Ingeniería de Software II — Universidad Nacional de Córdoba

Trabajo práctico N^o 3 — FaMAF

Ejercicio 1. Lea los capítulos 6 y 7 de *Concurrency* (Magee & Kramer 2006). En particular, estudie los mecanismos de análisis implementados por la herramienta LTSA.

Lea, también, el artículo:

> Bowen Alpern and Fred B. Scneider. Defining Liveness. *Information Processing Letter*, 21:181-185. 1985.

(no se preocupe si no entiende bien las pruebas)

Para este trabajo práctico, apóyese también con los capítulos 3 y 4 de:

> Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

Ejercicio 2. (En referencia al problema de los filósofos [Cap. 6, Concurrency, Magee & Kramer 1999].) Una solución al problema de los filósofos permite que solo cuatro filósofos se sienten a la mesa al mismo tiempo. Especifique el proceso `BUTLER` que, al componerlo con el modelo de la sección 6.2 permite que un máximo de cuatro filósofos ejecuten el evento `sitdown` sin que antes ocurra un evento `arise`. Muestre que este sistema es libre de deadlock.

Ejercicio 3. Demuestre que la intersección de propiedades de safety es una propiedad de safety.

Ejercicio 4. ¿Es cierto que la intersección también preserva las propiedades liveness?

Ejercicio 5. Encuentre alguna propiedad que sea simultáneamente de safety y de liveness.

Ejercicio 6. Con referencia al ejercicio anterior, demuestre que esa propiedad que encontró es única.

Ejercicio 7. Demuestre o refute lo siguiente:

"El complemento de una propiedad de safety es necesariamente una propiedad de liveness".

Ejercicio 8. Considere el alfabeto $\Sigma = \{a, b\}$. Determine si las siguientes propiedades son de *safety*, *liveness*, ambas, o ninguna. Justifique la respuesta:

i.	a^*b^ω	*iii.*	$a^*b^+a^\omega$	*v.*	$(ab)^\omega$
ii.	$(b+a)^+b^\omega$	*iv.*	$(a+b)^*(a^\omega+b^\omega)$	*vi.*	$(ab)^*a^\omega$

¿No se puede modificar? ¿Solo agregar?

Ejercicio 9. Para cada propiedad P del Ejercicio 8 que no sea de safety extiéndala agregando el menor conjunto de trazas de manera que lo sea. Use expresiones ω-regulares. Justifique su respuesta.

Ejercicio 10. Para cada propiedad P del Ejercicio 8 que no sea ni de safety ni de liveness, dé una propiedad de safety S y una de liveness L tal que $S \cap L = P$. Justifique su respuesta.

Ejercicio 11. Dé ejemplos de trazas que violan la siguiente propiedad:

```
property PS = (a->(b->PS|a->PS)|b->a->PS).
```

Ejercicio 12. Complemente sus especificaciones de sistemas concurrentes de los prácticos anteriores con propiedades de safety y liveness, y verifíquelas usando la herramienta LTSA.

Ejercicio 13. Un ascensor tiene una capacidad máxima de 10 personas. En el modelo del sistema de control del ascensor, los pasajeros que ingresan al ascensor se señalan con un evento `enter` y los que salen se señalan con un evento `exit`. Especifique una propiedad de safety en FSP que cuando sea compuesta con el ascensor verifique que el sistema nunca permita que el ascensor controlado por este exceda los diez ocupantes.

Ejercicio 14. (Ej. 7.6 del *Concurrency, State Models and Java Programs, 2nd ed.*, Magee & Kramer 2006.) Dos vecinos conflictivos tiene separado sus terrenos por un campito con moras. Ellos acordaron que se iban a permitir entrar al campito para recoger moras, pero bajo la condición de que a lo sumo uno de ellos está a la vez en el campito. Luego de varias negociaciones llegaron al acuerdo del siguiente protocolo.

Cada vez que uno de los vecinos desea entrar al campito, iza su propia bandera. Si este vecino ve que la bandera del otro ya estaba izada, arría la bandera propia e intenta nuevamente. Si, en cambio, la bandera del otro vecino no está izada, ingresa al campito y recoge las moras que desea. Una vez que haya culminado sale del campito y arría su bandera.

Modele este algoritmo para dos vecinos, `n1` y `n2`. Especifique la propiedad de *seguridad* requeria para el campito y verifique que asegure efectivamente el acceso en exclusión mutua de los vecinos. Especifique las propiedades de *progreso* requeridas para que ambos vecinos puedan recoger moras provisto que se encuentran bajo una *estrategia de scheduling equitativa*. ¿Existe alguna circunstancia adversa en la cual los vecinos no puedan progresar? ¿Qué ocurrre si alguno de los vecinos es egoista?

