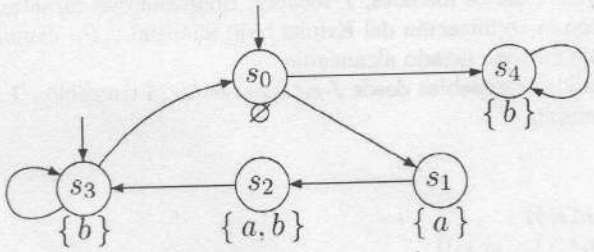
Ingeniería del Software 2

Segundo Parcial

Ejercicio 1

Dada la estructura de kripke K y la fórmula $\varphi = (a \,\mathtt{AU}\, b) \vee (\mathtt{EX}\,\mathtt{AG}\, b)$, determinar el conjunto de estados donde φ es válida y también si $K \models \varphi$.



Ejercicio 2

Para la estructura de kripke del ejercicio anterior desarrolle, utilizando el procedimiento de model checking de estados explícitos, el cómputo de estados que satisfacen

$$\mathtt{EF}\,\mathtt{AG}\, b$$

Ejercicio 3

Considere un sistema de ascensor para 3 pisos numerados del 0 al 2. En cada piso hay una puerta, un botón para llamar al ascensor y una luz que indica si el ascensor fue llamado desde ese piso. Dentro del ascensor hay N botones (uno por piso) y N luces indicando qué pisos el ascensor visitará.

a) Determine que proposiciones utilizaría en un Kripke para desribir este sistema. Explique que denota cada una.
b) Escriba en CTL las siguientes propiedades: *Las puertas son seguras. Es decir, las puertas nunca están abiertas en un piso si el ascensor no se encuentra en el piso.*
c) Las luces (internas y externas) indican correctamente pedidos pendientes. Esto incluye tanto consistencia entre luces internas y externas como el período en que debe estar prendido una luz (entre que se presiona el boton y el ascensor llega).
d) Escriba en CTL las siguientes propiedad: *El ascensor sólo visita pisos que fueron solicitados y no se mueve si no hay pedidos.*

Ejercicio 4

Considere una estructura de kripke donde los estados están caracterizados por el valor de dos proposiciones (a y b) y sus transiciones por la fórmula:

$$(a \wedge \neg b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge \neg b')$$

a) Arme el ROBDD para la fórmula
b) Dibuje la estructura de Kripke resultante
c) Computar $AX(a \wedge \neg b)$ usando la codificación precedente (mostrar los pasos del cómputo).

Ejercicio 5

Utilizar los resultados del paper de bmc para justificar la corrección y completitud del siguiente pseudo código para verificar que una fórmula Booleana P vale en todo estado alcanzable de una estructura de Kripke codificada con variables Booleanas, un predicado de inicio I y relación de transición T. Se entiende que hay s0 ... sn , etc corresponden a las variables de los bit vectors correspondientes a los estados de la traza (como en el paper).

Input: I fórmula Boolena que caracteriza los estados iniciales, T fórmula Booleana que caracteriza la transición de estados (i.e., I y T son la codificación del Kripke bajo análisis) y P, fórmula Booleana que se quiere saber si vale en todo estado alcanzable

Output: TRUE y vale P en todos los estados alcanzables desde I-estados según la transición T, ó FALSE y hay una traza contrajemplo

1 $i := 0$
2 $iPaths := I(s_0)$
3 $CounterExample := IsSAT?(iPaths \wedge \neg P(s_0))$
4 $IsExtensible := IsSAT?(iPaths \wedge T(s_0, s_1) \wedge (s_1 \neq s_0))$
5 **while** $IsExtensible \wedge \neg CounterExample$ **do**
6 $\quad i++$
7 $\quad iPaths := iPaths \wedge T(s_{i-1}, s_i)$
8 $\quad CounterExample := IsSAT?(iPaths \wedge \neg P(s_i))$
9 $\quad IsExtensible := IsSAT?(iPaths \wedge T(s_i, s_{i+1}) \wedge (\bigwedge_{d=0,..,i} : (s_{i+1} \neq s_d)))$
10
11 **return** $\neg CounterExample$

---

1) Queremos hallar los estados donde vale $\varphi = (a \, A U \, b) \vee (EX \, AG \, b)$ para el Kripke dado, es decir, $[[\varphi]]_K$.

