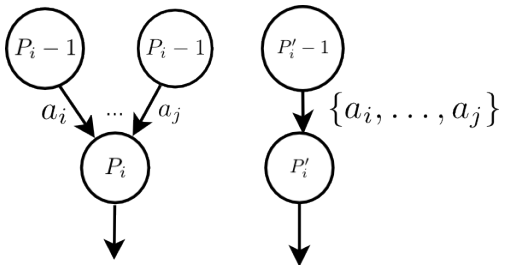

Figura 5.4: Unión de caminos

En los ROBDD existen métodos heurísticos basados en reordenación local que ofrecen buenos resultados en la práctica [31]. Aquí actuaremos de la misma manera, permutando únicamente cuando en un nodo aparece un a_m (carácter o subcadena), y en el otro un conjunto $\{a_i, \ldots, a_j\}$ (de otra forma, se fusionarían según la imagen 5.3):

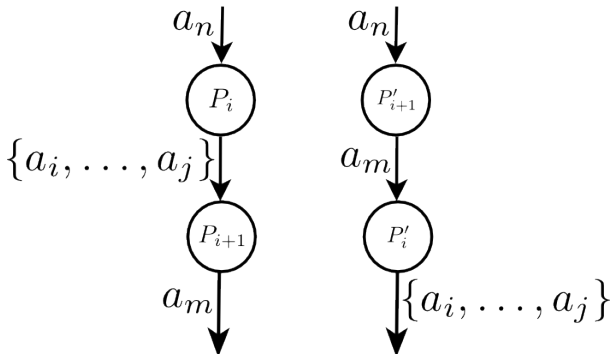

Figura 5.5: Permutación local

Las permutaciones locales de la imagen 5.5 tienen por labor trasladar las subcadenas hacia la raíz del árbol. Como si fueran burbujas, los nodos suben desde las ramas terminales arrastrando y engordando los segmentos comunes. Todavía falta por confirmar si en la práctica para este contexto la heurística se comporta igualmente bien, pero todo apunta a que la unificación de subcadenas al inicio, en la cabecera, es una buena opción.

5.4.3. Alternativas

En el capítulo 5.4.1 hemos visto que la reordenación inicial es un paso previo sencillo para alineamientos de secuencias homogéneas. Sin embargo, después del preprocesamiento inicial, ¿cómo recombinamos las columnas restantes para reducir el grafo?. En el fondo buscamos diseñar un proceso iterativo en el que, paso a paso, empuje hacia las primeras posiciones aquellas que sean "mejores".

La estructura idílica final es un alineamiento escalonado como el de la tabla 5.3, donde las casillas de color indican elementos comunes. Si el número de secuencias es mayor que el tamaño del alfabeto ($m > n$, $|\Sigma| = n$), todas las columnas alojarán más de un carácter repetido. Algunas de ellas, como las casillas azules, no podrán agruparse gráficamente dado que las cabeceras de color rojo las separan en conjuntos disjuntos.

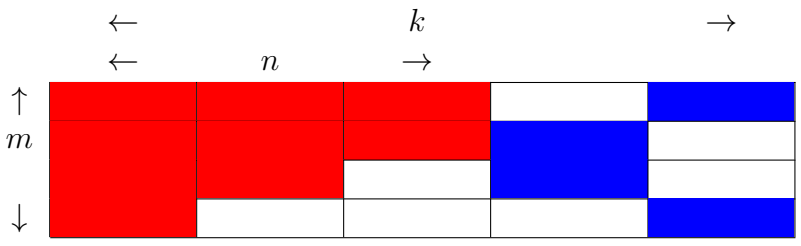

Tabla 5.3: Estructura ideal

Como primera elección, usamos la frecuencia del carácter más común como criterio de ordenación de las columnas. Las columnas donde su frecuencia sea mayor se colocarán al principio. De hecho, aquellas con todos los elementos exactos son un caso particular con frecuencia del 100 %. Por ejemplo, en la tercera componente de la tabla 5.1, la frecuencia de T y de C es del 75 %. En este ejemplo particular, optamos por cualquiera de las dos.

