Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Primer Semestre de 2018

IIC2213 - Lógica para ciencia de la Computación

Tarea 2 - Raimundo Herrera (rjherrera@uc.cl)

Problema 1

1. Por el teorema de Cook-Levin, sabemos que SAT es NP (de hecho es NP-completo).

   Mostrar que SAT pertenece a $\hat{\text{NP}}$ equivale a mostrar que existen un polinomio P y un lenguaje M tal que $x \in \text{SAT}$ si y solo si existe $y \in A^*$ con $|y| \leq P(|x|)$ y $x\#y$ pertenece a M.

   Aprovechando la definición de SAT, se sabe que dado un elemento z perteneciente a A^*, se tiene que $z \in \text{SAT}$ si y solo si existe una valuación w tal que z con esa valuación evalúa a 1.

   Sea M^* un lenguaje sobre $A \cup \{\#\}$ tal que $M^* = \{z\#w \mid w \text{ es una valuación que hace verdad a } z\}$, esto es, el conjunto de las formulas concatenadas con las asignaciones que las hacen verdad. Como SAT está en NP, la verificación de si w es una solución para z (que en SAT se entiende como una asignación que haga verdad a z) toma tiempo polinomial, por lo que $M^* \in \text{PTIME}$.

   Por otro lado, el certificado o asignación w es de tamaño polinomial respecto al tamaño de z puesto que de lo contrario no se podría verificar su validez en tiempo polinomial, de modo que $|w| \leq P(|z|)$. También es posible verlo puesto que las valuaciones tienen a lo más el largo de todos los literales presentes en la fórmula, por lo que con dicha cota se asegura la relación de tamaño.

   Con esto, se tiene que SAT pertenece a $\hat{\text{NP}}$ puesto que existe un polinomio P, en particular en este caso es lineal respecto al largo de la formula o entrada del problema z, y un lenguaje M, en este caso M^* tal que se cumplen las condiciones pedidas.

   De este modo, la intuición para el lenguaje M en el caso de SAT corresponde, como se dijo, a todos los problemas y sus certificados (o valuaciones que lo hacen verdad) concatenados por el caracter #, esto es, la definición de M^* otorgada más arriba, esto nos da luces de que en dicho lenguaje se encuentran todas las formulas satisfacibles junto con la valuación que prueba dicha satisfacibilidad.

2. Demostrar lo pedido equivale a mostrar que $\hat{\text{NP}} \subseteq \text{NP}$ y que $\text{NP} \subseteq \hat{\text{NP}}$. Es fácil ver que si SAT pertenece a $\hat{\text{NP}}$, entonces NP es un subconjunto de $\hat{\text{NP}}$, esto debido a que como se mencionó anteriormente, SAT es NP-completo, de este modo, de todos los lenguajes L pertenecientes a NP, existe una reducción a SAT, por lo que si de todos los lenguajes que están en NP existe una reducción a uno que está en $\hat{\text{NP}}$, entonces $\text{NP} \subseteq \hat{\text{NP}}$.

   Por el otro lado, hay que mostrar que $\hat{\text{NP}} \subseteq \text{NP}$. Utilizando la definición de NP basada en maquinas de Turing no deterministas, se tiene que para que un problema esté en NP, entonces debe existir una maquina de Turing no determinista tal que resuelve el problema en tiempo polinomial.

   Por la definicion de $\hat{\text{NP}}$, se sabe que si se obtiene un elemento y, es posible verificar la pertenencia de este elemento concatenado con x al lenguaje M en tiempo polinomial. Sea MT una máquina de Turing tal que verifica si y es una solución para x, esta maquina es de tiempo polinomial. Se puede definir una maquina de Turing no determinista tal que sea la composición de la máquina MT y que en primer lugar adivine una solución y para x y luego compute o llame a MT para saber si dicha solución es válida, con esa construcción, se tiene que $\hat{\text{NP}} \subseteq \text{NP}$.

