Formalización computacional del experimento de Lipton sobre el problema SAT

Mario J. Pérez-Jiménez; Fernando Sancho-Caparrini
Carmen Graciani-Díaz; Alvaro Romero-Jiménez

Resumen— *In this paper a computational formalization of Lipton's experiment about satisfiability problem of Propositional Logic is presented. We design a program that implements this experiment within a DNA based molecular model without memory: the unrestricted model of Adleman. We propose a formal description of the designed molecular program as a formal system. The formal verification of the program is established proving soundness and completeness of formal systems associated with it, using invariant method and inductive techniques. Finally, an extension of Lipton's experiment for arbitrarily expressed formulas is given.*

Palabras clave— **Computación molecular, Computación ADN, Experimento de Lipton, Problema SAT, Verificación de Programas.**

I. Introducción

La *Computación Natural* es una disciplina inspirada en el funcionamiento de los organismos vivos. Tiene como objetivo fundamental la simulación e implementación de los procesos dinámicos que se dan en la Naturaleza y que son susceptibles de ser interpretados como procedimientos de cálculo.

En la actualidad, la Computación Natural consta de tres ramas. La primera está basada en el funcionamiento del *cerebro* y tiene como modelo informático las *redes neuronales artificiales*, creadas por McCulloch y Pitts en 1943. La segunda rama está basada en las propiedades del **ADN**. Tiene como precedentes a los *algoritmos genéticos*, creados por Holland en 1975, y al *modelo Splicing* introducido por T. Head en 1987; y como modelo propio la *Computación Molecular basada en ADN*, creada por L. Adleman en noviembre de 1994. La tercera rama de la Computación Natural está basada en el funcionamiento de las células y tiene como modelo la *Computación con membranas* o *P–sistemas*, creada por Gheorghe Păun en octubre de 1998. Como es bien sabido, tanto las redes neuronales como los algoritmos genéticos han sido implementados en ordenadores electrónicos convencionales. La computación molecular basada en **ADN** ha sido implementada en laboratorios de biología molecular. En cambio los P–sistemas aún no han sido implementados en medios electrónicos ni biológicos.

Hacia finales de la década de los cincuenta, R.P. Feynman describe los ordenadores *sub–microscópicos* e introduce el concepto teórico de *computación a nivel molecular*, postulándolo como una innovación revolucionaria en la carrera por la miniaturización. Los organismos vivos, en general, y algunas moléculas en particular, son consideradas potencialmente como máquinas capaces de desarrollar procesos interpretables como operaciones de cálculo.

A principio de la década de los cincuenta comienza a ponerse de manifiesto la analogía existente entre algunos procedimientos matemáticos y ciertos procesos biológicos. En noviembre de 1994, L. Adleman [1] materializó esta similitud justificando que es posible usar procesos biológicos para atacar la resolubilidad de ciertos problemas matemáticos especialmente *difíciles*: mediante un experimento realizado en el laboratorio resolvió una instancia concreta de un problema computacionalmente intratable (el *problema del camino hamiltoniano*, en su versión dirigida y con dos nodos distinguidos), usando técnicas de biología molecular para la manipulación del **ADN**.

Con el experimento de Adleman nace propiamente la computación **ADN** y se pone de manifiesto las presumibles ventajas de esta computación respecto de los ordenadores electrónicos convencionales, en lo que respecta a velocidad de cálculo, consumo de energía y densidad de información.

De la misma manera que el *transistor* permitió por primera vez la manipulación electrónica del *silicio*, el experimento de Adleman puede ser considerado como el primer paso hacia la construcción de un prototipo de ordenador molecular basado en la manipulación bioquímica del *carbono*.

En abril de 1995, R.J. Lipton ([5]) resolvió una instancia concreta del *problema* **SAT** *de la satisfactibilidad de la Lógica Proposicional* siguiendo las ideas de Adleman. A lo largo de ese año, aparecen los primeros modelos de computación molecular basados en **ADN**, a través de la instanciación de una serie de operaciones bioquímicas consideradas como básicas o primitivas. En 1996, D. Beaver [3] demuestra que dichos modelos de computación son universales, en el sentido de que en ellos se puede simular cualquier máquina de Turing determinista.

En este trabajo se presenta una formalización computacional del experimento de Lipton que resuelve el *problema de la satisfactibilidad de la Lógica Proposicional*. Para ello, se diseña un programa en un modelo molecular sin memoria basado en **ADN** (el *modelo no restringido* de Adleman [2]) que implementa el experimento citado en dicho modelo. Se propone una descripción del programa molecular diseñado como un sistema formal de tal manera que la verificación del programa se establece a través de la adecuación

Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla, España
Mario.Perez@cs.us.es

y completitud del sistema formal asociado. Finalmente se presenta una extensión del experimento de Lipton a fórmulas proposicionales arbitrarias.