A pesar de ello, esta no siempre es la mejor opción. En la tabla 5.4 lo ideal es fusionar TGG y CA porque son las subcadenas más largas. Esto nos guía hacia la resolución del problema de las *m subcadenas comunes más largas* [28]. El problema de las m subcadenas comunes es una simplificación del problema de las subsecuencias LCS, donde sólo buscamos la subcadena más larga con soporte mayor o igual que m. Es también resoluble en tiempo lineal $\mathcal{O}(k)$ en función de la longitud k de las secuencias, pero deberíamos aplicarlo sucesivas veces hasta recubrir el tamaño completo del alineamiento.

	1	2	3	4	5		1	5	3	4	2
s_1	A	C	A	A	C	s'_1	A	C	T	G	G
s_2	A	C	A	C	A	s'_2	A	C	C	A	A
s_3	A	T	G	G	C	s'_3	A	C	A	T	T
s_4	A	T	G	G	A	s'_4	A	A	C	A	C
s_5	A	A	T	T	C	s'_5	A	A	T	G	G

Tabla 5.4: Reordenación alternativa

Complementariamente, una manera más sencilla de alcanzar nuestra meta es explotando nuevas reglas de reducción sobre un seqBDD. Las reglas descritas en el apartado 5.4.2 son una propuesta para construir nuestro nuevo diagrama a partir de un seqBDD previo.

Finalmente, la última propuesta es agrupar trozos de texto de tamaño homogéneo para conformar un nuevo "alfabeto". Las proteínas son un claro ejemplo de reescritura del texto original: están compuestas por aminoácidos, que son la síntesis bioquímica de tríos de nucleótidos. Esto significa que por cada tres nucleótidos del ADN, hay un aminoácido equivalente en su proteína asociada, lo que reduce la longitud final a la tercera parte. En términos matemáticos se asemeja a un "cambio de base", donde ganamos en abstracción y reducimos la longitud neta de la cadena, pero a cambio nos penaliza con un aumento del tamaño del alfabeto (que es perjudicial en cuanto a su gestión porque es un coste base que debemos guardar en memoria).

5.5. Complejidad mínima

El número total de nodos de un SDD está limitado por el tamaño del alfabeto y el número y longitud de las secuencias. En esta comparativa, analizamos el caso peor y mejor, y cómo de adecuada es la representación del ADN que hemos elegido. Partimos de m secuencias $S = \{s_1, s_2, \ldots, s_m\}$, $S \subseteq L_{DNA}(\Sigma)$ de longitud $\forall s \epsilon S, |s| = k$ con un alfabeto de tamaño $|\Sigma| = n$ y el lenguaje $L_{DNA}(\Sigma)$ formado por las posibles combinaciones de ADN. Entendemos como soporte $sop(a_i, j)$ el número de repeticiones del carácter a_i en la columna j del alineamiento.

Comenzamos el estudio de la complejidad con el mínimo grafo de decisión posible. El alineamiento ideal cumple una organización como la que aparece en la imagen 5.3. Por sencillez en la explicación, asumimos que las cadenas están reordenadas de mayor a menor similitud y con sus cabeceras comunes alineadas en un mismo bloque al principio. Las casillas coloreadas en la imagen simbolizan esos elementos iguales.

De inicio nos plantamos ante dos posibles contextos: $m > n$ y $m \leq n$. En el peor caso, el árbol tendrá un número de nodos igual a $n+1$ si $m > n$; o de m si $m \leq n$. Además, contará con dos arcos por nodo. Lo demostraremos por inducción. Partimos de la imagen 5.5. Imaginemos que y_1 es la subsecuencia común (de mayor soporte) más larga del conjunto inicial S y abarca a las m cadenas.

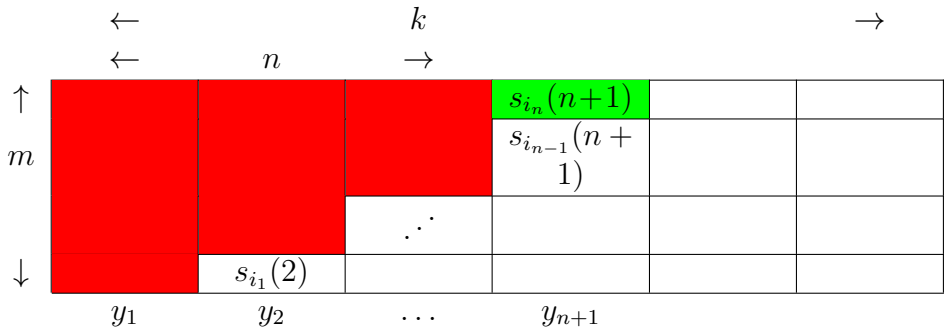

Tabla 5.5: Primer paso de la inducción

Lógicamente, $0 \leq |y_1| \leq k$. Los casos triviales son:

- $|y_1| = 0$, es decir, carecen de subcadenas comunes. Bastará una única variable para guardar S, en la cual se almacenarán directamente el conjunto de los m valores. Esto sólo puede darse cuando $m \leq n$. Es el peor caso en términos de memoria, ya que su coste es del orden $\mathcal{O}(m * k)$, suponiendo la representación de una cadena de longitud k

como constante. El tiempo de ejecución es simplemente $\mathcal{O}(m*k)$, puesto que consiste en leer secuencialmente los datos.

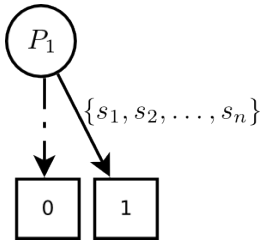

Figura 5.6: n cadenas completamente distintas ($|y_1| = 0$)

- $|y_1| = k$, donde todas las cadenas son idénticas y se da la mejor compresión. Al ser redundantes, será suficiente con un único nodo en el árbol y una transición a la hoja final. Su coste en memoria es $\mathcal{O}(k)$. Igualmente, el tiempo de ejecución requerido es $\mathcal{O}(m * k)$.

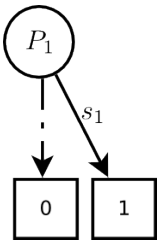

Figura 5.7: n cadenas completamente iguales ($|y_1| = k$)

Habitualmente, la subsecuencia estará limitada por $0 < |y_1| < k$. Por ejemplo, en el ADNmt entre humanos tendrá una cabecera común y_1 que posee una longitud $|y_1| = 0{,}9k$ (90 % de semejanza): en total, la estructura ocuparía $\mathcal{O}(\frac{(m+9)}{10}k)$.

Utilizamos el extremo $|y_1| = 1$ para la inducción, porque es el caso intermedio que tiene mayor recorrido. Debido a la ordenación, $y_1$está al comienzo de las secuencias. Puesto que $|y_1| = 1$, su rango está reducido a la primera columna. Eso significa que habrá al menos una secuencia $s_{i_1} \epsilon S$ cuyo carácter en la segunda columna sea diferente al resto ($s_{i_1}(2) \neq s_1(2) = \ldots = s_{i_1-1}(2) = s_{i_1+1}(2) = \ldots = s_m(2)$), o sea, con $sop(y_2, 2) = m - 1$. Creamos un nodo para guardar y_1, además de un arco a la siguiente subcadena común.

Avanzamos una posición para fijarnos sobre y_2. Nos hallamos ante la siguiente subsecuencia común más larga, esta vez sobre $S_1 = S \setminus s_{i_1}$. Ahora

necesitamos una variable que albergue dos valores: la secuencia $s_{i_2}[2..k]$, que es la diferente, y el carácter y_2. Siempre que leamos la secuencia $s_{i_2}[2..k]$ desde este punto, directamente sabremos que la cadena pertenece a nuestro conjunto de soluciones. Por tanto, un arco unirá directamente este nodo con una hoja terminal (marcada con *verdadero* para indicar su validez). En el caso de observar y_2 como parámetro de entrada, progresaremos en el tratamiento porque todavía no tenemos información suficiente del resto de posiciones.

Repetimos iterativamente este proceso. Según la inducción, en cada columna habrá al menos un carácter discordante pues nuestro alfabeto únicamente posee $|\Sigma| = n$. En total, entre columna y columna hay como límite $z = min(m, n+1)$ saltos de paso unidad, lo que corresponde a un máximo de z nodos. Salvo en el primer y último nodo, cada uno se bifurcará como máximo en dos conjuntos disjuntos (dos arcos de salida por paso), es decir, uno hacia la siguiente subcadena común y otra hacia la secuencia discordante.

