Ingeniería del Software II
4 - Propiedades de los sistemas reactivos y su análisis

Propiedades de los sistemas concurrentes

Con frecuencia, los programas concurrentes suelen ser reactivos, y sus características diferentes de las de los programas secuenciales convencionales.

Por esto, las propiedades que en general se desea garantizar de programas concurrentes difieren de las propiedades de programas secuenciales. Algunas de estas son:

- No violación de **invariantes** de sistema
- Ausencia de **starvation**
- Ausencia de **deadlock**
- Garantía de **exclusion mutua** en el acceso a recursos compartidos
- Ausencia de **livelock**

Categorías de propiedades

Una clasificación clásica de las propiedades de los sistemas reactivos incluye las siguientes cuatro categorías:

- Propiedades de **alcanzabilidad** ("reachability")
- Propiedades de **seguridad** ("safety")
- Propiedades de **vitalidad** ("liveness")
- Propiedades de **equidad** ("fairness")

Propiedades de alcanzabilidad (“reachability”)

“Es posible que el sistema llegue a algún estado de un conjunto dado”

Por ejemplo:

- una (o alguna) componente entra a la region crítica
- es posible llegar a un estado de ERROR (ej: deadlock, violación de exclusión mutua, violación de invariante, etc.)

En este último caso nos interesa que se cumpla la negación de la propiedad. Es usual que la propiedad de interés sea la negación de la alcanzabilidad.

Propiedades de seguridad (“safety”)

“Nunca va a pasar nada malo”

Por ejemplo:

- no es posible llegar a un estado de deadlock
- se garantiza exclusión mutua
- el sistema preserva un invariante dado

En general, la negación de una propiedad safety es una propiedad de alcanzabilidad

Propiedades de vitalidad ("liveness")

"Siempre es posible que algo bueno ocurra"

Por ejemplo:

- un (sub)proceso dado termina su ejecución
- es posible alcanzar un estado de estabilidad
- si se llama al ascensor, este llegará en algún momento

Propiedades de equidad ("fairness")

"Siempre ocurrirá algo de manera frecuente"

Por ejemplo:

- siempre que un proceso esté esperando para entrar a una región crítica, entonces logrará entrar
- siempre que un proceso solicite periódicamente un recurso finalmente se le será asignado.
- un proceso dado no sufre de inanición

Pospondremos su estudio en más detalle para la próxima unidad.

Propiedades como conjuntos de trazas

La semántica de procesos (en realidad, una de las semánticas más simples para procesos) puede definirse como el conjunto de todas sus ejecuciones. Cada ejecución de un proceso puede verse como la sucesión de eventos en los cuales el proceso se involucra (y en el orden en que lo hace).

Ejemplo: Consideremos el siguiente proceso:

```
MAKER = (make->ready->MAKER).
USER  = (ready->use->USER).

||MAKER_USER = (MAKER || USER).
```

La única traza posible para **`MAKER`** es:

```
make, ready, make, ready, make, ...
```

Algunas trazas posibles para **`MAKER_USER`** son:

```
make, ready, use, make, ready, use, ...
make, ready, make, use, ready, make, ready, use, ...
```

Propiedades como conjuntos de trazas (cont.)

Al igual que los procesos, las propiedades tienen una semántica dada en términos de conjuntos de trazas. Una propiedad P se identifica con todas las sucesiones de eventos atómicos que exhiben la propiedad P.

Por ejemplo, la propiedad “no ocurren dos producciones (**`make`**) sucesivas” contiene las siguientes trazas:

```
use, use, use, use, use, use, use, ...
make, use, make, use, make, use, make, use, ...
ready, ready, ready, ready, ready, ...
```

Propiedades como conjuntos de trazas (cont.)

Al igual que los procesos, las propiedades tienen una semántica dada en términos de conjuntos de trazas. Una propiedad P se identifica con todas las sucesiones de eventos atómicos que exhiben la

Por ejemplo, la propie

(**make**) sucesivas" contie

Notar que una propiedad describe un conjunto de trazas que no necesariamente se corresponde con las trazas definidas por el comportamiento por el modelo bajo estudio

```
use, use, use, use, use, use, use, ...
make, use, make, use, make, use, make, use, ...
ready, ready, ready, ready, ready, ...
```

Lenguajes ω-regulares

Dado un lenguaje regular $A \subseteq \Sigma^*$ definimos

$$A^\omega \stackrel{\text{def}}{=} \{\sigma_1\sigma_2\sigma_3\ldots \mid \forall i \geq 0 : \sigma_i \in A \wedge \sigma_i \neq \epsilon\}$$

es decir, A^ω es el lenguaje que contiene todas las concatenaciones infinitas de palabras no vacías de A.