Problema 2

- Sea f una función arbitraria tal que para todo φ una formula arbitraria cualquiera, entrega una valuación τ tal que dicha valuación satisface a φ, si esque φ es satisfacible. Para mostrar que f es una PC, entonces hay que mostrar que existe un lenguaje en $\hat{\text{NP}}$ con P y M sobre $A \cup \{\#\}$, tal que $|f(w)| \leq P(|w|)$ para todas las palabras w en A^* y tal que $w\#f(w)$ pertenece a M.

Para mostrar la existencia, podemos tomar SAT, que ya sabemos que pertenece a $\hat{\text{NP}}$. Ahora, considerando para SAT el lenguaje M tal que $M = \{z\#w \mid w \text{ es una solución válida para } z\}$, se tiene que en el caso particular de SAT, z y w se interpretan de tal modo que $M = \{\varphi\#\tau \mid \tau \text{ es una valuación que hace verdad a } \varphi\}$ es la forma de definir dicho lenguaje. De este modo, por las características de SAT, sabemos que toda valuación τ es verificable en tiempo polinomial sobre φ, con esto, tenemos que dicho M cumple lo solicitado (2).

Además, el largo de τ es polinomial respecto a φ, puesto que el largo de una valuación es menor o igual a la cantidad de literales presentes en dicha fórmula, por lo que se cumple que $|f(w)| \leq P(|w|)$, si se considera a $f(w)$ como τ y a w como φ (1).

Por lo tanto, toda formula f que reciba una fórmula y entregue una valuación que la satisfaga, es una función PC.

- En primer lugar hay que mostrar que si SAT pertenece a PTIME, entonces existe una función PC f que recibe una fórmula φ y entrega la valuación τ que la satisface, y es computable en tiempo polinomial. Esta dirección es inmediata.

  Recordar que un lenguaje está en PTIME si y solo si existe una máquina de Turing determinista, tal que corre en tiempo polinomial para todos los inputs, entregando 1 o 0 según si x pertenece o no al lenguaje, respectivamente.

  Si SAT pertenece a PTIME, entonces se sabe que existe una máquina de Turing determinista de tiempo polinomial que dado un x determina si es satisfacible o no. Dado eso, es evidente que existen las funciones PC en tiempo polinomial y son justamente las utilizadas para determinar si es satisfacible o no.

  En segundo lugar hay que mostrar que si existe una función PC f que recibe una fórmula φ y entrega la valuación τ que la satisface, y que es computable en tiempo polinomial, entonces SAT pertenece a PTIME.

  Si existe dicha función, entonces basta con crear una máquina de Turing que tenga como módulo interior, la computación de la valuación, que es computable en tiempo polinomial, y que luego verifique que dicha solución satisface a la fórmula en cuestión, como ambas operaciones son polinomiales, entonces la máquina de Turing es de tiempo polinomial, y al existir dicha máquina, entonces SAT pertenece a PTIME.

- En primer lugar hay que mostrar que si P=NP entonces todas las funciones PC son computables en tiempo polinomial.

  Si P=NP, entonces todos los problemas NP pueden ser resueltos en tiempo polinomial por una máquina de Turing determinista, por lo que, en particular, para todos esos problemas existe una forma de obtener una solución en tiempo polinomial. Si dicha solución se puede obtener en tiempo polinomial, entonces computar f también lo es, ya que justamente consiste en entregar dicho certificado o solución.

  En segundo lugar hay que mostrar que si todas las funciones PC son computables en tiempo polinomial entonces P=NP.

  Si todas las funciones PC son computables en tiempo polinomial, entonces en particular la función PC para SAT es computable en tiempo polinomial. Por la NP-completitud de SAT, se tiene que todos los problemas en NP son reducibles a SAT en tiempo polinomial. De este modo, se puede reducir cualquier problema a SAT y existe una función PC para SAT. Con esto, se tiene que se puede encontrar un certificado o solución para todos los problemas en NP con funciones computables en tiempo polinomial, por lo que todas las funciones se pueden computar en tiempo polinomial, por lo que P=NP.