(En la página 158, el libro da una ayuda para modelar las banderas.)

Ejercicio 15. (Ej. 7.7 del *Concurrency, State Models and Java Programs, 2nd ed.*, Magee & Kramer 2006, sobre el algoritmo de Peterson.) Afortunadamente para los vecinos del ejercicio anterior, un día, Gary Peterson decidió efectuarles una visita y explicarles cómo funciona su algoritmo. Les explicó que, además de las banderas, ambos vecinos deberán compartir un indicador de *turno* que puede tomar los valores 1 o 2. "Esto se utiliza para evitar deadlocks potenciales" aclaró Peterson.

Cuando un vecino quiere entrar al campito, iza su bandera y cede el turno a su vecino marcándolo apropiadamente en el indicador de turno. Si ve la bandera de su vecino izada y el indicador dice que es el turno de su vecino, él no puede entrar y debe esperar o a que su vecino arríe la bandera o ceda el turno. En caso contrario puede ingresar al campito y recojer moras. Al salir del campito deberá arriar su bandera.

Modele el algoritmo de Peterson para los dos vecinos. Verifique que efectivamente evita deadlock y que satisface exclusión mutua (*propiedades de seguidad*), y que ambos vecinos tendrán su turno para recoger moras (*propiedades de progreso*).

(En la página 158, el libro dá un pseudocódigo de una de las componentes del algoritmo y una ayuda para modelar el indicador del turno.)

Ejercicio 16. La Figura 1 describe un laberinto. Escriba un modelo del laberinto en FSP tal que utilizando análisis de deadlock le permita encontrar el camino de salida más corto comenzando desde una posición dada (i.e., el estado inicial es un parámetro)

La intención de este ejercicio es ver cómo hacer derivación automática de schedulers (o controladores) utilizando model checkers.

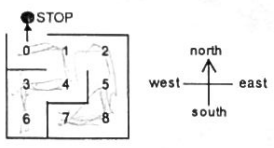


Figura 1: Laberinto

Guía N°3

Ejemplo

$\{a,b\}$

$P = b\Sigma^\omega = (a^*b\Sigma^\omega + a^\omega) \cap (\Sigma^* b \Sigma^\omega)$

- $a^*b\Sigma^\omega + a^\omega$: no ocurre otra cosa más que a antes de b (SAFETY)
- $\Sigma^* b \Sigma^\omega$: va a ocurrir una b. (LIVENESS)

Ejercicio 3

$\sigma \in P \Leftrightarrow \forall \sigma : \forall i \geq 0 : \exists \beta : \sigma[..i]\beta \in P$

P si cumple ⇒ es SAFETY

Entonces si P es SAFETY y Q es SAFETY, $P \cap Q$ es SAFETY

$\sigma \in (P \cap Q) \Leftrightarrow \sigma \in P$ y $\sigma \in Q$

$\Leftrightarrow \forall i \geq 0 : \exists \beta\ \sigma[..i]\beta \in P$ y $\forall i \geq 0 : \exists \beta\ \sigma[..i]\beta \in Q$

$\Leftarrow \forall i \geq 0\ \exists \beta\ \sigma[..i]\beta \in P \cap Q$

Ejercicio 4

$\forall \alpha \in \Sigma^* : \exists \beta \in \Sigma^\omega : \alpha\beta \in P$ si P cumple ⇒ es LIVENESS

¿También preserva intersección? Sí.

Si Q cumple $\forall \alpha \in \Sigma^* : \exists \beta \in \Sigma^\omega : \alpha\beta \in Q$

Si $P \cap Q$ cumple ⇒ $\forall \alpha \in \Sigma^* : \exists \beta \in \Sigma^\omega : \alpha\beta \in P \cap Q$

$\begin{cases} \exists \beta \in \Sigma^\omega\ \alpha\beta \in P & \text{por } \in P \\ \exists \beta \in \Sigma^\omega\ \alpha\beta \in Q & \text{por } \in Q \end{cases}$

$\Leftarrow \exists \beta \in \Sigma^\omega\ \alpha\beta \in P \cap Q$ por def de $\cap$

Ejercicio 5,6

Para ser propiedad de safety debe cumplir $\forall \sigma \in \Sigma^\omega$ si $\sigma \notin P \Leftrightarrow \exists i \geq 0 . \forall \beta \in \Sigma^\omega$ $\sigma[..i]\beta \notin P$.