Comenzamos viendo para eso $[[a]]_K = \{s_1, s_2\}$ y $[[b]]_K = \{s_2, s_3, s_4\}$.

Luego, para $[[a \, AU \, b]]_K$ vemos que $a \, AU \, b = AF \, b \wedge (a \, AW \, b) = \neg(EG \neg b) \wedge \neg(\neg b \, EU \neg(a \vee b))$

Por ende, viendo que $[[a \vee b]]_K = [[a]]_K \cup [[b]]_K = \{s_1, s_2, s_3, s_4\}$, $[[\neg(a \vee b)]]_K = S \setminus \{s_1, s_2, s_3, s_4\} = \{s_0\}$

Siendo $[[\neg b]]_K = S \setminus \{s_2, s_3, s_4\} = \{s_0, s_1\}$ vemos los estados a los que llegamos desde $\{s_0\}$ dando pasos hacia atrás y tenemos $\{s_3\}$, por lo que $[[\neg b \, EU \neg(a \vee b)]]_K = \{s_0\}$ y su negación lleva a $\{s_1, s_2, s_3, s_4\}$

Ahora, para $EG \neg b$ vemos que podemos para $[[\neg b]]_K = \{s_0, s_1\}$ no encontrar una componente fuertemente conexa, por lo que $[[EG \neg b]]_K = \{\}$ y entonces $[[\neg EG \neg b]]_K = \{s_0, s_1, s_2, s_3, s_4\}$

Tomando la conjunción, llegamos a que $[[a \, AU \, b]]_K = \{s_1, s_2, s_3, s_4\} \cap \{s_0, s_1, s_2, s_3, s_4\} = \{s_1, s_2, s_3, s_4\}$ ✓

Por otra parte, $EX \, AG \, b = EX \neg EF \neg b$ donde desde $[[\neg b]]_K = \{s_0, s_1\}$ buscamos los estados que llevan a estos dando pasos hacia atrás y llegamos a $\{s_0, s_1, s_2, s_3\}$. Por ende, su negación da $\{s_4\} = [[AG \, b]]_K$

Ahora buscamos $[[EX \, AG \, b]]_K$ al dar un paso hacia atrás de $\{s_4\}$ y llegamos a $\{s_0, s_4\}$ ✓

Finalmente, de la disyunción sale que $[[\varphi]]_K = \{s_1, s_2, s_3, s_4\} \cup \{s_4, s_4\} = \{s_0, s_1, s_2, s_3, s_4\} = S$ ✓ Particularmente vale φ en los estados iniciales de K ($\{s_0, s_3\}$) por lo que $K \models \varphi$ ✓

1 ✓ 2 ✓ 3 ✓- 4 ✓ 5 ✓ 9 75

2) Queremos buscar a través de model checking explícito los estados en que vale $\varphi = EF\,AG\,b = EF(\neg EF \neg b)$

Para ello aplicamos una estrategia bottom-up que parte de los conjuntos característicos de las subfórmulas para caracterizar las que compone.

* $[[b]]_K = \{s_2, s_3, s_4\} \Rightarrow [[\neg b]]_K = S \setminus \{s_2, s_3, s_4\} = \{s_0, s_1\}$

* $[[EF(\neg b)]]_K$ lo calculamos partiendo de $\{s_0, s_1\}$ y dando pasos hacia atrás, lo que nos lleva a $\{s_0, s_1, s_2, s_3\}$.