II. El experimento de Lipton

El experimento de Lipton propone la resolución de una instancia concreta del *problema de la satisfactibilidad de la Lógica Proposicional* siguiendo las ideas de Adleman, con la peculiaridad de que el tubo de ensayo inicial no depende del dato de entrada concreto sino únicamente de su "tamaño" (número de variables de la fórmula). De esta manera se tiene una solución molecular de cualquier instancia del problema **SAT** de tamaño prefijado y, en consecuencia, a diferencia del experimento de Adleman, proporciona realmente el primer esquema algorítmico molecular.

A. El problema de la satisfactibilidad de la Lógica Proposicional

El lenguaje de la Lógica Proposicional consta de:

(a) Un conjunto infinito numerable, **VP**, de variables (que denominaremos proposicionales).
(b) Dos conectivas lógicas: $\neg$ (negación) y $\vee$ (disyunción).
(c) Dos símbolos auxiliares: "(" y ")".

A partir de la negación y disyunción, se definen de manera natural las conectivas lógicas $\wedge$ (conjunción), $\longrightarrow$ (implicación) y $\longleftrightarrow$ (doble implicación).

El conjunto **PForm** de las fórmulas proposicionales se define recursivamente (mediante un procedimiento generativo) como sigue: es el menor conjunto Γ que verifica, simultáneamente, las condiciones siguientes:

(a) $\mathbf{VP} \subseteq \Gamma$.
(b) Es cerrado bajo negación (es decir, si $\varphi \in \Gamma$ entonces $\neg\varphi \in \Gamma$).
(c) Es cerrado bajo disyunción (es decir, si $\varphi, \psi \in \Gamma$ entonces $\varphi \vee \psi \in \Gamma$).

Un *literal* es una variable proposicional o la negación de una variable proposicional. Una *cláusula* es una disyunción de literales. Se dice que una fórmula proposicional está en *forma normal conjuntiva* si es una conjunción de cláusulas; es decir, si es una conjunción de disyunción de literales.

Una *valoración* o *asignación de verdad* es una aplicación de **VP** en $\{0,1\}$. Toda valoración se extiende unívocamente de **VP** al conjunto **PForm** de manera natural (de acuerdo con las clásicas *tablas de verdad* de las conectivas lógicas).

Diremos que una valoración, σ, es *relevante* para una fórmula proposicional, φ, si y sólo si σ es una función booleana cuyo dominio contiene a $\mathrm{Var}(\varphi)$, siendo $\mathrm{Var}(\varphi)$ el conjunto de las variables proposicionales de φ. Obsérvese que el número de valoraciones relevantes para una fórmula proposicional φ tal que $\sigma(x) = 0$ para cada $x \in \mathbf{VP} - \mathrm{Var}(\varphi)$, es $2^{|\mathrm{Var}(\varphi)|}$.

Una fórmula proposicional se dice que es *satisfactible* si existe, al menos, una valoración que asigna 1 a dicha fórmula.

Problema de la satisfactibilidad de la Lógica Proposicional: *Dada una fórmula proposicional en forma normal conjuntiva, determinar si es satisfactible.*

Como es bien sabido, el problema de la satisfactibilidad de la Lógica Proposicional es **NP**–completo [4].

B. Preparación del experimento

A continuación describimos brevemente el experimento que permite resolver cualquier instancia del problema antes citado, a través de la manipulación de moléculas de **ADN**.

Sea $\varphi \equiv c_1 \wedge ... \wedge c_p$ con $c_i = l_{i,1} \vee ... \vee l_{i,r_i}$, una fórmula proposicional en forma normal conjuntiva, cuyo conjunto de variables es $\mathrm{Var}(\varphi) = \{x_1, ..., x_n\}$. Asociamos a la fórmula φ un grafo dirigido, $G_n = (V_n, E_n)$, definido como sigue:

$$\begin{cases} V_n = \{a_i, x_i^j, a_{n+1} : \ 1 \leq i \leq n, \ 0 \leq j \leq 1\} \\ \\ E_n = \{(a_i, x_i^j), (x_i^j, a_{i+1}) : \ 1 \leq i \leq n, \ 0 \leq j \leq 1\} \end{cases}$$

El grafo G_n, descrito en la figura 1, verifica las siguientes propiedades:

- Existen 2^n caminos simples desde a_1 hasta a_{n+1} en G_n.
- Existe una biyección natural entre el conjunto de los caminos anteriormente citados y las valoraciones relevantes para φ, de acuerdo con el siguiente criterio: si $\gamma = a_1 x_1^{j_1} a_2 x_2^{j_2} ... x_n^{j_n} a_{n+1}$ es un camino desde a_1 hasta a_{n+1}, entonces se le asocia la valoración, σ, relevante para φ, caracterizada por las relaciones $\sigma(x_i) = j_i$ $(1 \leq i \leq n)$.

