DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Taller Nº 6

Linear Temporal Logic, Automatas de Büchi y Model checking

Ingeniería de Software II
Segundo cuatrimestre de 2020

Integrante	LU	Correo electrónico
Manuel Mena	313/14	manuelmena1993@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1.

1.1.

$[] <> enBase$

Es de liveness ya que no puedo encontrar un contraejemplo finito.

1.2.

$[](bateriaBaja \rightarrow (modoAhorro \cup enBase))$

Es de safety ya que puedo encontrar un contraejemplo finito donde tiene $bateriaBaja$ pero no entra en $modoAhorro$ hasta llegar a $enBase$.

1.3.

$[]\neg(girandoAIzquierda \wedge girandoADerecha)$

Es de safety ya que puedo encontrar un contraejemplo finito donde valga $girandoAIzquierda \wedge girandoADerecha$.

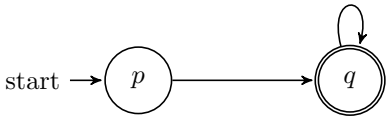
1.4.

$[](detectaPared \rightarrow (girandoIzquierda \cup \neg detectaPared))$

Es de safety ya que puedo encontrar un contraejemplo finito donde $detectaPared$ pero no vale $girandoIzquierda$ por mas que siga valiendo $detectaPared$.

2.

2.1.

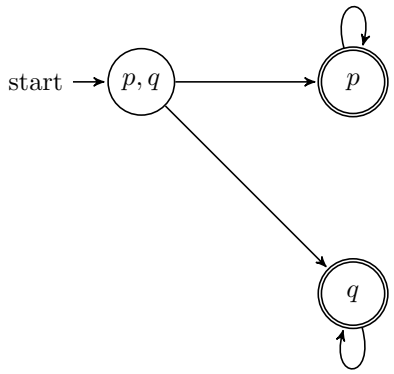


La traza $p \rightarrow q$ no vale para la primer formula pero si para la segunda.

2.2.

$$
\begin{aligned}
\sigma[i] \models []\alpha \quad & \iff \forall j \geq i(\sigma[j] \models \alpha) \\
\sigma[i] \models []p \wedge []q \quad & \iff \forall j \geq i(\sigma[j] \models p) \wedge \forall j \geq i(\sigma[j] \models q) \\
& \iff \forall j \geq i(\sigma[j] \models p) \wedge (\sigma[j] \models q) \\
& \iff \forall j \geq i(\sigma[j] \models p \wedge q) \\
& \iff \sigma[i] \models [](p \wedge q)
\end{aligned}
$$

2.3.

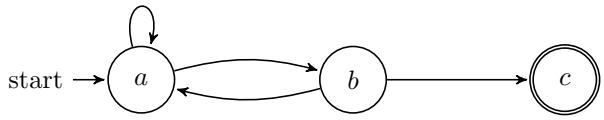


La traza $p, q \longrightarrow p$ vale para la segunda formula pero no para la primera.

2.4.

$$
\begin{aligned}
\sigma[i] \models \alpha U \beta \quad & \iff \exists k \geq i (\sigma[k] \models \beta \wedge \forall j, i \leq j < k (\sigma[k] \models \alpha)) \\
\sigma[i] \models (a \vee b) U b \quad & \iff \exists k \geq i (\sigma[k] \models b \wedge \forall j, i \leq j < k (\sigma[k] \models (a \vee b))) \\
& \iff \exists k \geq i (\sigma[k] \models b \wedge \forall j, i \leq j < k (\sigma[k] \models a \vee \sigma[k] \models b)) \\
& \iff \exists k \geq i (\sigma[k] \models b \wedge \forall j, i \leq j < k (\sigma[k] \models a) \vee \forall j, i \leq j < k (\sigma[k] \models b)) \\
& \iff \exists k \geq i (\sigma[k] \models b \wedge \forall j, i \leq j < k (\sigma[k] \models a)) \\
& \iff \sigma[i] \models a U b
\end{aligned}
$$

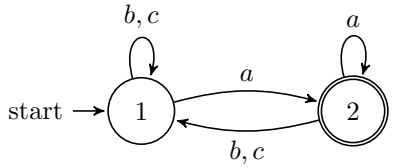
2.5.