- $f : A^* \to A^*$ es una función PCC (por polinomial con contra certificado) si existe un lenguaje L en $coNP$, con P y M el polinomio y el lenguaje sobre $A \cup \{\#\}$ que atestiguan la pertenencia de L en coNP, tal que (1) $|f(w)| \leq P(|w|)$ para todo $w \in A^*$, y (2) para todo $w \in L$ se tiene que $w\#f(w)$ pertenece a M.

Si NP=coNP, entonces verificar positivamente que una palabra pertenece a un lenguaje es equivalente a verificar negativamente lo mismo. Esto es por la definición de NP y coNP.

Ahora, por mi definición de PCC, las funciones son PCC si existe un lenguaje L con P y M descritos que atestiguan la pertenencia de un lenguaje L a $coNP$, y que cumplan las condiciones. Por lo que, si NP=coNP, en particular se tiene también que $\hat{\text{NP}}$=coNP ya que $\hat{\text{NP}}$=NP. Así, las funciones PCC atestiguarían la pertenencia de un lenguaje a $\hat{\text{NP}}$ al atestiguar la pertenencia a coNP, por lo que, como las funciones PC atestiguan la pertenencia a $\hat{\text{NP}}$, las funciones PCC serían exactamente iguales a las PC.

Problema 3

- Hay que mostrar que el fragmento del álgebra relacional que usa π, $\sigma_{i=j}$, $\times$, $\cap$ puede ser expresado con consultas conjuntivas. Para esto, consideremos que cada operador de los a definir, recibe como una especie de hiperparámetro a las relaciones con las que se desea operar, donde, para cada caso, corresponden a relaciones del vocabulario. Así, por ejemplo π se puede considerar como $\pi(R)$ o bien como π_R, y de este mismo modo para los 4 operadores a continuación.

  π_i: Sea x una tupla de $k-1$ variables libres, donde k corresponde a la aridad de la relación R. Para saber si una tupla pertenece a la proyección, donde se entiende proyección como excluir al elemento de subíndice i de la relación R, se puede escribir la siguiente fórmula conjuntiva $\varphi_{\pi,i}$:

  $$\varphi_{\pi,i}(x_1,\ldots,x_{k-1}) = \exists y\ R(x_1,\ldots,x_{i-1},y,x_{i+1},\ldots,x_{k-1})$$

  Si se desea considerar, a diferencia de lo visto en la actividad 9, a la proyección en su forma tradicional, considerando que ahora se recibe una tupla de una variable, y se desea saber si dicha tupla pertenece a la proyección del elemento de índice i sobre la relación R de aridad k, basta con escribir la siguiente fórmula conjuntiva $\varphi^*_{\pi,i}$:

  $$\varphi^*_{\pi,i}(x) = \exists y_1 \ldots y_{k-1}\ R(y_1,\ldots,y_{i-1},x,y_{i+1},\ldots,y_{k-1})$$

  $\sigma_{i=j}$: Sea x una tupla de k variables libres, donde k corresponde a la aridad de la relación R. Para saber si una tupla pertenece a la selección donde x_i es igual a x_j, se puede escribir la siguiente fórmula conjuntiva $\varphi_{\sigma,i=j}$:

  $$\varphi_{\sigma,i=j}(x_1,\ldots,x_k) = (x_i = x_j) \wedge R(x_1,\ldots,x_k)$$

  En caso de no considerar como válido utilizar la igualdad (en algunos libros vi que se en consultas conjuntivas se podía utilizar relaciones y la igualdad, pero en la definición del enunciado no se dice así que prefiero asegurar), se puede forzar la misma sin utilizarla explícitamente, esto es:

  $$\varphi_{\sigma,i=j}(x_1,\ldots,x_k) = R(x_1,\ldots,x_i,\ldots,x_{j-1},x_i,x_{j+1},\ldots,x_k)$$