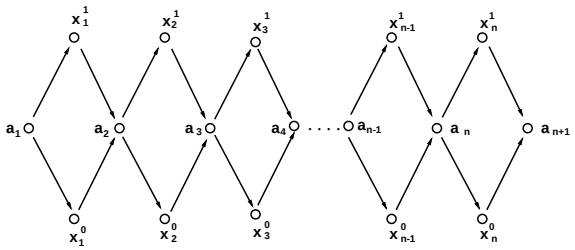

Figura 1. Grafo dirigido asociado a una fórmula proposicional con n variables

Para elaborar el tubo de ensayo inicial, T_0, se procede de manera similar a la seguida en el experimento de Adleman. A cada nodo, $i \in V_n$, del grafo se le asocia un oligo (cadena corta de **ADN**), s_i, de longitud 20 mer (en realidad este valor dependerá de n). Para

cada i se designa por s'_i (respectivamente, s''_i) los diez primeros (respectivamente, últimos) caracteres de la cadena s_i. A cada arco, $(i,j) \in E_n$, del grafo se le asocia el siguiente oligo, $e_{ij} = 3' - \overline{s}''_i \ \overline{s}'_j$.

Se parte de un tubo de ensayo que contiene una cierta cantidad de disolvente. Se le añade a la solución una cierta cantidad de oligos s_i, para cada nodo $i \in V_n$, la misma de oligos e_{ij}, para cada arco $(i,j) \in E_n$, y la misma de oligos $3' - \overline{s}'_{a_1}$ y $3' - \overline{s}''_{a_{n+1}}$. La solución se somete a un proceso de renaturalización, usando la enzima ligasa para empastar los huecos que pudieran aparecer entre nucleótidos. Finalmente se seleccionan las cadenas simples con bordes $5' - s_{a_1}$, para lo cual se somete la solución a un proceso de desnaturalización similar al seguido en el experimento de Adleman.

De esta manera, se obtiene una serie de doblehebras que codifican caminos del grafo desde a_1 hasta a_{n+1}. Las cantidades iniciales de oligos que se consideren deben garantizar que cada camino desde a_1 hasta a_{n+1} (y sólo esos) esté codificado en el tubo inicial. Además, debido a la simetría del grafo auxiliar considerado todos los caminos aparecen repetidos el mismo número de veces, aproximadamente. Por tanto, en el tubo de ensayo inicial se obtiene un multiconjunto de moléculas que codifican todas las valoraciones relevantes para la fórmula φ.

Para cada literal, $l_{i,j}$, que aparece en φ, notaremos

$$l^1_{ij} = \begin{cases} x^1_m & \text{, si } \ l_{i,j} = x_m \\ x^0_m & \text{, si } \ l_{i,j} = \neg x_m \end{cases}$$

Es decir, si una molécula contiene $l^1_{i,j}$ como subcadena, entonces el literal $l_{i,j}$ tiene asignado el valor 1 por la valoración que codifica dicha molécula.

La idea del experimento de Lipton es la siguiente: a partir del tubo inicial, T_0, que codifica todas las valoraciones relevantes para la fórmula de entrada, se procede como sigue:

- Se elabora un nuevo tubo, T_1, seleccionando de T_0 todas las valoraciones que hacen verdadera la cláusula c_1. Para ello:
  - ⋆ Se eligen las que hacen verdadero el literal $l_{1,1}$.
  - ⋆ De las que hacen falso $l_{1,1}$, se eligen las que hacen verdadero el literal $l_{1,2}$.
  - ⋆ De las que hacen falso $l_{1,1} \vee l_{1,2}$, se eligen las que hacen verdadero el literal $l_{1,3}$.
  - ⋆ Y así sucesivamente con todos los literales de c_1.
- Se elabora un nuevo tubo, T_2, seleccionando de T_1 todas las valoraciones que hacen verdadera la cláusula c_2 (así dichas valoraciones hacen verdadera la fórmula $c_1 \wedge c_2$). Para ello:
  - ⋆ Se eligen las que hacen verdadero el literal $l_{2,1}$.
  - ⋆ De las que hacen falso $l_{2,1}$, se eligen las que hacen verdadero el literal $l_{2,2}$.
  - ⋆ De las que hacen falso $l_{2,1} \vee l_{2,2}$, se eligen las que hacen verdadero el literal $l_{2,3}$.
  - ⋆ Y así sucesivamente con todos los literales de c_2.
- El proceso se reitera p veces hasta obtener el tubo de salida T_p, a partir de T_{p-1}, que contiene todas las valoraciones que hacen verdadera la fórmula $c_1 \wedge \ldots \wedge c_p$.

III. Formalización computacional del experimento de Lipton

En esta sección se presenta un marco formal (un modelo de computación molecular) en el que el experimento citado se puede considerar como la ejecución de un procedimiento *mecánico* (en dicho modelo).

A. Modelo de computación molecular

Recordemos que un modelo de computación, M, está caracterizado básicamente por: (a) una estructura de datos, $\mathcal{D}_M$; (b) un conjunto de programas, $\mathcal{P}_M$; y (c) una función semántica, $\mathcal{S}_M$, que asigna al par ordenado (P,n), siendo P un programa y n un número natural mayor o igual que 1, la función de aridad $n \geq 1$ que *calcula* el programa P.

Los datos de un modelo de computación molecular van a ser unos objetos, denominados *tubos*, asociados a un alfabeto arbitrario prefijado, Σ. Los elementos de cada tubo codificarán moléculas de ADN, con tal de asociar a cada símbolo del alfabeto un oligo verificando ciertas condiciones.

