Formalización y resolución de problemas

Laboratorio 4
Lógica para Computación
Otoño 2024

El objetivo de este laboratorio es poner en práctica una de las posibles aplicaciones de la lógica formal, específicamente en la resolución de problemas. A grandes rasgos, la idea es formalizar problemas enunciados en el lenguaje natural para hallar soluciones de forma automática mediante herramientas de decisión de satisfacibilidad. En el primer ejercicio vamos a poder aprovechar la implementación realizada en el Laboratorio 3 basada en el Método de Tableaux. Para los otros ejercicios, la complejidad computacional del problema de decisión de satisfacibilidad nos condiciona a usar una herramienta de porte industrial. En esos casos vamos a usar el *SMT solver* CVC5 que es de código abierto y ofrece un entorno de ejecución online accesible a través del navegador web [1].

Ejercicios

1. (**Veraces y mentirosos**). En una extraña isla, viven tan sólo dos tipos de habitantes: los *caballeros*, que siempre dicen la verdad, y los *escuderos*, que siempre mienten. Además, los forasteros no pueden distinguir el tipo de los habitantes a simple vista.
   En una ocasión, usted visita la isla y se encuentra con tres habitantes A, B y C, entablando un diálogo. Hay distintas versiones de como fue exactamente dicho diálogo, las cuales se presentan a continuación. En cualquier caso, el objetivo es determinar, cuándo sea posible, el tipo de los tres habitantes.

   1.
   Usted a todos: ¿Quiénes de ustedes son escuderos?.
   A: Todos somos escuderos.
   B: Solo uno de nosotros es caballero.

   2.
   Usted a todos: ¿Quiénes de ustedes son escuderos?.
   A: Todos somos escuderos.
   B: Solo uno de nosotros es escudero.

   3.
   Usted a A: ¿Eres caballero o escudero?.
   A: ... (*Responde algo, pero no se le entiende bien*).
   Usted a B: ¿Qué es lo que dijo A?.
   B: A dijo que es un escudero.
   C interviene: No le creas a B, está mintiendo.

[1] Ver Teorías de satisfacibilidad módulo.

Deberá completar su formalización en el archivo `Lab4.hs` de la siguiente manera:

- Declarar las variables que se usarán en la formalización, incluyendo un comentario de como se interpretan.
- Completar la formalización como una fórmula de tipo `L`.
- Completar el comentario con la respuesta al problema, según el resultado que se obtiene usando las funciones implementadas en el Laboratorio 3.

Sugerencia. En general, pensar:

- ¿Bajo qué condiciones, la declaración de un habitante es verdadera?.
- ¿Bajo qué condiciones, la declaración de un habitante es falsa?.
- ¿Cómo podemos expresar dichas condiciones formalmente?

2. **(Planificación de vigilancia).** Para garantizar la seguridad en grandes espacios de manera eficiente han surgido sistemas de vigilancia autónomos con los cuales vigilantes de seguridad robotizados (de aquí en más, robots) se ocupan de la protección perimetral patrullando sin ayuda humana y detectando objetos, actividades o incluso temperaturas anómalas.[2] La efectividad de estos sistemas de vigilancia depende, en parte, de una planificación apropiada entre los distintos robots disponibles y las distintas zonas a vigilar.

   En este ejercicio nos interesa modelar los posibles escenarios de un sistema de vigilancia que consiste en r robots, z zonas de vigilancia y t franjas temporales. Dichos valores conforman los parámetros del problema. Por ejemplo, si tenemos $r = 2$, $z = 2$ y $t = 4$, las figuras 1 y 2 nos muestran dos de las posibles planificaciones de vigilancia en la cual se cumplen las siguientes condiciones obviamente deseables:

   A. Todo robot en cualquier momento vigila alguna zona.

   B. Nunca asignamos más de un robot en la misma zona.

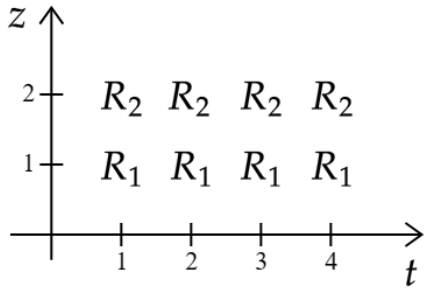


Figura 1: Siempre el mismo robot en cada zona.

