IIC2213 — Lógica para Ciencia de la Computación

Notas Semana 6.5

1. Ejercicios pendientes semana 6

Teorema (Monotonía 2). Sean Σ_1 y Σ_2 dos conjuntos de fórmulas en $L(P)$. Entonces $\mathit{modelos}(\Sigma_1 \cup \Sigma_2) \subseteq \mathit{modelos}(\Sigma_1)$.

Ejercicio. Demuestra este teorema.

Demostración. Sean Σ_1 y Σ_2, y consideremos un conjunto S_τ en $\mathit{modelos}(\Sigma_1 \cup \Sigma_2)$. Tenemos que demostrar que $S_\tau \in \mathit{modelos}(\Sigma_1)$. Para hacer eso, tenemos que demostrar que $\tau \models \varphi$ para cada φ en Σ_1.

Sea pues φ una fórmula arbitraria de Σ_1. Como S_τ está en $\mathit{modelos}(\Sigma_1 \cup \Sigma_2)$, por definicion de *modelos* debe ser que τ haga verdad a todas las formulas en $\Sigma_1 \cup \Sigma_2$. Como φ está en Σ_1, también está en $\Sigma_1 \cup \Sigma_2$, y por tanto $\tau \models \varphi$.

Observación. Demuestra lo siguiente:

- $\Sigma \models \varphi$ si y solo si la fórmula $(\bigwedge \Sigma) \rightarrow \varphi$ es una tautología, y por tanto $\neg\varphi \wedge (\bigwedge \Sigma)$ es una contradicción.
- $\mathit{modelos}(\Sigma) \cap \mathit{modelos}(\varphi) = \emptyset$ si y solo si $\Sigma \cup \{\varphi\}$ es una contradicción.

Demostración. Solo escribimos la primera parte. Si $\Sigma \models \varphi$, entonces toda valuación que hace verdad a todas las fórmulas de Σ hace verdad a φ. Demostremos ahora que $(\bigwedge \Sigma) \rightarrow \varphi$ es tautología. Para eso, tenemos que mostrar que todas las valuaciones la satisfacen. Sea pues τ una valuación arbitraria. Si τ no hace verdad a $(\bigwedge \Sigma)$ entonces el lado derecho de $(\bigwedge \Sigma) \rightarrow \varphi$ es falso, y por tanto $\tau \models (\bigwedge \Sigma) \rightarrow \varphi$. Si τ hace verdad a $(\bigwedge \Sigma)$, entonces por la hipótesis de la primera linea tenemos que τ hace verdad a ψ, y luego τ hace verdad a ambos lados de la implicancia, por lo que $\tau \models (\bigwedge \Sigma) \rightarrow \varphi$.

Para el otro lado, asumamos que $(\bigwedge \Sigma) \rightarrow \varphi$ es una tautología. Luego todas las valuaciones la satisfacen, y en particular aquellas que hacen verdad al lado isquierdo. Pero eso significa que si una valuación τ hace verdad a $(\bigwedge \Sigma)$, tiene que hacer verdad a φ (de lo contrario, esa valuación no haría verdad a $(\bigwedge \Sigma) \rightarrow \varphi$). Con esto demostramos que $\Sigma \models \varphi$

2. Solución Tarea 3

Parte a). Para esta parte vamos a asumir que nuestra máquina usa dos símbolos adicionales, X y U. El símbolo X va a ser usado para borrar celdas del input, pero sin utilizar el blanco, que es necesario para saber cuando se acaba el string. El símbolo U va a ser usado para marcar los unos que nos interesan. Además, vamos a usar un segundo símbolo (que en realidad podría ser otro #, pero mejor así por claridad) $\downarrow$ para marcar donde vamos en el grafo.

El algoritmo consiste en primero parsear adecuadamente el input, luego reemplazar el primer # por un $\downarrow$ y luego aplicar, iterativamente, la siguiente función.