Las instrucciones básicas de un modelo de computación molecular pueden ser de dos tipos:

- *Moleculares*: operaciones basadas en propiedades de las moléculas de ADN.
- *Robóticas*: operaciones secuenciales estándar (bucles, condicionales, asignaciones, etc.) que, básicamente, proporcionan la secuenciación de las operaciones moleculares. Estas operaciones no actúan directamente sobre la estructura de las moléculas de ADN.

La función semántica del modelo se define de manera natural: (a) para las instrucciones moleculares, el resultado correspondiente de la operación molecular que representa; (b) para las instrucciones robóticas, la semántica secuencial convencional.

La elección de las operaciones moleculares primitivas de los modelos que se estudian se debe, entre otros motivos, al hecho de que todas ellas son implementables hoy día en el laboratorio con las técnicas actuales de biología molecular.

B. Modelo no restringido de Adleman

Para introducir un modelo de computación molecular basta definir el concepto de tubo y explicitar las operaciones moleculares primitivas del mismo.

Definición: *Un tubo en el modelo no restringido es un multiconjunto finito de cadenas del alfabeto* $\Sigma_{ADN} = \{A, C, G, T\}$.

Las cadenas del alfabeto Σ_{ADN} pueden ser im-

plementadas de manera natural en el laboratorio a través de moléculas de ADN, gracias al direccionamiento de las mismas. En los modelos moleculares que tienen al ADN como substrato computacional, el alfabeto habitual es Σ_{ADN}. No obstante, con frecuencia y por cuestiones técnicas, usaremos como alfabeto de esos modelos otro distinto, de tal manera que cada símbolo del nuevo alfabeto está codificado por un determinado oligo en Σ^*_{ADN} (conjunto de todas las cadenas del alfabeto Σ_{ADN}).

Las instrucciones moleculares básicas del modelo no restringido de Adleman [2] son las siguientes:

- `Extraer`(T, γ): dado un tubo, T, y una cadena, γ, de Σ_{ADN}, devuelve dos tubos:

  $$+(T,\gamma) = \{\{\sigma \in T : \ \gamma \text{ es subcadena de } \sigma\}\}$$
  $$-(T,\gamma) = \{\{\sigma \in T : \ \gamma \text{ no es subcadena de } \sigma\}\}$$

  En esta expresión, la doble llave indica que se trata de un multiconjunto en lugar de un conjunto.
- `Mezclar`(T_1, T_2): dados dos tubos, T_1 y T_2, devuelve un nuevo tubo, $T_1 \cup T_2$, que es la unión de ambos, como multiconjuntos.
- `Amplificar` $(T, \{T_1, T_2\})$: dado un tubo T, devuelve dos tubos, T_1 y T_2 que son copias exactas de T.
- `Detectar`(T): dado un tubo, T, devuelve **SI**, en el caso en que T contenga alguna molécula de ADN, y **NO** en caso contrario.

Obsérvese que en el modelo no restringido de Adleman únicamente la operación molecular `extraer` implementa el paralelismo masivo.

C. *Implementación del experimento de Lipton en el modelo no restringido*

Consideremos el alfabeto $\Sigma = \{a_i, x_i^{j_i}, a_{n+1} : \ 1 \leq i \leq n \wedge \forall i \ (1 \leq i \leq n \rightarrow j_i = 0 \vee j_i = 1)\}$

El tubo de entrada, T_0 es el siguiente multiconjunto:

$$\{\{\sigma \in \Sigma^{2n+1} : \exists j_1 \ldots \exists j_n [(j_1, \ldots, j_n) \in \{0,1\}^n \wedge \sigma = a_1 x_1^{j_1} a_2 x_2^{j_2} \ldots x_n^{j_n} a_{n+1}]\}\}$$

Obsérvese que cada elemento del tubo de ensayo inicial determina de manera natural un camino en el grafo G_n que va desde el nodo a_1 al nodo a_n, y, por tanto, está identificado por una molécula que codifica una valoración relevante para la fórmula.

La idea del experimento de Lipton descrito en la sección 2 sugiere el diseño del siguiente programa molecular que resuelve el problema de la satisfactibilidad de la Lógica Proposicional, en el modelo no restringido de Adleman:

```
Entrada: T_0
         Para i ← 1 hasta p hacer
              T_1 ← T_0;   T_0 ← ∅
              Para j ← 1 hasta r_i hacer
                   T' ← +(T_1, l^1_{i,j})
                   T_1 ← −(T_1, l^1_{i,j})
                   T_0 ← T_0 ∪ T'
              Fin Para
         Fin Para
         Detectar(T_0)
```

El número de operaciones moleculares que se ejecutan en el programa es lineal en el número de literales, k, de la fórmula de entrada (en concreto, k operaciones de `extracción`, k operaciones de `mezcla` y 1 operación de `detección`). Además, el número total de tubos usados es

$$1 + \sum_{i=1}^{p} (3 + 3 \cdot r_i) = 1 + 3p + 3k \in O(k)$$

IV. Verificación formal del programa diseñado

Para verificar formalmente un programa molecular diseñado para resolver un problema, hay que demostrar dos resultados básicos:

(a) Toda molécula del tubo de salida representa una solución correcta del problema (*corrección del programa molecular*); es decir, si el tubo de salida no está vacío, entonces existe alguna solución correcta del problema; con otras palabras, para un problema de decisión, si el programa devuelve SI, entonces la respuesta del problema es SI.