La última cabecera tendrá una longitud $|y_z| = m - n$. Aquí se divide la casuística en dos partes. En la primera situación, $m \leq n$ no presenta demasiados problemas. Si $m \leq n$ (imagen 5.8), entonces $|y_m| = 0$: a lo sumo sólo quedan dos secuencia de S por procesar. Desde el nodo m partirá un arco hacia una hoja terminal, marcado con $s_{i_{m-1}}[m..k]$ y $s_{i_m}[m..k]$. El número total de nodos es m y el de arcos $2m - 1$.

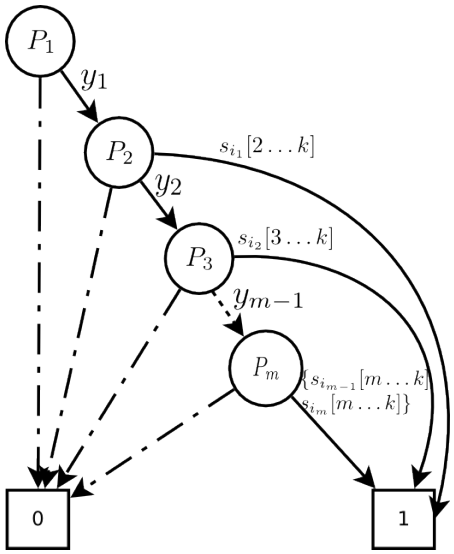

Figura 5.8: Esquema de la estructura final ($|y_m| = 0$)

La longitud media de la cabecera en escalón hasta $(m-1)$ es $\alpha = \frac{(m-1)(m+2)}{2m}$ (esquema 5.6). Después, a partir de la columna m todos los caracteres son distintos ($|y_m| = 0$). En total, pasamos de un coste en memoria

de $\mathcal{O}(m * k)$ a $\mathcal{O}(m * (k - \alpha + 1))$ porque sólo necesitamos almacenar una cabecera.

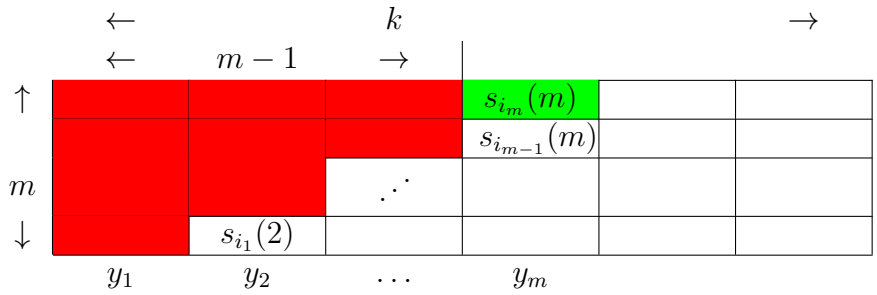

Tabla 5.6: Paso de inducción con $(m \leq n)$

Por contra, si $m > n$, entonces $|y_n| > 0$. En consecuencia, todas las columnas del alineamiento tendrán al menos un carácter $a_i \epsilon \Sigma$ tal que su soporte sea $\forall j \epsilon [1, k]$, $sop(a_i, j) > 1$. Esto significa que habrá elementos duplicados del alfabeto en las columnas: habrá bloques aislados del esquema 5.7 con soporte máximo $sop(a_i, j) = (m - n)$ para un carácter.