- Leer el primer símbolo del output, reemplazar por B. El símbolo me señala si el vértice actual participa o no en el conjunto oscuro.
- Si el primer símbolo es un cero (el vértice no participa), entonces:
  - avanzo hasta el primer $\downarrow$, lo reemplazo por X. reemplazando por X todos los 0s y 1s, hasta que llego a otro gato. Ahí, lo reemplazo por $\downarrow$ y paso a otro estado para continuar ejecutando. *Con esto estamos borrando todo el arreglo de nodos adyacentes del nodo actual, lo que está bien, por que el nodo actual no participa del conjunto oscuro por lo que no tengo que chequear nada. El $\downarrow$ avanza un # para dejar señalado donde empieza el arreglo del próximo nodo*
  - De ahí en adelante, voy a marcar por una X cada primer símbolo en $\{0, 1\}$ después de un # (podrían haber algunas X o Us antes). *Esto corresponde en borrar la información del nodo actual en las adyacencias de cada uno de los otros nodos. Nuevamente, eso está bien, no participa del conjunto oscuro así que no me interesa.*
- Si el primer símbolo es un uno, entonces
  - avanzo hasta el primer $\downarrow$, lo reemplazo por X. Luego busco el primer símbolo 0 o 1, si es un 1 lo reemplazo por U, si es un 0 lo reemplazo por X. *Ahora procesé solo el nodo actual en el arreglo de adyacencia del nodo actual. Si hay un self loop voy a poner una U, si no una X*
  - continuo hasta el próximo #, el que reemplazo por $\downarrow$. *Nuevamente, preparo para después.*
  - Ahora voy a marcar con una X cada primer símbolo en $\{0, 1\}$ después de un #, **si el símbolo es un** 0, y lo marco con U si es un 1.
- Termino cuando hay un blanco B justo después del $\downarrow$.

Una vez que termine, tengo que chequear si el input tiene algún U. Si no tiene, el conjunto es oscuro. Si tiene, el conjunto no es oscuro

Parte b). Acá lo que hacemos es:

- Asegurarse que el input está bien compuesto (parsear).
- Asegurarse que $k \leq n$ (un grafo no puede tener un conjunto oscuro más grande que sus nodos)
- Hacer una máquina no determinista que lea el string 1^k y genere, no deterministicamente, todos los posibles strings de tamaño n y que tienen k unos.
- esos strings deben quedar pegados a $\#C(G)$, de forma de poder llamar a la parte a).

Para hacer la máquina no determinista:

- Escribir 01^n entremedio del 1^k y del $\#C(G)$
- Aplicar iterativamente (partiendo al inicio del 1^k) y en un estado q^*:
  - Reemplazar un 1 por B
  - Avanzar hasta el primer 0. Después, no-determínisticamente cambiar un cero por un 1. Es decir, si estamos en un estado q, hacer $\delta(q, 0) = \{(q, 0, \rightarrow), (q_1, 1, \leftarrow)\}$, hacer que q_1 se devuelva al blanco y de ahí volver al q^*. Para asegurar que haya que cambiar un 1 si o si, hacer $\delta(q, \#) = \emptyset$, de esta forma todas las hojas donde la máquina pasa de largo sin escribir algún 1 no llegan a nada, por que se detiene en un estado final.
  - Terminamos cuando, en el estado q^*, leemos un 0. Ahí escribimos un blanco y pasamos al estado inicial de la parte a)

3. Problema de Modelación

Este problema lo trabajamos en una tarea del año pasado, les puede servir como ejemplo para ver cómo se modelan procesos usando lógica proposicional.

3.1. Enunciado

El módulo de conflictos de un sistema operativo funciona de la siguiente forma. Se necesitan correr procesos $p_1, \ldots, p_n$, en T milisegundos (ms); por simplicidad asumimos que cada proceso toma exactamente 1 ms en completarse.

El proceso de planificar los procesos consiste en asignar, a cada proceso, un índice de tiempo entre 0 y $T - 1$, que corresponde al tiempo donde se ejecutará el proceso, de forma que alcancen a completarse todos. El problema es que los procesos compiten por recursos de memoria de la CPU. La memoria cuenta con 8 recursos aritméticos distintos: $r_1, \ldots, r_8$, y cada uno de los procesos $p_1, \ldots, p_n$ usa algunos de estos recursos. Esta información viene dada por una función f que asigna a cada proceso un conjunto $f(p_i) \subseteq \{r_1, \ldots, r_8\}$ de recursos.

El problema a resolver es: ¿Hay alguna forma de poder planificar los procesos requeridos en T milisegundos? Tu meta para esta pregunta es mostrar como podemos resolver este problema con un SAT solver: Dados procesos $p_1, \ldots, p_n$, una cantidad T de milisegundos y una función f que asigna a cada proceso un conjunto $f(p_i) \subseteq \{r_1, \ldots, r_8\}$ de recursos, decidir si acaso existe una forma de asignar a cada proceso un índice de tiempo entre 0 y $T-1$, de manera que, para cada par de procesos p y p' asignados al mismo tiempo t ($0 \leq t \leq T-1$), se tiene que $f(p) \cap f(p') = \emptyset$: los procesos no usan ningún recurso en común.

Específicamente, debes mostrar como construir, para inputs $p_1, \ldots, p_n$, T y f descritos arriba, una fórmula φ tal que φ es satisfacible si y solo si existe una forma de planificar los procesos como se describió más arriba.