(b) Toda molécula del tubo inicial que codifica una solución correcta del problema debe estar en el tubo de salida (*completitud del programa molecular*); es decir, si el tubo de salida está vacío, entonces no existe ninguna solución correcta del problema; con otras palabras, para un problema de decisión, si el programa devuelve NO, entonces la respuesta del problema es NO.

A continuación, vamos a describir como sistema formal, el programa P diseñado en la sección anterior para resolver el problema **SAT**; de tal manera que demostrar la verificación de dicho programa sea equivalente a establecer la adecuación y completitud del sistema formal asociado.

Un *sistema formal* consta, en esencia, de (a) una *sintaxis*, obtenida a partir de un alfabeto prefijado y de una regla de formación de fórmulas del sistema; (b) una *semántica* que proporciona el concepto de *validez*; y (c) una *deducción* que proporciona el concepto de demostrabilidad de una fórmula en el sistema.

Se dice que un sistema formal es *adecuado* si y sólo si toda fórmula demostrable es válida, y se dice que es *completo* si y sólo si toda fórmula válida es demostrable.

Para cada fórmula proposicional φ en formal normal conjuntiva (escribiremos $\varphi \in E_{\mathbf{SAT}}$) notaremos $I(P, \varphi)$ el conjunto de tubos iniciales del programa P relativos a la fórmula φ. Para cada tubo $T \in I(P, \varphi)$ notaremos $P(\varphi, T)$ el resultado de ejecutar el pro-

grama P con dato de entrada φ y tubo de ensayo inicial T.

Verificar que el programa P resuelve el problema **SAT** consiste en demostrar que para cada fórmula $\varphi \in E_{\mathbf{SAT}}$, y cada tubo inicial, $T \in I(P,\varphi)$, se tiene que $P(\varphi,T) = 1$ si y sólo si la fórmula φ es satisfactible. En esta equivalencia, la implicación directa recibe el nombre de *corrección* del programa P para el problema **SAT**, y la implicación recíproca recibe el nombre de *completitud* del programa P para el problema **SAT**.

El sistema formal, $\mathcal{S}(\mathbf{SAT},P)$, asociado al programa P que resuelve el problema **SAT**, consta de:

(a) Un alfabeto, $\Sigma^* = \Sigma \cup E_{\mathbf{SAT}}$.
(b) Un conjunto de fórmulas: una fórmula es un par ordenado (φ,T), en donde $\varphi \in E_{\mathbf{SAT}}$ y $T \in I(P,\varphi)$.
(c) Un concepto de validez: una fórmula (φ,T) es válida si y sólo si φ es satisfactible.
(d) Un concepto de deducción: una fórmula (φ,T) es demostrable si y sólo si $P(\varphi,T) = 1$.

Se tiene que el programa molecular P resuelve el problema **SAT** si y sólo si el sistema formal asociado, $\mathcal{S}(\mathbf{SAT},P)$ es adecuado y completo.

Para establecer la verificación formal del programa molecular P respecto del problema **SAT**, en primer lugar vamos a proceder a re-etiquetar los tubos que aparecen en el programa diseñado, a fin de individualizarlos y poder hacer un seguimiento pormenorizado de los mismos a lo largo de la ejecución del programa (siguiendo [6]).

```
Entrada:  T_0 (en las condiciones anteriores)
          Para i ← 1 hasta p hacer
               T_{i,0} ← ∅;   T''_{i,0} ← T_{i-1}
               Para j ← 1 hasta r_i hacer
                    T'_{i,j} ← +(T''_{i,j-1}, l^1_{i,j})
                    T''_{i,j} ← −(T''_{i,j-1}, l^1_{i,j})
                    T_{i,j} ← T_{i,j-1} ∪ T'_{i,j}
               Fin Para
               T_i ← T_{i,r_i}
          Fin Para
          Detectar(T_p)
```

Para establecer la adecuación del programa, consideremos las siguientes fórmulas:

- Para cada i $(1 \leq i \leq p)$, se define $\varphi_i \equiv c_1 \wedge ... \wedge c_i$. Sea φ_0 una tautología.
- Para cada i,j tales que $1 \leq i \leq p,\ 1 \leq j \leq r_i$, se define $L_{i,j} \equiv l_{i,1} \vee ... \vee l_{i,j}$.
- Para cada i,j tales que $1 \leq i \leq p,\ 1 \leq j \leq r_i$, se define
$\psi(i,j) \equiv \forall \sigma \in T_{i,j}\ (\sigma(\varphi_{i-1} \wedge L_{i,j}) = 1) \wedge$
$\forall \sigma \in T''_{i,j}\ (\sigma(\varphi_{i-1}) = 1 \wedge \sigma(L_{i,j}) = 0)$
- Para cada i $(1 \leq i \leq p)$, se define
$\theta(i) \equiv \forall j\ (1 \leq j \leq r_i \rightarrow \psi(i,j))$

Teorema 1: *La fórmula θ es un invariante del bucle principal. Es decir, $\forall i\ (1 \leq i \leq p \rightarrow \theta(i))$.*

Prueba: Por inducción sobre i.