* Además como buscamos $[[\neg EF(\neg b)]]_K$ tenemos $S \setminus \{s_0, s_1, s_2, s_3\} = \{s_4\} = [[AG\,b]]_K$

* Finalmente buscamos $[[EF(AG\,b)]]_K$ lo cual obtenemos dando pasos hacia atrás desde $\{s_4\}$ y como al primer paso llegamos a $\{s_0, s_4\}$ y de ahí podemos seguir dando pasos hacia atrás hasta llegar a $\{s_0, s_1, s_2, s_3\}$ tenemos que:

$[[EF\,AG\,b]]_K = \underline{\{s_0, s_1, s_2, s_3, s_4\}}$

3) a) Se sabe que tenemos 3 pisos enumerados de 0 a 2, para cada piso hay un botón y una luz externa que indica si el ascensor se llamó desde ese piso. Dentro del ascensor hay un botón por piso y una luz por piso que indica el piso a visitar por el ascensor.

En base a lo anterior definimos las proposiciones:

- boton_piso_i = "El botón del piso i esta siendo presionado"
- luz_piso_i = "La luz del piso i está prendida"
- ascensor_boton_i = "El botón del piso i del ascensor está siendo presionado"
- ascensor_luz_i = "La luz del piso i del ascensor está prendida"
- puerta_abierta = "La puerta del ascensor está abierta"
- en_movimiento = "El ascensor está en movimiento"
- en_i = "El ascensor se encuentra en el piso i"

Asumo que tras presionar un botón, tanto desde el ascensor como del piso correspondiente, se prende la luz correspondiente a ese piso.

Además notamos que si un ascensor está en el piso i no puede estar en el resto, es decir, $en\_i \leftrightarrow \bigvee_{l \neq i} en\_l$ con $i \in \{0,1,2\}$

También asumo que la luz del piso i se apaga al llegar al piso i, entre que las puertas se abren y se cierran

b) Queremos escribir que las puertas sean seguras en el sentido que se abren cuando el ascensor está visitando un piso (no se está moviendo y está en el piso dado). Luego, eso equivale a formular:

$A(puerta\_abierta \rightarrow ((en\_0 \wedge \neg en\_movimiento) \vee (en\_1 \wedge \neg en\_movimiento) \vee (en\_2 \wedge \neg en\_movimiento)))$

c) Para que haya consistencia en los pedidos pendientes notamos que hace falta que las luces tanto del ascensor como de cada piso estén prendidas correspondientemente, es decir, $A((luz\_piso\_0 \leftrightarrow ascensor\_luz\_0) \wedge (luz\_piso\_1 \leftrightarrow ascensor\_piso\_1) \wedge (luz\_piso\_2 \leftrightarrow ascensor\_piso\_2)) = \varphi_1$

También hace falta que la luz se mantenga prendida desde que se presionó el botón del piso hasta que el ascensor visite el piso, lo cual lleva a:

$A((boton\_piso\_i \vee ascensor\_boton\_i) \rightarrow (luz\_piso\_i) A\mathcal{U} (\neg en\_movimiento \wedge en\_i \wedge puerta\_abierta)) = \varphi_{2i}$ para $i \in \{0,\dots,2\} \Rightarrow \varphi_2 = \bigwedge_{i=0}^{2} \varphi_{2i}$

Finalmente, tenemos $\varphi_1 \wedge \varphi_2 = \varphi_1 \wedge \bigwedge_{i=0}^{2} \varphi_{2i}$

FALTA UNA VUELTA

d) Finalmente, nos falta demostrar que un ascensor visita sólo los pisos solicitados y no se mueve de no haberlos. Para ello vemos primero la segunda parte de la propiedad como $A(en\_movimiento \rightarrow (luz\_piso\_0 \vee luz\_piso\_1 \vee luz\_piso\_2)) = \varphi_3$