  $\times$: Sea x una tupla de $k+l$ variables libres, donde k corresponde a la aridad de la relación R_i y l a la aridad de R_j. Para saber si una tupla pertenece al producto cruz entre las relaciones R_i y R_j, se puede escribir la siguiente fórmula conjuntiva $\varphi_\times$:

  $$\varphi_\times(x_1,\ldots,x_k,\ldots,x_{k+l}) = R_i(x_1,\ldots,x_k) \wedge R_j(x_{k+1},\ldots,x_{k+l})$$

  $\cap$: Sea x una tupla de k variables libres, donde k corresponde a la aridad de la relación R_i y R_j. Para saber si una tupla pertenece a la intersección entre las relaciones R_i y R_j, se puede escribir la siguiente fórmula conjuntiva $\varphi_\cap$:

  $$\varphi_\cap(x_1,\ldots,x_k) = R_i(x_1,\ldots,x_k) \wedge R_j(x_1,\ldots,x_k)$$

Ahora, para poder expresar la unión, es necesario agregar el conectivo $\vee$ a las consultas conjuntivas, sin embargo, parece un poco contraintuitivo ya que dejaría de tener sentido el nombre *conjuntivas*. Si se desea ser más estricto con la nomenclatura, se podría agregar la negación, que sabemos hace que el conjunto de operadores sea funcionalmente completo, con esto, por De Morgan, se podría simular el mismo comportamiento de agregar la disyunción a las consultas conjuntivas. Para mostrar el caso,

$$\varphi_{\cup}(x_1, \ldots, x_k) = R_i(x_1, \ldots, x_k) \vee R_j(x_1, \ldots, x_k)$$

o bien, alternativamente sin utilizar el conectivo $\vee$ y utilizando la negación

$$= \neg(\neg R_i(x_1, \ldots, x_k) \wedge \neg R_j(x_1, \ldots, x_k))$$

- Para mostrar que CCEVAL es NP-completo, en primer lugar mostraré una reducción de SAT a CCEVAL, con esto mostraré que el problema es NP-hard, y luego mostraré, construyendo una máquina de Turing no determinista que resuelve el problema en tiempo polinomial, que CCEVAL pertenece a NP. Con ambas se muestra que CCEVAL es NP-completo.

  En primer lugar, la reducción, en vez de hacerla de SAT la haré desde 3SAT, que es el problema de satisfacibilidad donde las fórmulas están en 3CNF. Para la reducción consideremos el siguiente mapeo:

  Sea ψ una fórmula en 3CNF, tomemos todas las clausulas de ψ y para cada clausula C_i definamos una relación R_i de aridad 3 tal que una tupla (x_1^i, x_2^i, x_3^i) pertenece a R_i si y solo si C_i evalúa a verdadero para dicho mapeo de sus literales a la tupla. Luego, agreguemos al vocabulario las constantes T y F, para que se interpreten como 0 o 1 (o verdadero o falso), según corresponda. Ahora consideremos la conjunción de todas esas relaciones R_i como la consulta conjuntiva, donde para este caso, se determinan cuantificadores existenciales para cada una de las variables de cada R_i.

  De esta manera, se tiene para el caso de una clausula, que su mapeo sería a $\exists x_1 \exists x_2 \exists x_3 R(x_1, x_2, x_3)$ por ejemplo. Así, tenemos tanto a la estructura, que consiste de las clausulas como relaciones R_i y de las constantes agregadas; tenemos la consulta conjuntiva φ; y solo resta la asignación, que justamente está estrechamente relacionada con los cuantificadores existenciales, donde cada uno mapea un elemento de la asignación con una variable para las relaciones que representan clausulas. Ahora se puede proceder mapeando dicha asignación a la que hace verdadera o falsa a la fórmula original ψ en 3CNF.

  Con lo anterior se tiene una forma de transformar cualquier problema en SAT (ya que cualquier problema se puede expresar en 3CNF) como un problema en CCEVAL, y esta transformación es polinomial ya que todas las operaciones descritas lo son, pues son simples mapeos 1 a 1 de elementos. Así, como todo problema en 3SAT puede ser reducido a un problema CCEVAL en tiempo polinomial, entonces CCEVAL es NP-hard.