En primer lugar veamos, por inducción sobre j, que se verifica $\forall j\ (1 \leq j \leq r_1 \rightarrow \psi(1,j))$.

- La fórmula $\psi(1,1)$ es verdadera, ya que

$$\begin{aligned}
\sigma \in T_{1,1} &\Longrightarrow \sigma \in T'_{1,1} = +(T_0, l^1_{1,1}) \\
&\Longrightarrow \sigma \in T_0 \wedge \sigma(l_{1,1}) = 1 \Longrightarrow \sigma(L_{1,1}) = 1 \\
\sigma \in T''_{1,1} &\Longrightarrow \sigma \in -(T_0, l^1_{1,1}) \\
&\Longrightarrow \sigma \in T_0 \wedge \sigma(l_{1,1}) = 0 \Longrightarrow \sigma(L_{1,1}) = 0
\end{aligned}$$

- Sea $j < r_1$ tal que la fórmula $\psi(1,j)$ es verdadera. Se tiene que

$$\begin{aligned}
\sigma \in T_{1,j+1} &\Longrightarrow \sigma \in T_{1,j} \vee \sigma \in T'_{1,j+1} \\
&\Longrightarrow (\sigma(L_{1,j}) = 1) \vee (\sigma \in T''_{1,j} \wedge \sigma(l_{1,j+1}) = 1) \\
&\Longrightarrow \sigma(L_{1,j+1}) = 1 \\
\sigma \in T''_{1,j+1} &\Longrightarrow \sigma \in T''_{1,j} \ \wedge\ \sigma(l_{1,j+1}) = 0 \\
&\Longrightarrow \sigma(L_{1,j}) = 0 = \sigma(l_{1,j+1}) \Longrightarrow \sigma(L_{1,j+1}) = 0
\end{aligned}$$

Sea $i < p$ tal que $\theta(i)$ es verdadera. Veamos que $\forall j\ (1 \leq j \leq r_{i+1} \rightarrow \psi(i+1,j))$, por inducción sobre j.

- La fórmula $\psi(i+1,1)$ es verdadera, ya que

$$\begin{aligned}
\sigma \in T_{i+1,1} &\Longrightarrow \sigma \in T_i \wedge \sigma(l_{i+1,1}) = 1 \\
&\Longrightarrow \sigma(\varphi_i) = 1 \wedge \sigma(L_{i+1,1}) = 1 \\
\sigma \in T''_{i+1,1} &\Longrightarrow \sigma \in T_i \wedge \sigma(l_{i+1,1}) = 0 \\
&\Longrightarrow \sigma(\varphi_i) = 1 \wedge \sigma(L_{i+1,1}) = 0
\end{aligned}$$

- Sea $j < r_{i+1}$ tal que la fórmula $\psi(i+1,j)$ es verdadera. Se tiene que

$$\begin{aligned}
\sigma \in T_{i+1,j+1} &\Longrightarrow \sigma \in T_{i+1,j} \vee \sigma \in T'_{i+1,j+1} \\
&\Longrightarrow (\sigma(\varphi_i \wedge L_{i+1,j}) = 1) \vee \\
&\quad \vee\ (\sigma \in T''_{i+1,j} \wedge \sigma(l_{i+1,j+1}) = 1) \\
&\Longrightarrow (\sigma(\varphi_i \wedge L_{i+1,j+1}) = 1) \vee \\
&\quad \vee\ (\sigma(\varphi_i) = 1\ \wedge \sigma(L_{i+1,j}) = 0 \wedge \\
&\qquad \wedge\ \sigma(l_{i+1,j+1}) = 1) \\
&\Longrightarrow \sigma(\varphi_i \wedge L_{i+1,j+1}) = 1 \\
\sigma \in T''_{i+1,j+1} &\Longrightarrow \sigma \in T''_{i+1,j} \wedge \sigma(l_{i+1,j+1}) = 0 \\
&\Longrightarrow \sigma(\varphi_i) = 1 \wedge \sigma(L_{i+1,j}) = 0 \wedge \\
&\quad \wedge\ \sigma(l_{i+1,j+1}) = 0 \\
&\Longrightarrow \sigma(\varphi_i) = 1 \wedge \sigma(L_{i+1,j+1}) = 0
\end{aligned}$$

□

Corolario 2 (Corrección): *Toda molécula del tubo de salida codifica una valoración que hace satisfactible la fórmula φ. Es decir, $\forall \sigma \in T_p\ (\sigma(\varphi) = 1)$.*

Prueba: Se tiene que $T_p = T_{p,r_p}$ y que la fórmula $\psi(p,r_p)$ es verdadera (por serlo $\theta(p)$). En consecuencia, para cada $\sigma \in T_p = T_{p,r_p}$ se tiene que $\sigma(\varphi) = \sigma(\varphi_{p-1} \wedge L_{p,r_p}) = 1$.