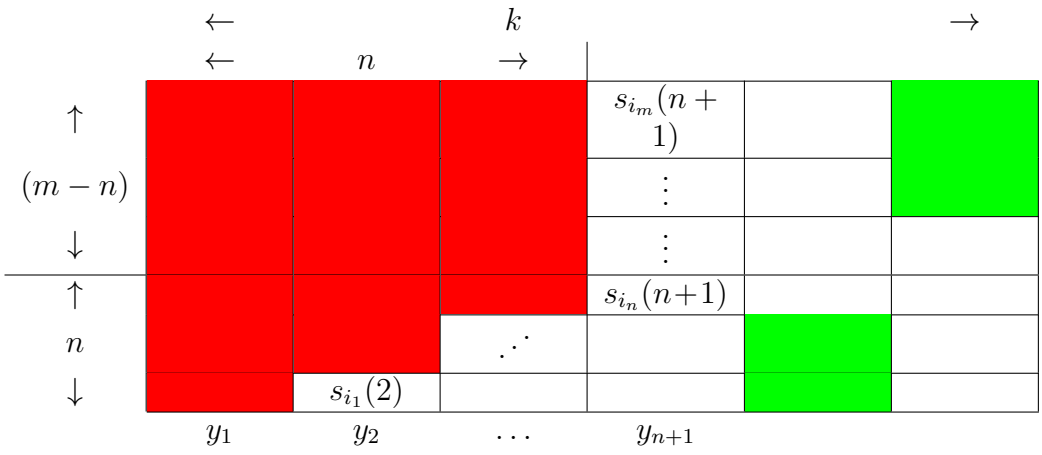

Tabla 5.7: Paso de inducción con $(m > n)$

Definimos una cota superior para averiguar el tamaño máximo de este caso, ya que encontrar una aproximación exacta es difícil por la distribución aleatoria de los subbloques. Imaginemos por un momento que dichos subbloques no existen y únicamente compartimos caracteres comunes hasta la cabecera y_n. Será necesario un último nodo adicional, tal y como se aprecia en la imagen 5.9. El arco final recogerá $S_n = \{s_{i_n}[n + 1..k], \ldots, s_{i_m}[n + 1..k]\}$. El número total de nodos es $n + 1$ y el de arcos $2n$.

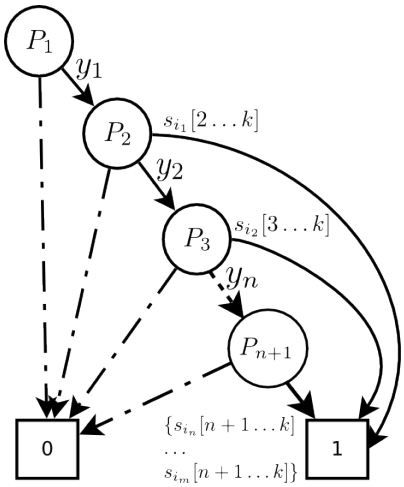

Figura 5.9: Esquema de la estructura final ($|y_n| > 0$)

La longitud media mínima de la cabecera en la escalera inicial es de $\frac{(n+1)}{2}$. En el tramo restante de las $(m-n)$ primeras secuencias, su longitud media es de n elementos. Simplificando, el coste es de $\alpha = \frac{n}{m}\left(\frac{(n+1)}{2} + (m-n)\right)$.

En total, la ocupación en memoria pasará de $\mathcal{O}(m * k)$ a $\mathcal{O}(m * (k - \alpha + 1))$ por el mismo motivo que en el caso anterior. Lógicamente, aumentar el número de muestras m no penaliza de igual manera que la longitud k de la secuencia, dado que α decrece proporcionalmente más rápido con el tamaño del conjunto de datos.

En resumen, se puede decir que este apartado ha demostrado que, incluso en el caso peor, el conjunto de muestras mantiene su tamaño tras un procesamiento en tiempo polinómico. Mientras tanto, en media, la reducción habitual es proporcional al porcentaje de similitud absoluto de las secuencias: cuanto mayor sea el número de muestras m y menor sea su longitud k, mejor se comportará la compactación.

Capítulo 6

Conclusiones

6.1. Conclusiones

El propósito de este trabajo ha sido enlazar conceptos entre dos mundos aparentemente alejados como es la filogenética y la verificación formal. Esquemáticamente, se ha hecho:

1. La interpretación de un árbol filogenético tradicional como un sistema de transiciones, cuyos estados corresponden a características compartidas por poblaciones de individuos definidas por un patrón genético común (herencia de ADN). Cuando miembros de esas poblaciones mutan y cambian alguna posición de su ADN ("evolucionan"), una transición se dispara entre estados del sistema. Entonces, se ha demostrado que existen transformaciones directas que asimilan un árbol filogenético a una estructura de Kripke equivalente sobre la que podrán usarse técnicas de model checking.

2. La definición de una lógica temporal adecuada para la definición de propiedades filogenéticas, semántica en la que se basan las estructuras de Kripke derivadas de los árboles filogenéticos. En el contexto de la filogenética, donde la evolución se representa como una sucesión de eventos causales a lo largo de una filogenia, los operadores temporales con semántica branching-time son indispensables. Las propiedades temporales preguntan entonces sobre relaciones de parentesco (eventos pasados o futuros) en el árbol.