3.2. Solución

Dados:
- $P = \{p_1, ..., p_n\}$ el conjunto de procesos.
- T milisegundos.
- $R = \{r_1, ..., r_8\}$ recursos.
-$f(p_i) \subseteq \{r_1, ..., r_8\}$

Por construir, fórmula φ satisfacible si y solo si existe una forma de planificar estos procesos.

Para i en $\{1, \ldots, n\}$, t en $\{1, \ldots, T\}$, j en $\{1, \ldots, 8\}$, definimos proposiciones a_{it}, b_{itj} y s_{ij}. La intuición es que una valuación de esas variables representa un intento de planificación, de forma que las variables asignadas a 1 tienen la siguiente intuición:

- $a_{it} = 1$ si el proceso p_i es asignado al tiempo $t \in \{0, ..., T-1\}$, $a_{it} = 0$ en otro caso.
- $b_{itj} = 1$ si el proceso p_i es usa el recurso j en el tiempo t. $b_{itj} = 0$ en otro caso.
- $s_{ij} = 1$ si $r_j \in f(p_i)$, $s_{ij} = 0$ en otro caso.

Sean:

$\varphi_0 = \bigwedge_{r_j \in f(p_i)} s_{ij} \wedge \bigwedge_{r_j \notin f(p_i)} \neg s_{ij}$ (La valuación a las variables s se comporta de acuerdo a los parámetros que nos dieron)

$\varphi_1 = \bigwedge_{i=1}^{n} \bigvee_{t=0}^{T-1} a_{it}$ (todos los procesos son asignados)

$\varphi_2 = \bigwedge_{i=1}^{n} (a_{it} \rightarrow \bigwedge_{t'=0, t \neq t'}^{T-1} \neg a_{it'})$ (procesos son asignados a un solo tiempo)

$\varphi_3 = \bigwedge_{t=0}^{T-1} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{8} (s_{ij} \wedge a_{it}) \rightarrow b_{itr}$ (procesos usan los recursos que les corresponden)

$\varphi_4 = \bigwedge_{t=0}^{T-1} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{8} (b_{itj} \rightarrow \bigwedge_{i'=1, i' \neq i}^{n} \neg b_{i'tj})$ (los procesos no comparten recursos)

Construimos $\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$, por demostrar que φ es satisfacible $\leftrightarrow$ existe una forma de planificar estos procesos.

($\rightarrow$), por contradicción suponemos φ satisfacible y no se pueden planificar estos procesos de forma que si p y p' son asignados al mismo tiempo entonces $f(p) \cap f(p') = \emptyset$.

De la definición anterior y sin pérdida de generalidad, podemos suponer que p_1 y p_2 son asignados a $t = 0$, pero $f(p) \cap f(p') \neq \emptyset$ (necesariamente se tiene que cumplir para algún tiempo y par de procesos). Nuevamente, sin pérdida de generalidad (pues al menos deben compartir un recurso), supongamos que $r_1 \in f(p)$ y $r_1 \in f(p')$.

Como φ es satisfacible, por construcción sabemos que $a_{10} = 1, a_{20} = 0$ (por φ_1, φ_2), $b_{101} = 1$ y $b_{201} = 1$ (por φ_3). Pero si $b_{201} = 1$ entonces, por φ_4, necesariamente $b_{101} = 0 \rightarrow\leftarrow$. (1)

($\leftarrow$), por demostración directa, suponemos que existe una forma de asignar a todos los procesos de manera que si p y p' son asignados al mismo tiempo entonces $f(p) \cap f(p') = \emptyset$.

φ_0 es satisfacible asignando $s_{ij} = 1$ si $r_j \in f(p_i)$, $s_{ij} = 0$ en otro caso. Como asignamos cada proceso a solo un tiempo, φ_1 y φ_2 se cumplen asignando a_{it} para cada proceso y tiempo trivialmente. Además, podemos asignar $b_{itj} = 1$ para cada $a_{it} = 1$ tal que $r_j \in f(p_i)$ y $b_{itj} = 0$ en los otros casos. De esta manera, si repetimos lo anterior para todo tiempo y recurso, se cumple φ_3. También sabemos que se cumple φ_4 dada la suposición inicial, es decir, si dos procesos son tales que $f(p) \cap f(p') \neq \emptyset$, entonces no estarán asignados al mismo tiempo. Entonces tenemos que φ es satisfacible por esa asignación (2)

Finalmente, por (1) y (2) se tiene que φ es satisfacible $\leftrightarrow$ existe una forma de planificar estos procesos. Por lo tanto, se puede resolver el problema planteado con un SAT solver. ■