□

Acabamos de probar que toda molécula del tubo de salida codifica una asignación de verdad que hace satisfactible la fórmula. Veamos finalmente que si el tubo de salida es vacío, entonces la fórmula no es satisfactible; es decir, veamos que toda molécula del tubo inicial que codifique una valoración que hace verdadera la fórmula, supera todos los filtros del programa y permanece en el tubo de salida.

Para establecer la completitud del programa, consideramos la siguiente fórmula:

$$\delta(i) \equiv \forall \sigma \in T_0 \ (\sigma(c_1 \wedge \ldots \wedge c_p) = 1 \rightarrow \sigma \in T_i)$$

para cada i $(1 \leqslant i \leqslant p)$.

Teorema 3: *La fórmula δ es un invariante del bucle principal. Es decir, $\forall i \ (1 \leq i \leq p \rightarrow \delta(i))$.*

Prueba: Sea $\sigma \in T_0$ tal que $\sigma(c_1 \wedge ... \wedge c_p) = 1$. Entonces para cada i $(1 \leq i \leq p)$ existe j $(1 \leq j \leq r_i)$ tal que $\sigma(l_{i,j}) = 1$. Notemos $m_i = \text{mín}\{j : 1 \leq j \leq r_i \wedge \sigma(l_{i,j}) = 1\}$. Probemos por inducción sobre i que $\forall i \ (1 \leq i \leq p \rightarrow \sigma \in T_i)$.

Por definición se tiene que $\forall j \ (1 \leq j < m_1 \rightarrow \sigma(l_{1,j}) = 0)$. Como $\sigma \in T_0 = T''_{1,0}$ resulta que $\forall j \ (1 \leq j < m_1 \rightarrow \sigma \in T''_{1,j})$. Luego $\sigma \in T''_{1,m_1-1}$. Teniendo presente que $\sigma(l_{1,m_1}) = 1$, se deduce que $\sigma \in +(T''_{1,m_1-1}, l^1_{1,m_1}) = T'_{1,m_1} \subseteq T_{1,r_1} = T_1$.

La prueba en el paso inductivo es análoga al caso base, usando la hipótesis de inducción.

□

Corolario 4 (Completitud): *Toda molécula del tubo de ensayo inicial que codifica una valoración que hace satisfactible la fórmula φ, está en el tubo de salida. Es decir, $\forall \sigma \in T_0(\sigma(\varphi) = 1 \rightarrow \sigma \in T_p)$.*

Prueba: Basta tener presente que tras la ejecución del programa, la fórmula $\delta(p)$ es verdadera.

□

V. Extensión del experimento de Lipton

En el experimento de Lipton se proporciona una solución molecular del problema de la satisfactibilidad de fórmulas proposicionales en forma normal conjuntiva. Veamos seguidamente cómo es posible generalizar dicho experimento a fin de decidir la satisfactibilidad de una fórmula proposicional arbitraria, en el modelo no restringido de Adleman [7].

Proposición 3.1. *Sea φ una fórmula proposicional con k conectivas lógicas. Sea T un tubo que contiene moléculas que codifican valoraciones relevantes para φ. Entonces, con $O(k)$ operaciones de extracción, unión y amplificación, se pueden generar los tubos siguientes:*

$$\begin{cases} T^1(T,\varphi) = \{\{\sigma \in T : \ \sigma(\varphi) = 1\}\} \\ T^0(T,\varphi) = \{\{\sigma \in T : \ \sigma(\varphi) = 0\}\} \end{cases}$$

Demostración: Por inducción fuerte sobre el número de conectivas lógicas, k, de la fórmula.

- Si $\varphi \equiv x_i$ y T es un tubo que contiene codificaciones relevantes para φ, entonces
  $T^1(T,\varphi) = +(T, x^1_i)$ y $T^0(T,\varphi) = -(T, x^1_i)$
  Si $\varphi \equiv \neg x_i$ y T es un tubo que contiene codificaciones relevantes para φ, entonces
  $T^1(T,\varphi) = +(T, x^0_i)$ y $T^0(T,\varphi) = -(T, x^0_i)$
- Supongamos cierto el resultado para cada $r \leq k$ y sea φ una fórmula con $k+1$ conectivas lógicas. Sea T un tubo que codifica valoraciones relevantes para φ.
  Si $\varphi \equiv \neg\varphi_1$, entonces por hipótesis de inducción con $O(k)$ operaciones de extracción, unión y amplificación, se pueden obtener los tubos

  $$\begin{cases} T^1(T,\varphi_1) = \{\{\sigma \in T : \ \sigma(\varphi_1) = 1\}\} \\ T^0(T,\varphi_1) = \{\{\sigma \in T : \ \sigma(\varphi_1) = 0\}\} \end{cases}$$