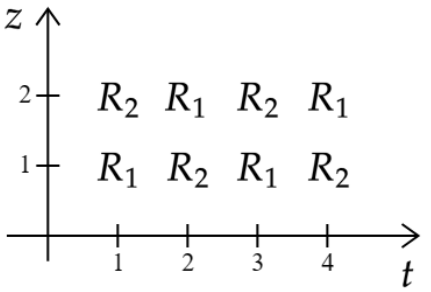


Figura 2: Rotando robots por zona.

   Decimos que una planificación que cumple con las condiciones A y B es *básica*.

   En este ejercicio, se pide lo siguiente:

   1. ¿Es posible realizar una planificación básica bajo las siguientes condiciones sobre los parámetros del problema? Justifique.
      i. $r > z$
      ii. $r < z$

   2. ¿Es posible en una planificación básica que un robot se encargue de vigilar más de una zona durante una misma franja temporal? ¿Depende de alguna condición? Justifique.

   3. Formalizar el problema de realizar una planificación básica para r robots, z zonas de vigilancia y t franjas temporales. La respuesta quedará expresada mediante la función de Haskell:

$$\textbf{planAB :: Nat} \rightarrow \textbf{Nat} \rightarrow \textbf{Nat} \rightarrow \textbf{L}$$

[2] El primer robot con estas capacidades fue desarrollado entre 1966 y 1972, nombrado Shakey, siendo el primer robot autónomo capaz de comprender y razonar sobre su entorno.

4. Encontrar una planificación básica cuando $r = 3$, $z = 4$ y $t = 5$ usando la función **planAB** y CVC5.

   Nota sobre la entrega: la formalización que produzca la planificación solicitada será entregada en el script `planAB.smt`. *Además, deberá graficar las planificación encontrada sobre la imagen* `planAB.png`

5. En escenarios más realistas surge la necesidad de asegurar otras condiciones además de las mencionadas anteriormente. Ahora vamos a extender el problema considerando también que hay robots con mejor rendimiento que otros y que hay zonas con mayor importancia que otras, pretendiendo que se cumpla la siguiente nueva condición:

   C. Los mejores robots son asignados en las zonas con mayor importancia.

   Tanto el rendimiento de los robots como la importancia de las zonas se pueden cuantificar usando números naturales. Luego, podemos asociarle a cada robot (o zona) su rendimiento (o importancia) usando una función finita que podemos codificar como una lista de tipo [**(Nat, Nat)**], es decir una tabla.

   Para esta parte, deberá formalizar el problema de realizar una planificación extendida para r robots, z zonas de vigilancia y t franjas temporales. La respuesta quedará expresada mediante la función de Haskell:

   planABC :: Nat → Nat → Nat → [(Nat, Nat)] → [(Nat, Nat)] → L

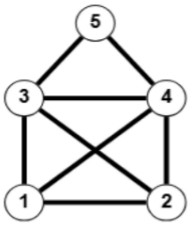
6. Encontrar una planificación extendida cuando $r = 3$, $z = 3$ y $t = 5$ usando la función **planABC** y CVC5. La información de rendimiento e importancia sobre robots y zonas queda determinada por las siguientes tablas:

Robot	Rendimiento
1	200
2	150
3	100

Zona	Importancia
1	100
2	230
3	100

   Nota sobre la entrega: la formalización que produzca la planificación solicitada será entregada en el script `planABC.smt`. *Además, deberá graficar la planificación encontrada sobre la imagen* `planABC.png`

3. **(Clique de grafo).** En matemáticas, un *grafo* (no dirigido) $G = (V, A)$ es una estructura formada por un conjunto finito de vértices V y un conjunto finito de aristas E entre vértices. Cuando existe una arista entre dos vértices decimos que son adyacentes. En nuestra codificación de grafos, siempre vamos a identificar los vértices V enumerándolos en el rango $1..|V|$ y representamos las aristas E mediante una lista de tipo **[(Nat, Nat)]**, entendida como una relación binaria dada por extensión, donde cada pareja (v_1, v_2) nos dice que v_1 y v_2 son adyacentes. Como el grafo es no dirigido, siempre que tenemos (v_1, v_2) también tenemos (v_2, v_1). Por lo tanto, un grafo es un objeto de tipo **(Nat, [(Nat, Nat)])**. Por ejemplo, el grafo de la figura 3 tiene 5 vértices y 8 aristas, y se usan $8 \times 2 = 16$ parejas para codificar las aristas.