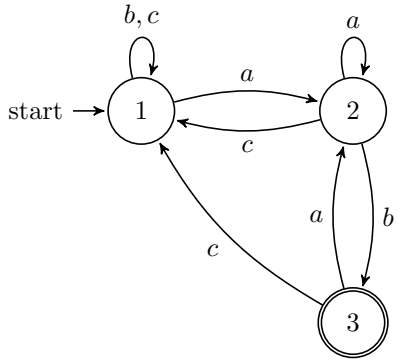
La traza $a \longrightarrow b \longrightarrow a \longrightarrow b \longrightarrow c$ vale para la primer formula pero no para la segunda, ya que en el cuarto estado se satisface bUc pero no se satisface a para todos los estados anteriores.

3.

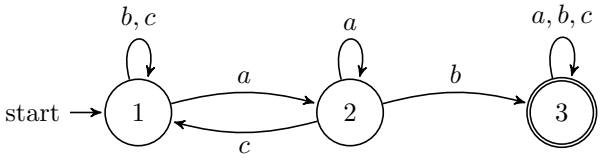
3.1.



3.2.



3.3.



4.

4.1.

$<> ([]p \vee []q)$

Una vez que llega p o q, vale p o q siempre, respectivamente. Debido al loop de init sobre si mismo, se agrega $<>$ indicando que esto va a suceder cuando se cambie de estado.

4.2.

$[](\neg(\neg p||q) - ><> q)$

En todos los estados ocurre que si llega algo distinto de $\neg p$ o q, caemos en el estado 1 en donde se puede ciclar sobre si mismo, como ocurria con init del ejercicio anterior, pero para aceptar la cadena en algun momento debe valer q.

4.3.

$<> c||[](\neg p||q)$

Puede que llegue c o que se entre en el estado 1 y se cicle alguna cantidad de veces pero que en algun momento llegue c, o bien, que valga $(\neg p||q)$ y a partir de ese estado valga por siempre.

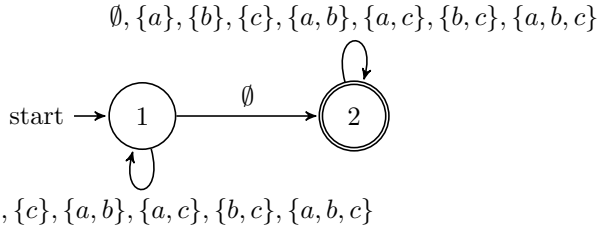
5.

5.1.

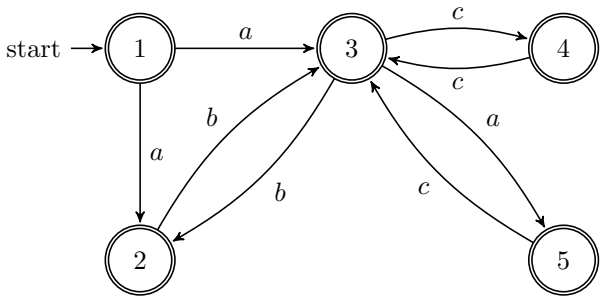
5.1.1.

Paso 1: Niego la propiedad y genero su automata de Büchi

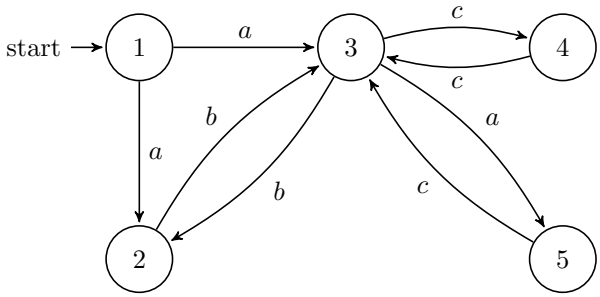
$\neg([](a \vee b \vee c)) =<> (\neg a \wedge \neg b \wedge \neg c)$



Paso 2: Convertir el LTS a un automata de Büchi



Paso 3: Hacemos el producto entre los autómatas



Paso 4: Si el lenguaje de aceptación del producto de los autómatas es vacío, se cumple la propiedad

En este caso lo es ya que no tiene ningún estado que acepte.

5.1.2.