  Entonces basta devolver los tubos

  $$\begin{cases} T^1(T,\varphi) = T^0(T,\varphi_1) \\ T^0(T,\varphi) = T^1(T,\varphi_1) \end{cases}$$

  Sea $\varphi \equiv \varphi_1 \vee \varphi_2$. Sea s_i el número de conectivas lógicas de φ_i. Por hipótesis de inducción, con $O(s_1)$ operaciones de extracción, unión y amplificación se pueden obtener los tubos

  $$\begin{cases} T^1(T,\varphi_1) = \{\{\sigma \in T : \ \sigma(\varphi_1) = 1\}\} \\ T^0(T,\varphi_1) = \{\{\sigma \in T : \ \sigma(\varphi_1) = 0\}\} \end{cases}$$

  Amplificamos el tubo $T^0(T,\varphi_1)$ obteniendo dos copias. Entonces, por hipótesis de inducción, con $O(s_2)$ operaciones de extracción, unión y amplificación obtenemos los tubos $T^1(T^0(T,\varphi_1),\varphi_2)$ y $T^0(T^0(T,\varphi_1),\varphi_2)$. En tal situación, basta devolver los tubos

  $$T^1(T,\varphi_1 \vee \varphi_2) = T^1(T,\varphi_1) \cup T^1(T^0(T,\varphi_1),\varphi_2)$$
  $$T^0(T,\varphi_1 \vee \varphi_2) = T^0(T^0(T,\varphi_1),\varphi_2)$$

  Si $\varphi \equiv \varphi_1 \wedge \varphi_2$, entonces el razonamiento es análogo al realizado en el caso de la disyunción.

□

Corolario 3.2. *El problema de la satisfactibilidad para fórmulas proposicionales se puede resolver realizando una operación de detectar y $O(k)$ operaciones de extracción, unión y amplificación (en donde k es el número de conectivas lógicas de la fórmula).*

Demostración: Dada una fórmula proposicional, φ, con n variables y k conectivas lógicas procedemos, al igual que en la preparación del experimento de Lipton, diseñando el grafo dirigido asociado, G_n, y elaborando, a partir de éste, el tubo de ensayo inicial, T_0, que contiene cadenas de ADN que codifican todas las valoraciones relevantes con dominio $\text{Var}(\varphi)$. De la proposición anterior resulta que realizando $O(k)$ operaciones de extracción, unión y amplificación se obtienen los tubos

$$\begin{cases} T^1(T_0,\varphi) = \{\{\sigma \in T_0 : \ \sigma(\varphi) = 1\}\} \\ T^0(T_0,\varphi) = \{\{\sigma \in T_0 : \ \sigma(\varphi) = 0\}\} \end{cases}$$

Basta aplicar la operación `detectar`$(T^1(T_0,\varphi))$.

□

VI. Conclusiones

En este trabajo se ha presentado una formalización computacional del experimento de Lipton que resuelve el problema de la satisfactibilidad de la Lógica Proposicional. El marco formal elegido ha sido un modelo de computación molecular sin memoria basado en **ADN**: el modelo no restringido de Adleman. El experimento de Lipton se ha implementado en el modelo a través de un programa molecular que, a su vez, se ha descrito como un sistema formal de tal manera que la verificación del programa se ha establecido probando la adecuación y completitud del sistema asociado. Finalmente se ha propuesto una extensión del experimento de Lipton a fórmulas proposicionales arbitrarias.

Creemos que la verificación formal de programas moleculares es un paso fundamental para su procesamiento a través de sistemas de razonamiento automático como ACL2 o PVS, con los que estamos trabajando en la actualidad.

Referencias

[1] Adleman, L. Molecular Computation of Solutions to Combinatorial Problems, *Science*, 268, November 1994, 1021–1024.

[2] Adleman, L. On constructing a molecular computer, in *DNA based computers*, R.J. Lipton and E.B. Baum, eds., American Mathematical Society, 1996, 1–22.

[3] Beaver, D. A universal molecular computer, in *DNA based computers*, R.J. Lipton and E.B. Baum, eds., American Mathematical Society, 1996, 29–36.

[4] Garey M.R.; Johnson D.S. *Computers and intractability*. W.H. Freeman and Company, New York, 1979.

[5] Lipton R.J. DNA Solution of Hard Computational Problems, *Science*, 268, April 1995, 542–545.

[6] Pérez–Jiménez, M.J.; Sancho, F.; Graciani, C.; Romero, A. Soluciones moleculares del problema SAT de la Lógica Proposicional. En A. Nepomuceno y otros (eds), *Lógica, Lenguaje e Información*, JOLL'2000, 243–252, Ed. Kronos, 2000.

[7] Pérez–Jiménez, M.J. Computación molecular sin memoria basada en ADN. En A. Nepomuceno, J.F. Quesada y F. Salguero (eds), *Información: Tratamiento y Representación*, cap. 15, 271–313, Servicio de Publicaciones de la Universidad de Sevilla, 2001.