(5, [(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5),
(2, 1), (3, 1), (4, 1), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4)])

Figura 3: A la izquierda un grafo en su representación habitual como diagrama y a la derecha nuestra codificación del mismo como un objeto de tipo **(Nat, [(Nat, Nat)])**. El orden en el que se listan las parejas es irrelevante y se asume que no se repiten.

Un *clique* de un grafo es un subconjunto de vértices del grafo tal que todo par de vértices distintos son adyacentes. En otras palabras, un clique es un subgrafo del grafo original en el que cada vértice está conectado a todos los demás vértices del subgrafo (un clique siempre es un grafo completo).
En ciencias de la computación, el *problema del k-clique* consiste en dado un grafo, decidir si existe en este un clique formado por k vértices (decimos tamaño k). Naturalmente, el problema correspondiente de optimización consiste en maximizar k y por lo tanto encontrar el clique más grande posible, el cual a lo sumo tendrá el mismo tamaño que el grafo dado. Cuando identificamos un clique es conveniente describirlo listando los vertices que lo conforman. Por ejemplo, en el grafo de la figura 3 tenemos:

- Cinco 1-cliques: $[1]$, $[2]$, $[3]$, $[4]$ y $[5]$.
- Ocho 2-cliques: $[1, 2]$, $[1, 3]$, $[1, 4]$, $[2, 3]$, $[2, 4]$, $[3, 4]$, $[3, 5]$ y $[4, 5]$.
- Cuatro 3-cliques: $[1, 2, 3]$, $[1, 2, 4]$, $[2, 3, 4]$ y $[3, 4, 5]$.
- Un 4-clique: $[1, 2, 3, 4]$. Este es el clique más grande.
- Ningún 5-clique.

Se trata de un problema con aplicaciones muy amplias y explicitas en bioinformática, química computacional y otras ciencias.[3] Como ejemplo concreto, el grafo de la figura

[3] Algunas de estas aplicaciones son: identificación y predicción de proteínas asociadas a enfermedades, estudio de la organización del genoma, representación de estructuras moleculares y sus interacciones para el desarrollo de fármacos, análisis de interacciones en redes sociales, entre otros.

4 modela las interacciones entre proteínas en la levadura común (Saccharomyces cerevisiae). Este grafo de 209 vértices (proteínas) y 1776 aristas (interacciones) muestra cómo las proteínas se conectan y comunican dentro de la célula. Los grupos densamente interconectados de proteínas se pueden identificar mediante cliques en el grafo y sugieren funciones específicas dentro de la célula que son de interés para los especialistas. El clique más grande en este grafo tiene tamaño 12.