Para ser prop. de liveness debe cumplir $\forall \alpha \in \Sigma^* \exists \beta \in \Sigma^\omega$ tq $\alpha\beta \in P$

Por lo tanto P debe iniciar con Σ^* pero las palabras deben ser infinitas

Veamos entonces particularizando sin perder generalidad para $\Sigma = \{a,b\}$

Sea $P = (a+b)^* a^\omega$ NO SAFETY ya que la palabra $\sigma = ababb\,b^\omega \notin P$
sin embargo $\sigma[..i]\,a^\omega \in P$

Lo mismo ocurre colocando cualquier caracter de Σ, inclusive agrupados tal como. $P = (a+b)^*(ab)^\omega$ ó $P = (a+b)^*(bba)^\omega$

Solo nos queda como alternativa $P = (a+b)^*(a+b)^\omega$ =

Aquí vemos que cumple safety pues. $\forall \sigma \in \Sigma^\omega$ $\sigma \in P$ por lo tanto antecedente falso $\Rightarrow$ implicación verdadera

También cumple. Liveness pues $\forall \alpha \in \Sigma^*$ con $\beta = a^\omega$ ó b^ω
$\alpha\beta \in P$

Duda con $(ab)^\omega$	$\sigma = ababab\ldots \notin P$ $\sigma[..i]a^\omega \in P$
	corrección de $P = (a+b)^*((a+b)^*)^\omega$

NO! ab^ω sí pertenece a $(a+b)^\omega = P$

Ejercicio 8 $\Sigma = \{a,b\}$

(i) a^*b^ω NO LIVENESS porque no puedo iniciar con ba
no vale $\forall \alpha \in \Sigma^* \exists \beta \in \Sigma^\omega$ $\alpha\beta \in$ (i)

(ii)

NO SAFETY porque sea $\sigma \notin P \Rightarrow \exists i \geq 0 \; \forall \beta \in \Sigma^\omega$ $\sigma[..i]\beta \notin P$
con $\sigma = a^\omega$ y $\beta = b^\omega$ por ejemplo

(ii) $(b+a)^+ b^\omega$ LIVENESS: porque vale $\forall \alpha \in \Sigma^* \exists \beta \in \Sigma^\omega$ $\alpha\beta \in$ (ii)

$\Sigma^* = (b+a)^* = (b+a)^+ + \varepsilon$
Si fuese $\varepsilon \Rightarrow$ no arranca con a ni con b
$\Rightarrow$ no pertenecería al lenguaje

NO SAFETY: con $\sigma = a^\omega b^\omega$ $\sigma \notin P \Rightarrow \exists i \ldots$

(iii) $a^* b^+ a^\omega$ NO LIVENESS ; porque sea $\alpha = abab$ $\nexists \beta \in \Sigma^\omega$ tq $\alpha\beta \in$ (iii)

NO SAFETY : porque $a^\omega \notin$ (iii) Pero si $\exists \beta \in \Sigma^\omega$ tq $\forall i \geq 0$
$a^\omega[\dots i]\beta \in$ (iii)

(iv) $(a+b)^* (a^\omega + b^\omega)$ LIVENESS : si porque $\Sigma^* = (a+b)^*$

NO SAFETY : sea $\sigma = (ab)^\omega$ $\sigma \notin P$ pero
si $\exists \beta \in \Sigma^\omega$ $\forall i \geq 0$ tq $\sigma[\dots i]\beta \in P$

(v) $(ab)^\omega$ NO LIVENESS : $\alpha = aab$ $\nexists \beta \in \Sigma^\omega$ tq $\alpha\beta \in$ (v)

SAFETY : Solo hay un $\sigma \in \Sigma^\omega$ tq $\sigma \in$ (v)
y entonces $\sigma[\dots i]$ termina en a o en
b. si termina en a β será $(ba)^\omega$
si termina en b β será $(ab)^\omega$

(vi) $(ab)^* a^\omega$ NO LIVENESS porque sea $\alpha = abba$ vemos que se
viola la regla. $\forall \alpha \in \Sigma^*$ $\exists \beta \in \Sigma^\omega$ tq $\alpha\beta \in$ (vi)

NO SAFETY: no porque sea $\sigma = (ab)^\omega$ es claro
que $\sigma \notin P$ sin embargo. $\forall i \geq 0$ $\exists \beta \in \Sigma^\omega$
tq $\sigma[\dots i]\beta \in P$ por ejemplo con $\beta = a^\omega$

como $\sigma = ababab\dots$
(i i i señalando posiciones)

Puedo cortar en cualquier i y agregar β
y tendré $\dots aa^\omega = a^\omega$ ó $ab\dots aba^\omega$
ambas pertenecientes a (vi)

Ejercicio 9

(i) $a^*b^\omega + a^\omega$

$(ab)^\omega$

El único caso que no permite decir que P es de safety es que la b nunca aparezca, lo cual no nos dice que existe violación de la regla hasta ese momento. Cualquier otra combinación no pertenece al lenguaje como aba... ó está contemplada como b^ω

(ii) $(b+a)^+ b^\omega + (a+b)^{*\omega}$

Igual que (iv)

(iii) $a^*b^+a^\omega + a^*b^\omega + a^\omega$

En este caso vemos que debe ocurrir una b pero si $\sigma = a^\omega$ es posible en cualquier momento que cortamos a σ arreglar la traza con $\sigma[..i]ba^\omega$

Lo mismo ocurre si tenemos trazas como $a...bb...=\sigma$ es verdad que se pueden cortar en cualquier parte y arreglarlas con $\sigma[..i]a^\omega$.