Por otra parte, para que el ascensor visite el piso i debe estar en este piso y no en movimiento. Por ende, tomando el momento en que el ascensor visita el piso i, tenemos:

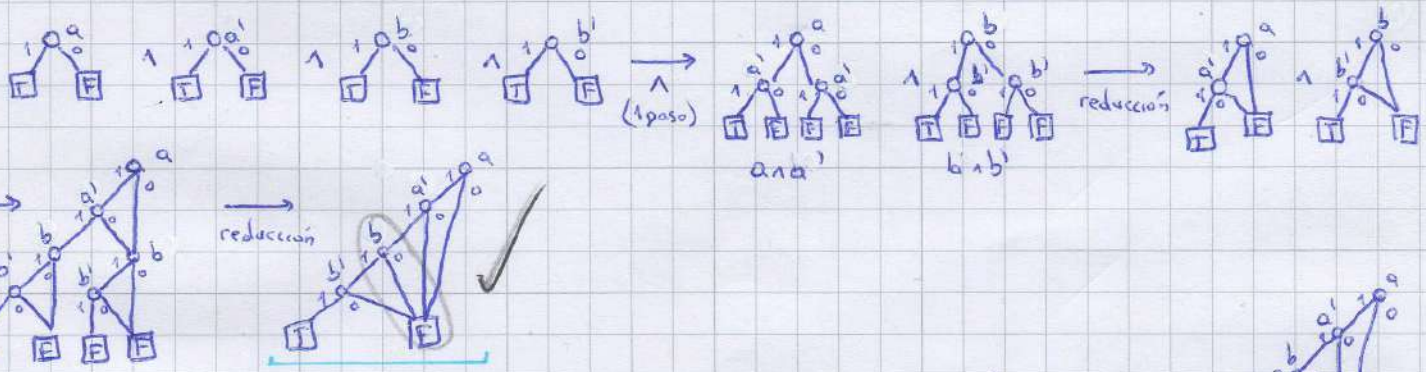
$\varphi_{2_i} = A(\text{en\_}i \wedge \neg \text{en\_movimiento} \wedge \text{puerta\_abierta} \leftrightarrow \text{luz\_piso\_}i)$ para $i \in \{0, \dots, 2\} \Rightarrow \varphi_2 = \bigwedge_{i=0}^{2} \varphi_{2_i}$

Finalmente tenemos $\varphi_1 \wedge \varphi_2 = \varphi_1 \wedge \bigwedge_{i=0}^{2} \varphi_{2_i}$

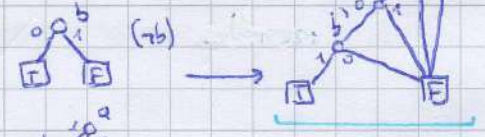
4) a) Para armar el ROBDD de la fórmula tomaremos la relación de orden: $a < a' < b < b'$. Luego, denotando a

$f = \underbrace{(a \wedge \neg b \wedge a' \wedge b')}_{f_1} \vee \underbrace{(a \wedge b \wedge a' \wedge b')}_{f_2} \vee \underbrace{(a \wedge b \wedge a' \wedge \neg b')}_{f_3} = f_1 \vee f_2 \vee f_3$ podemos formar el ROBDD de f en base a los de sus sub fórmulas:

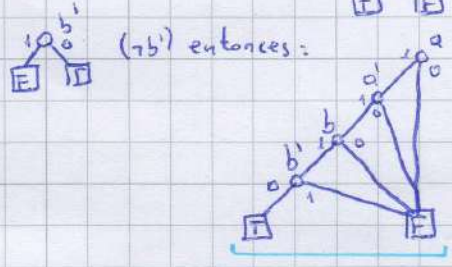
f_2:



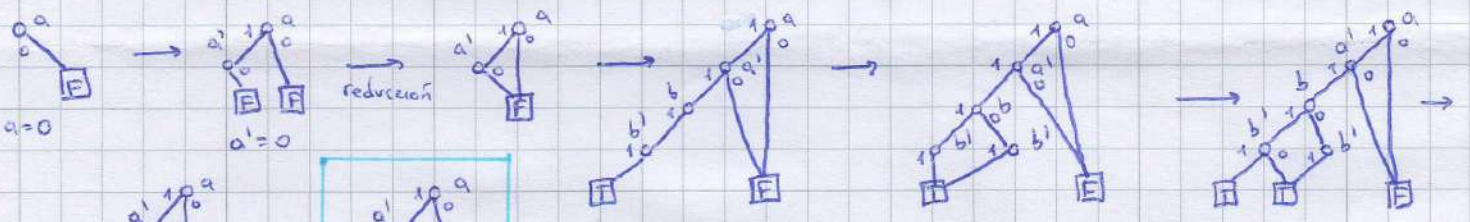
f_1: Siguiendo una lógica similar a f_2 pero reemplazando b por $\neg b$: ($\neg b$)

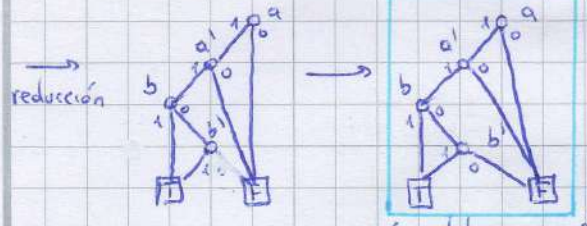


f_3: De forma similar, reemplazo b' por $\neg b'$ en f_2: ($\neg b'$) entonces:

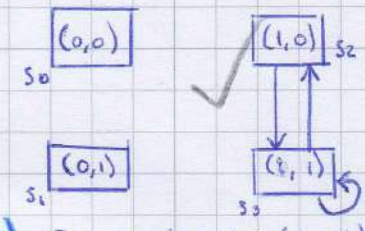


Finalmente, hacemos la disyunción de las 3 subfórmulas en un nuevo ROBDD:





b) Considerando que los estados se notan a través de las proposiciones a y b, dibujamos para $S = \{(0,0), (0,1), (1,0), (1,1)\}$



donde tomamos que los estados se denotan con (a,b) y las transiciones $T(S_{(1,0)}, S_{(1,1)})$, $T(S_{(1,1)}, S_{(1,1)})$ y $T(S_{(1,1)}, S_{(1,0)})$ se dan por f_1, f_2 y f_3 respectivamente

c) Para computar $AX(a \wedge \neg b)$ notamos que esto es equivalente a probar $\neg EX(\neg(a \wedge \neg b))$ que nos permite aplicar la fórmula $EX(p) = \exists V'. \underbrace{F}_{R} \wedge p[V'/V]$ correspondiente a la verificación de Symbolic Model Checking. Así tenemos:

$EX(\neg(a \wedge \neg b)) = \exists V'. F \wedge \neg(a \wedge \neg b)[V'/V] = \exists V'. F \wedge \neg(a' \wedge \neg b') = \exists V'. F \wedge (\neg a' \vee b') = \exists V'. f_1 \vee f_2$ (puesto que $f_3 \wedge (\neg a' \vee b') = false$) $= (f_1 \vee f_2)[(V,V)/(a',b')] \vee (f_1 \vee f_2)[(V,F)/(a',b')] \vee (f_1 \vee f_2)[(F,V)/(a',b')] \vee (f_1 \vee f_2)[(F,F)/(a',b')]$

$= (f_1 \vee f_2)[(V,V)/(a',b')]$ (puesto que tanto f_1 como f_2 tiene la conjunción con a' y b') $= (a \wedge \neg b) \vee (a \wedge b) = a = \{s_2, s_3\}$

Sin embargo, como se pedía $AX(a \wedge \neg b) = \neg EX(\neg(a \wedge \neg b))$ tenemos $[[AX(a \wedge \neg b)]] = S \setminus \{s_2, s_3\} = \{s_0, s_1\}$

5) En el presente ejercicio lo que se pretende es en base a los resultados provistos por el paper de BMC notar que el algoritmo es efectivamente correcto al verificar que una fórmula P vale en todo estado alcanzable y contempla todos los casos en que esto ocurre. Esto lo hace partiendo de las fórmulas booleanas I y T que caracterizan a los estados iniciales y las transiciones, así como la fórmula P en sí. Lo que se desea probar entonces es si el sistema satisface AG_p.