  Ahora corresponde probar que CCEVAL es NP. Para esto basándome en la definición de NP que utiliza máquinas de Turing, basta construir una máquina de Turing no determinista que resuelva el problema en tiempo polinomial para demostrar la pertenencia a NP. Con esto, se puede construir dicha máquina del siguiente modo. En cada estado la máquina elige un valor para la asignación de cada x_i, esto es, para cada variable cuantificada existencialmente, el no determinismo de la máquina se aprovecha para generar todos las posibles asignaciones, y por otro lado, esta maquina verifica si esa asignación hace verdadera a la formula, esto es, si satisface o no a la consulta. Esa verificación toma tiempo polinomial ya que dada una asignación, verificar el valor de verdad es polinomial, por lo tanto la máquina de Turing existe y es como se menciona, capaz de, generando una asignación, verificar si es solución para el problema.

  Con lo anterior se tiene que el prroblema está en NP, por ende, sumado a que es NP-hard, se tiene que CCEVAL es NP-completo.

- Para mostrar que el problema 1SI1NO es NP-hard, construiré una reducción polinomial desde 3SAT a 1SI1NO. Consideremos una construcción similar a la que se usó para la demostración de CCEVAL.

  Esto es, sea ψ una fórmula en 3CNF, definamos una estructura $\mathfrak{A}_1$ donde para cada clausula C_i de la formula ψ, se construye una relación R_i tal que una tupla pertenece a ella si y solo si C_i evalua

a verdadero cuando sus literales son asignados a los valores de la tupla. Por otro lado, nuevamente añadamos el 1 y el 0 como elementos del dominio.

Ahora, definamos una estructura $\mathfrak{A}_2$ de modo análogo, pero con la salvedad de que para cada clausula C_i de la formula ψ, se construye una relación R_i tal que una tupla pertenece a ella si y solo si C_i evalua a falso cuando sus literales son asignados a los valores de la tupla.

De este modo, sea la consulta conjuntiva φ construida de la misma manera que para CCEVAL, esto es, utilizando cuantificadores existenciales para cada variable (que representa un literal) y la conjunción de todas las relaciones que representan clausulas, notar que no tiene variables libres puesto que están todas cuantificadas existencialmente.

Ahora, se puede ver que la formula original en 3CNF solo se satisface en la estructura $\mathfrak{A}_1$ ya que se construyó de tal manera, y no se satisface en la segunda ya que es de cierto modo la negación de la primera. El problema de 3-satisfacibilidad se reduce a este problema polinomialmente, ya que todos los mapeos son 1 a 1 y se pueden realizar en tiempo polinomial ya que solo implican transformar de clausulas a relaciones y de solo literales a variables y cuantificadores, en una cantidad acotada polinomialmente por los tamaños de dichos elementos.

Luego, como se puede reducir de 3SAT a 1SI1NO, entonces el problema es NP-hard.

Finalmente, para ver que es coNP-hard, se puede observar que el complemento del problema es exactamente el mismo, solo que con las estructuras intercambiadas, de este modo, preguntarse si $C(\mathfrak{A}_1, \mathfrak{A}_2, \varphi)$ pertenece al complemento del problema, equivale a preguntarse si $C(\mathfrak{A}_2, \mathfrak{A}_1, \varphi)$ pertenece al problema original. De este modo, es evidente que, al ser el mismo problema, solamente intercambiando estructuras, el problema es coNP-hard, y la demostración es análoga a la anterior.

* Como nota adicional, no está demostrado que NP=coNP o que NP≠coNP, sin embargo, si se asume que NP≠coNP, entonces si un problema es NP-hard y coNP-hard, como 1SI1NO este no puede ser NP-completo, pues sería también en coNP-completo, y por ende se llegaría a una contradicción con lo asumido.