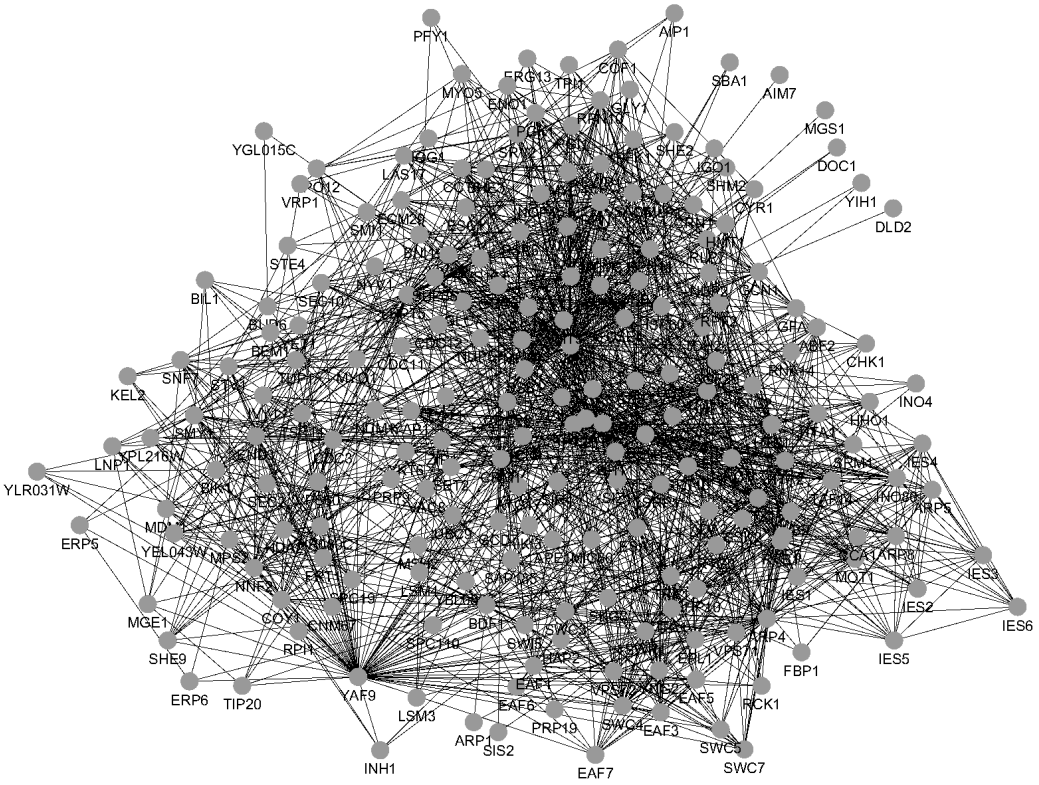


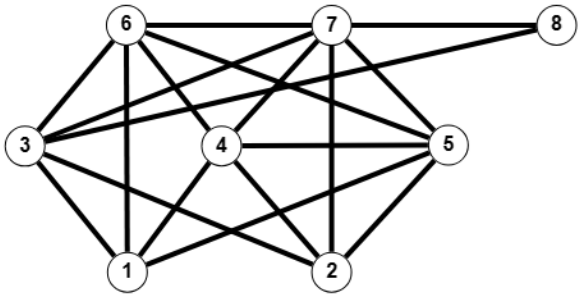
Figura 4: Interacción de proteínas en *Saccharomyces cerevisiae*.

En este ejercicio, se pide lo siguiente:

1. Formalizar el problema del k-clique para un grafo y valor k arbitrarios. La respuesta quedará expresada mediante la función de Haskell:

$$\textbf{kClique :: G} \rightarrow \textbf{Nat} \rightarrow \textbf{L}$$

2. Considere el siguiente grafo



Buscar dos 3-cliques usando la función **kClique** y CVC5. Exhibir sus vértices.

Nota sobre la entrega: las formalizaciones utilizadas serán entregadas en los scripts `3Clique1.smt` *y* `3Clique2.smt` *respectivamente.*

3. Si podemos resolver el problema (de decisión) del k-clique, entonces también podemos resolver el problema correspondiente de optimización viéndolo como una búsqueda: encontrar el valor de k más grande con el cual se puede resolver el problema del k-clique.

   ¿Cuál es el clique más grande en el grafo del punto anterior?.

   Nota sobre la entrega: la formalización utilizada será entregada en el script `greatestClique.smt`.

4. Un *clique maximal* en un grafo es un clique que no está contenido en ningún otro clique del grafo. En otras palabras, un clique maximal es aquel al que no se le pueden agregar más vértices sin dejar de ser un clique.
   Siendo más precisos, dado un grafo, decimos que un k-clique C del grafo es *maximal* si y solo si no existe un vértice del grafo fuera de C tal que sea adyacente a cada uno de los k vértices de C. Volviendo al ejemplo de la figura 3 tenemos:

   - El 3-clique $[1, 2, 3]$ no es maximal, porque está contenido en el clique $[1, 2, 3, 4]$. O sea, existe un vértice (en este caso el 4) que está fuera del 3-clique y que es adyacente a todos sus vértices.
   - El 3-clique $[3, 4, 5]$ es maximal. Notar que esto no significa que sea el clique mas grande en el grafo.
   - El 4-clique $[1, 2, 3, 4]$ es maximal, y es el clique más grande.

   Para esta parte, deberá formalizar el problema del k-clique maximal para un grafo y valor k arbitrarios. La respuesta quedará expresada mediante la función de Haskell:

   maxkClique :: G → Nat → L

5. Considere nuevamente el grafo de la parte 2. Determinar si existe un 3-clique maximal usando la función **maxkClique** y CVC5. En caso afirmativo, exhibirlo.

   Nota sobre la entrega: la formalización utilizada será entregada en el script `max3Clique.smt`.

Sugerencia. Para los ejercicios 2 y 3 se recomienda implementar las siguientes funciones de soporte que le permitirán construir en Haskell fórmulas de LP paramétricas. Además, estas se apoyan en el trabajo realizado en el Laboratorio 3, por lo cual se asume que el módulo de dicho laboratorio se encuentra disponible en el mismo directorio que el laboratorio actual.