Los demás casos como b^ω están contemplados o no corresponden con el lenguaje.

(iv) $(a+b)^*(a^\omega+b^\omega) + (a+b)^{*\omega}$

Este es el caso que nos obliga a contemplar todas las trazas debido a que la propiedad es de liveness como es posible arrancar con cualquier inicio entonces no existen prefijos malos y es posible $\forall \sigma \in \Sigma^\omega$ arreglar $\sigma[..i]a^\omega$ ó $\sigma[..i]b^\omega$ ∴ este caso es inutil ya que todas las $\sigma \in \Sigma^\omega$ son de esta forma. $(iv) = \Sigma^\omega$.

(v) $(ab)^\omega$

Ya es de safety ya que hay solo una traza posible

(vi) $(ab)^*a^\omega + (ab)^\omega$

En este caso, agregamos solo el contraejemplo del ej. 8. y vemos que cualquier otra combinación queda descartada por el lenguaje ya sea por comenzar con b ó por no cumplir la secuencia ababab...

Ejercicio 10

(i) $a^* b^\omega = (a^* b^\omega + a^\omega) \cap ((a+b)^* b^\omega)$

SAFETY — LIVENESS.

(iii) $a^* b^+ a^\omega = (a^* b^+ a^\omega + a^\omega + a^* b^\omega) \cap ((a+b)^* b^+ a^\omega)$

SAFETY — LIVENESS

(vi) $(ab)^* a^\omega = ((ab)^* a^\omega + (ab)^\omega) \cap ((a+b)^* a^\omega)$

SAFETY — LIVENESS.

Ejercicio 11

Property PS = (a → b → PS
a → PS
b → a → PS.)

bb... viola la propiedad.
abbbb... viola la propiedad
abababbb...

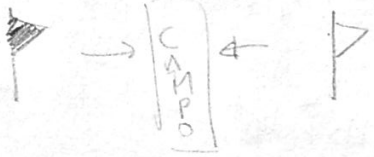
Ejercicio 13

Const MAX = 10
Range T = [0..11]
ASCENSOR = ASCENSOR[0]
ASCENSOR[cap:T] = (when (cap < 11) enter → ASCENSOR[cap+1]
when (cap > 0) exit → ASCENSOR[cap-1]).

(último número definido)

property CAP_CHECK = CAP_CHECK[0]
CAP_CHECK[cap:T] = (when (cap < MAX) enter → CAP_CHECK[cap+1]
when (cap > 0) exit → CAP_CHECK[cap+1]).

EJERCICIO 14



FLAG[up:Bool] = (when (up) down → FLAG[False]
| when (!up) up → FLAG[True]
| query[up] → FLAG[up]).

||FLAGS = ([1..2] : FLAG).

VECINO1 = (2.query[0] → 1.up → montes1 → 1.down → VECINO1)

VECINO2 = (1.query[0] → 2.up → montes2 → 2.down → VECINO2)

property

[] → (montes1 y montes2)

```
Pos [n:T] = (when (n = 0) (este → Pos[1]
                           |norte → STOP.)
          | when (n = 1) (oeste → Pos[0]
                          |este → Pos[2])
          :
          )
```

```
LAB = Pos [5]
        ↑
      parametro
```

```
Pos [vengo desde:T] [estoy:T] = when (desde≠3 y estoy=6)  norte → Pos[6][3]
                              | when (desde≠4 y estoy=3)  este → Pos[3][4]
                              | when (desde≠1 y estoy=4)  norte → Pos[4][1]
                              | when (desde≠3 y estoy=4)  oeste → Pos[4][5]
                              | when (desde≠4 y estoy=1)  sur → Pos[1][4]
                              | when (desde≠0 y estoy=1)  oeste → Pos[1][0]
                              | when (desde≠2 y estoy=1)  este → Pos[1][2]
                              | when (estoy=0)  norte → STOP.
                              | when (desde≠5 y estoy=2)  sur → Pos[2][5]
                              | when (desde≠6 y estoy=5)  sur → Pos[5][6]
                              | when (desde≠7 y estoy=6)  oeste → Pos[6][7]
                              | when (desde≠6 y estoy=7)  este → Pos[7][6]
```