3. El uso de técnicas estándar de model checking para la verificación automática de propiedades filogenéticas descritas en fórmulas de lógica temporal sobre árboles filogenéticos (descritos como estructuras de Kripke). Actualmente, el model checking es una técnica madura con

una buena base teórica y soportado por multitud de herramientas software.

A nuestro parecer, esta aportación es novedosa por la aplicación de un entorno de chequeo de modelos a la filogenética. De ahí se desprenden una serie de conclusiones interesantes que son, a saber:

1. El estudio de propiedades filogenéticas bajo diferentes modelos de evolución. Se ha tomado el árbol filogenético por defecto como modelo estándar a lo largo del trabajo; aunque fácilmente pueden considerarse otros entornos alternativos (redes filogenéticas, etc) que incluyan explícitamente eventos de recombinación y otras transformaciones complejas.

2. El entorno formal de razonamiento, derivado de la lógica temporal escogida, ayuda a comprender la estructura subyacente de una propiedad y su relación con el resto de propiedades. Entre otras cosas, esto permite descomponer la estructura subyacente de una propiedad en otras subproposiciones más sencillas, posiblemente verificadas y/o almacenadas anteriormente. De aquí se puede lograr una mejora importante en el rendimiento del proceso de model checking.

3. La utilidad de los resultados de verificación. Mientras que la verificación de una propiedad verdadera no reporta gran interés, el fallo en el cumplimiento de una fórmula involucra la generación de contraejemplos con los estados conflictivos (o cadena de eventos) que los causan. Los filogenetistas pueden utilizar esos resultados para refinar sus propiedades o descubrir otras nuevas.

4. Por último, pero no por ello menos importante, la importancia de software en el análisis filogenético. Gracias a que los algoritmos de verificación son de propósito general, no están sujetos a grandes cambios. Ahora los esfuerzos que antes se dedicaban a los detalles de implementación, ahora los investigadores podrán dedicarlos a la especificación de propiedades e interpretación de resultados.

6.2. Trabajo futuro

A pesar de las conclusiones positivas que hemos extraído de este trabajo, todavía quedan algunos aspectos que mejorar. Por un lado, está el tema de la escalabilidad. Aunque se ha demostrado que el rendimiento es aceptable para analizar genes individuales, está la asignatura pendiente de expandir

el sistema a propiedades complejas que involucren relaciones entre distintos trozos del ADNmt completo.

Además, puesto que el coste temporal se incrementa cuadráticamente con la longitud de la cadena, y el consumo de memoria se dispara para conjuntos de datos de tamaño medio, el siguiente paso natural es paralelizar el entorno en segmentos de ADN más pequeños que puedan analizarse en un tiempo razonable. El rebanado de propiedades en subproposiciones más sencillas es el que más nos interesa (frente a la descomposición tradicional en subestructuras de Kripke o computaciones más sencillas y cortas). Nuestro objetivo más inmediato consiste en estudiar cómo se componen los resultados parciales para recuperar la solución global de una fórmula CTL inicial, ya que además de verificar cada subpropiedad independientemente hemos de comprobar que existe al menos un camino común a todas ellas.

Por otro lado, queda la implementación de la estructura de datos propuesta para mejorar la eficiencia en el almacenamiento de conjuntos de secuencias de ADN. Aunque se ha estudiado su viabilidad teóricamente, y se han localizado las herramientas software con las que construir el sistema final, todavía queda por delante la fase de implementación, integración y experimentación dentro un model checker con pruebas de datos reales.

Finalmente, como último aspecto, se deja la puerta abierta para la definición y uso de lógicas más expresivas (por ejemplo, con la inclusión del "pasado"); o la extensión del conjunto de propiedades atómicas para enunciar propiedades cuantitativas en filogenética.

Nomenclatura

ADNmt ADN Mitocondrial, página 7

AP Atomic Propositions, página 13

BDD Binary Decision Diagram, página 34

CTL Computational Tree Logic, página 16

LCS Longest Common Subsequence, página 38

ROBDD Reduced Ordered Binary Decision Diagram, página 34

SDD Set Decision Diagram, página 36

seqBDD Sequence Binary Decision Diagram, página 35

Bibliografía

[1] Bioperl. http://www.bioperl.org.

[2] Clustal. http://www.clustal.org/.

[3] GenBank. www.ncbi.nlm.nih.gov/genbank/.

[4] Cadence SMV Model Checker. http://www.kenmcmil.com/, 2001.