- **bigAnd :: [Nat] → (Nat → L) → L**
  Dada una lista de índices I y una fórmula indexada α, devuelve la conjuntaria de α sobre I, o sea $\bigwedge_{i \in I} \alpha_i$.
  Ejemplo: Para representar $\bigwedge_{i \in [1,2,3]} p_i$ en Haskell escribimos
  bigAnd $[1,2,3]$ $(\lambda i.\, \mathsf{V}\,(\text{“p”}\mathsf{++}\,\mathsf{show\,i}))$ = $(\mathsf{V}\ \text{“}p1\text{”})\ \mathsf{And}\ (\mathsf{V}\ \text{“}p2\text{”})\ \mathsf{And}\ (\mathsf{V}\ \text{“}p3\text{”})$
- **bigOr :: [Nat] → (Nat → L) → L**
  Dada una lista de índices I y una fórmula indexada α, devuelve la disyuntoria de α sobre I, o sea $\bigvee_{i \in I} \alpha_i$.
  Ejemplo: Para representar $\bigvee_{i \in [1,2,3]} p_i$ en Haskell escribimos
  bigOr $[1,2,3]$ $(\lambda i.\, \mathsf{V}\,(\text{“p”}\mathsf{++}\,\mathsf{show\,i}))$ = $(\mathsf{V}\ \text{“}p1\text{”})\ \mathsf{Or}\ (\mathsf{V}\ \text{“}p2\text{”})\ \mathsf{Or}\ (\mathsf{V}\ \text{“}p3\text{”})$
- **v :: Var → Nat → L**
  Dada una variable p y un índice i, devuelve la variable p indexada en i.
  Ejemplo: Para representar p_1 en Haskell escribimos
  $\mathsf{v}\ \text{“}p\text{”}\ 1\ =\ \mathsf{V}\ \text{“}p1\text{”}$
- **v3 :: Var → Nat → Nat → Nat → L**
  Dada una variable p y tres índices i, j y k, devuelve la variable p indexada en i, j y k. Los índices deberán quedar separados por una barra baja.
  Ejemplo: Para representar $p_{1,2,3}$ en Haskell escribimos
  $\mathsf{v3}\ \text{“}p\text{”}\ 1\ 2\ 3\ =\ \mathsf{V}\ \text{“}p1\_2\_3\text{”}$

Ilustramos con un ejemplo como se pueden combinar las funciones anteriores. Para cualquier par de índices $I = [1..n]$ y $J = [1..m]$, podemos por ejemplo representar en Haskell la siguiente conjunción de disyunciones de literales triplemente indexados (una FNC)

$$\bigwedge_{i \in I} \bigvee_{j \in J} p_{i,j,i}$$

escribiendo

$$\mathsf{bigAnd}\ [1..n]\ (\lambda i.\ \mathsf{bigOr}\ [1..m]\ (\lambda j.\ \mathsf{v2}\ \text{“}p\text{”}\ i\ j\ i))$$

En particular, si $n = 2$ y $m = 2$, esto construye en Haskell la fórmula de LP

$$\mathsf{Bin}\,(\mathsf{Bin}\,(\mathsf{V}\ \text{“}p1\_1\_1\text{”})\ \mathsf{Or}\,(\mathsf{V}\ \text{“}p1\_2\_1\text{”}))\ \mathsf{And}\ (\mathsf{Bin}\,(\mathsf{V}\ \text{“}p2\_1\_2\text{”})\ \mathsf{Or}\,(\mathsf{V}\ \text{“}p2\_2\_2\text{”}))$$

que en la sintaxis del formato SMT-LIB (requerido para trabajar con CVC5) se escribe con notación prefija y parentizada

$$(\mathsf{and}\ (\mathsf{or}\ p1\_1\_1\ p1\_2\_1)\ (\mathsf{or}\ p2\_1\_2\ p2\_2\_2))$$

y en nuestra sintaxis concreta se escribe

$$(p_{1,1,1} \vee p_{1,2,1}) \wedge (p_{2,1,2} \vee p_{2,2,2})$$

Afortunadamente, no vamos a necesitar escribir a mano en la sintaxis del formato SMT-LIB, porque disponemos de la función auxiliar toPrefix que convierte a dicha sintaxis. De esta manera, Haskell nos servirá como interfaz para usar CVC5.