Esto es equivalente a probar si existe alguna instanciación en que valga $EF_{\neg p}$ y de ser así negar su resultado. Para esto, el algoritmo usa un contador i, una variable ¡Paths que va agrupando los caminos alcanzables del algoritmo, otra Is Extensible que a través de la fórmula de transiciones T verifica que por cada paso ¡Paths incluya un nuevo bit vector s_i, e Counter Example, que es el resultado de la fórmula que dados los caminos formados por ¡Paths y la fórmula P verifica si al hacer la conjunción con su negación evaluada en el i-ésimo bit vector se llega a una posible instanciación.

De ser así, el ciclo termina considerando que Is SAT? (...) retornará TRUE, refutando la condición de la guarda y retornando FALSE. Efectivamente, esto sucede Is SAT? (...) retorna TRUE, lo cual se condice con que $\neg P(s_i)$ sea TRUE.

En caso contrario, el ciclo termina si no se cumple Is Extensible, la cual por su definición se da cuando $T(s_i, s_{i+1})$ es FALSE (no hay estado al cual transicionar) o se da la negación de la conjunción siguiente, que establece que la transición se puede dar a un nuevo estado no visitado. En efecto, considerando que la traza es finito al considerarse una cantidad sólo n de bit vectors, a lo sumo el ciclo terminará cuando se llegue a s_n y s_{n+1} no cumpla con la condición del ciclo.

Ahora, lo anterior sólo nos dice que el algoritmo efectivamente termina, pero debemos atender a su resultado: counter Example. Por lo dicho anteriormente, este puede ser TRUE ó FALSE según los resultados de evaluar Is SAT? ($¡Paths \wedge \neg P(s_i)$) en el i-ésimo paso. Aún así, podemos desenrollar la fórmula hasta este paso usando las sucesivas iteraciones de T para llegar a que:

$$\text{Counter Example} = \text{Is SAT?}\left(I(s_0) \wedge \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \wedge \neg P(s_i)\right)$$

De ahí notamos la primera parte de la conjunción como la restricción de la traza $s_0, \dots, s_i$, mientras que la segunda ve si efectivamente en el i-ésimo paso no vale P. Nótese que por Is Extensible $T(s_j, s_{j+1})$ a su vez está restringiendo a que $s_{j+1} \neq s_L$ con $L \in \{0, \dots, j\}$.

Más aún, del paper sabemos que para una fórmula LTL f, estructura de Kripke M, si se satisface $M \models EF$ luego existe $j \in \mathbb{N}$ para el que vale $M \models_n EF$ (y viceversa). En nuestro caso, M viene dado por I y T y la fórmula LTL es $\neg P$ (proposicional) por lo que de valer $M \models_n^{EF\neg P}$ sabemos que $M \models^{EF\neg P}$, es decir $M \not\models AGP$.

La profundidad se respeta gracias a Is Extensible, y M viene conformado y restringido por ¡Paths y Is Extensible. Luego, considerando que $M \not\models_k EF\neg P$ se satisface cuando Counter Example es TRUE en el paso $L \in \{0, \dots, k\}$ entonces estamos viendo todas las profundidades que llegan a un nuevo estado alcanzable cada vez, terminando en que para $k=n$ se vieron todos los estados alcanzables, por lo que hay un contraejemplo según lo que indique Counter Example y el algoritmo satisface lo pedido.