[5] SMV Model Checker. http://www.cs.cmu.edu/~modelcheck/smv.html, 2001.

[6] UTILIB. https://software.sandia.gov/trac/utilib, 2010.

[7] André Arnold. *Systèmes de transitions finis et sémantique des processus communicants*. Masson, Paris, 1992.

[8] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. The MIT Press, Cambridge, MA, 2008.

[9] Sergey Berezin, Sérgio Campos, and Edmund Clarke. Compositional reasoning in model checking. In Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors, *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*, pages 81–102. Springer Berlin / Heidelberg, 1998.

[10] Roberto Blanco, Gregorio de Miguel Casado, José Ignacio Requeno, and José Manuel Colom. Temporal logics for phylogenetic analysis via model checking. In *IEEE International Conference on Bioinformatics & Biomedicine 2010*, December 2010.

[11] Randal E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *IEEE 22nd Design Automation Conference*, 1985.

[12] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE T. Comput.*, C-35:677–691, Aug. 1986.

[13] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logics of Programs*, pages 52–71, Yorktown Heights, NY, May 1981.

[14] Jean-Michel Couvreur, Emmanuelle Encrenaz, Emmanuel Paviot-Adet, Denis Poitrenaud, and Pierre-André Wacrenier. Data decision diagrams for petri nets analysis. Technical report, LIP6, 2002.

[15] Joseph Felsenstein. *Inferring phylogenies*. Sinauer, Sunderland, MA, 2003.

[16] Orna Grumberg and Helmut Veith, editors. *25 years of model checking: history, achievements, perspectives*. Springer, Berlin, 2008.

[17] Donald E. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching*, chapter 6.3. Addison-Wesley, 1998.

[18] Alexandre Hamez; Yann Thierry-Mieg; Fabrice Kordon. Hierarchical set decision diagrams and automatic saturation. Technical report, LIP6, 2008.

[19] Christopher James Langmead and Sumit Kumar Jha. Predicting protein folding kinetics via temporal logic model checking. In *Proc. WABI 2007*, pages 252–264, Philadelphia, PA, Sep. 2007.

[20] Elsa Loekito, James Bailey, and Jian Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowledge and Information Systems*, pages 1–34, September 2009.

[21] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems: specification*. Springer, Berlin, 1991.

[22] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[23] D. Michael Miller and Rolf Drechsler. Implementing a multiple-valued decision diagram package. Technical report, Department of Computer Science, University of Victoria, Canada, 1998.

[24] D. Michael Miller and Rolf Drechsler. On the construction of multiple-valued decision diagrams. Technical report, Department of Computer Science, University of Victoria, Canada, 2002.

[25] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Design Automation, 1993. 30th Conference on*, pages 272 – 277, June 1993.

[26] Pedro T. Monteiro, Delphine Ropers, Radu Mateescu, Ana T. Freitas, and Hidde de Jong. Temporal logic patterns for querying dynamic models of cellular interaction networks. *Bioinformatics*, 24:i227–i233, 15 Aug. 2008.

[27] Julio Montoya, Ester López-Gallardo, Carmen Díez-Sánchez, Manuel J. López-Pérez, and Eduardo Ruiz-Pesini. 20 years of human mtDNA pathologic point mutations: carefully reading the pathogenicity criteria. *Biochim. Biophys. Acta*, 1787:476–483, May 2009.

[28] Michael Arnold; Enno Ohlebusch. *Algorithmica*, chapter Linear Time Algorithms for Generalizations of the Longest Common Substring Problem, pages 1–13. Springer New York, 2009.

[29] Martin B. Richards, Vincent A. Macaulay, Hans-Jürgen Bandelt, and Bryan C. Sykes. Phylogeography of mitochondrial DNA in western Europe. *Ann. Hum. Genet.*, 62:241–260, May 1998.

[30] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *Proc. CMSB 2008*, pages 251–268, Rostock, Oct. 2008.

[31] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 42–47, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[32] The Y Chromosome Consortium. A nomenclature system for the tree of human Y-chromosomal binary haplogroups. *Genome Res.*, 12:339–348, Feb. 2002.

[33] Jean-Michel Couvreur; Yann Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. Technical report, LIP6, 2005.

[34] V. G. Timkovskii. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics and Systems Analysis*, 25(5):565–580, 1989.