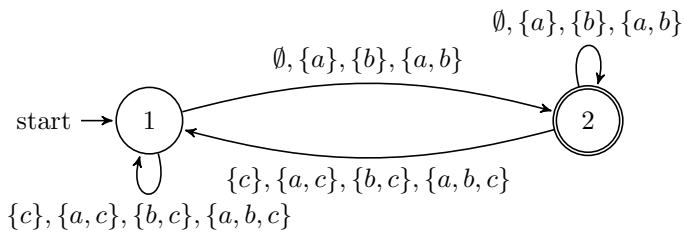
La formula vale porque dice que siempre vale alguno a, b o c lo cual es cierto porque todos los estados transicionan mediante alguno de esos simbolos, y todos los estados aceptan. LTSA nos lo indica con este output:

Formula !PROP1 = (true U (!c (!b !a)))
Analysing...
Depth 3 – States: 5 Transitions: 8 Memory used: 14098K
No deadlocks/errors
Analysed in: 0ms

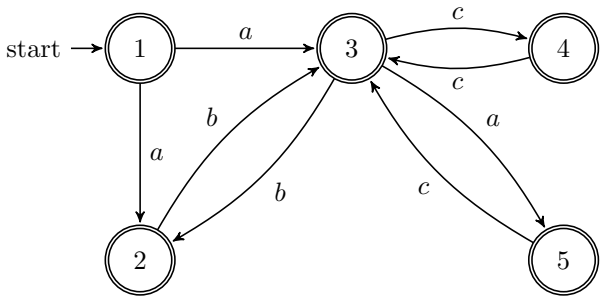
5.2.

Paso 1: Niego la propiedad y genero su automata de Büchi

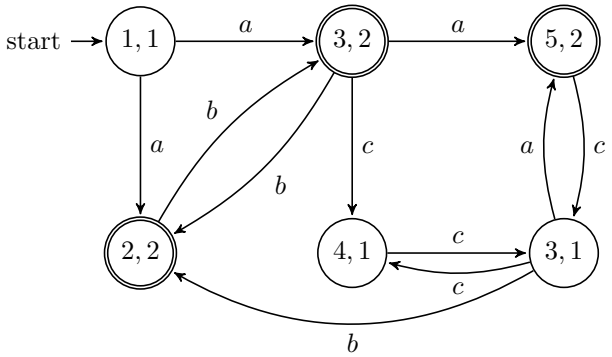
[] <> $\neg c$



Paso 2: Convertir el LTS a un automata de Büchi



Paso 3: Hacemos el producto entre los autómatas



Paso 4: Si el lenguaje de aceptación del producto de los autómatas es vacío, se cumple la propiedad

En este caso, no es vacío. Podemos observar en el producto de los autómatas que existen cadenas de aceptación. Por lo tanto, no se cumple la propiedad

5.2.1.

No vale porque no se puede llegar en ningun momento a que siempre valga c. LTSA nos lo dice con este output:

Formula !PROP2 = (false R (true U !c))
LTL Property Check...
– States: 8 Transitions: 34 Memory used: 15243K
Finding trace to cycle...
Depth 2 – States: 3 Transitions: 11 Memory used: 15689K
Finding trace in cycle...
Depth 1 – States: 1 Transitions: 1 Memory used: 16089K
Violation of LTL property: @PROP2
Trace to terminal set of states:
a a Cycle in terminal set:
c a LTL Property Check in: 2ms

5.3.

La propiedad no vale ya que hay transiciones donde no se cumple a. El output de LTSA es:

Formula !PROP3 = (true U !a)
Analysing...
Depth 2 – States: 2 Transitions: 3 Memory used: 14157K
Trace to property violation in PROP3:
a
b

Analysed in: 1ms

5.4.

La propiedad vale ya que para cuando en los dos estados desde donde se puede ir desde el inicial, se pasa a traves de a, y a partir de esos estados ya vale $(b \vee c)$. El output de LTSA:

Formula !PROP4 = (!a R (!c !b))
LTL Property Check...
– States: 8 Transitions: 28 Memory used: 15442K
No LTL Property violations detected.
LTL Property Check in: 0ms

5.5.

No vale porque todos los ciclos pasan por el estado del medio que puede transicionar en con todos los simbolos, por lo tanto no es cierto que siempre valga alguno de ellos.

Formula !PROP5 = (((true U !c) (true U !b)) (true U !a))
Analysing...
Depth 2 – States: 2 Transitions: 3 Memory used: 15254K
Trace to property violation in PROP5:
a
b
Analysed in: 0ms

5.6.

No vale ya que se puede ciclar entre los estados del medio y abajo a la izquierda. Este es el output de LTSA:

Formula !PROP6 = X (!c X !c)
Analysing...
Depth 3 – States: 4 Transitions: 7 Memory used: 15069K
Trace to property violation in PROP6:
a
b
a
Analysed in: 0ms