Un lenguaje L se dice ω-*regular* si existen lenguajes regulares A_i y B_i, $0 \leq i \leq k$, tal que $\epsilon \notin B_i \neq \varnothing$ y

$$L = \bigcup_{0 \leq i \leq k} A_i \cdot B_i^\omega$$

donde · denota la concatenación habitual de lenguajes.

Propiedad: Los lenguajes ω-regulares son cerrados por union, intersección y complemento.

Lenguajes ω-regulares

Dado un lenguaje regular $A \subseteq$

$$A^\omega \stackrel{\text{def}}{=} \{\sigma_1\sigma_2\sigma_3 \ldots$$

También las
denominaremos **fragmentos de trazas**

es decir, A^ω es el lengu… qu… las concatenaciones infinitas de palabras no vacías de A.

> Un lenguaje L se dice ω-*regular* si existen lenguajes regulares A_i y B_i, $0 \leq i \leq k$, tal que $\epsilon \notin B_i \neq \varnothing$ y
>
> $$L = \bigcup_{0 \leq i \leq k} A_i \cdot B_i^\omega$$

donde $\cdot$ denota la concatenación habitual de lenguajes.

Propiedad: Los lenguajes ω-regulares son cerrados por union, intersección y complemento.

Lenguajes ω-regulares

Ejemplo:

La propiedad P sobre el proceso **MAKER_USER** puede escribirse como cualquiera de las siguientes expresiones ω-regulares:

$$\big(\,(\mathtt{make} + \varepsilon)\,(\mathtt{ready} + \mathtt{use})\,\big)^{\omega}$$

$$\overline{\Big(\big(\mathtt{make} + \mathtt{ready} + \mathtt{use}\big)^{*}\,\mathtt{make}\,\mathtt{make}\,\big(\mathtt{make} + \mathtt{ready} + \mathtt{use}\big)^{\omega}\Big)}$$

Lenguajes ω-regulares

Ejemplo:

La propiedad P sobre el proceso **`MAKER_USER`** pue[illegible] escribirse como cualquiera de las siguientes ex[illegible] ω-regulares:

Los lenguajes ω-regulares son cerrados por complemento

$$\big(\,(\mathtt{make} + \varepsilon)\,(\mathtt{ready} + \mathtt{use})\,\big)^{\omega}$$

$$\overline{\Big(\big(\mathtt{make} + \mathtt{ready} + \mathtt{use}\big)^{*}\,\mathtt{make}\,\mathtt{make}\,\big(\mathtt{make} + \mathtt{ready} + \mathtt{use}\big)^{\omega}\Big)}$$

Formalización de propiedades safety

Observar:

- Si una traza viola una propiedad de safety, lo hace en un "instante" finito.
- Si un prefijo de una traza (infinita) viola una propiedad de safety, no hay forma de remediarlo (cualquiera sea la forma en que se continúe, la traza no dejará de violar la propiedad).

Es decir:

Un conjunto de trazas $P \subseteq \Sigma^\omega$ es una propiedad de safety si cumple

$$\forall \sigma : \quad \sigma \notin P \implies \exists i \geq 0 : \forall \beta : \ \sigma[..i]\beta \notin P$$

o equivalentemente:

donde $\sigma[..i]$ denota el prefijo i-esimo de σ.

Formalización de propiedades safety

Observar:

- Si una traza viola una propiedad de safety, lo hace en un "instante" finito.
- Si un prefijo de una traza (infinita) viola una propied[...] de remediarlo (cualquiera sea la forma en que s[...] de violar la propiedad).

Es decir:

Un conjunto de trazas $P \subseteq \Sigma^\omega$ es una propiedad de safe[...] si cumple

$$\forall \sigma : \quad \sigma \notin P \implies \exists i \geq 0 : \forall \beta : \ \sigma[..i]\beta \notin P$$

Una propiedad P es de safety si toda palabra en su complemento tiene un "prefijo malo"

o equivalentemente:

donde $\sigma[..i]$ denota el prefijo i-esimo de σ.

Formalización de propiedades safety

Observar:

- Si una traza viola una propiedad de safety, lo hace en un “instante” finito.
- Si un prefijo de una traza (infinita) viola una propiedad de safety, no hay forma de remediarlo (cualquiera sea la forma en que se continúe, la traza no dejará de violar la propiedad).

Es decir:

Un conjunto de trazas $P \subseteq \Sigma^{\omega}$ es una propiedad d

$$\forall\sigma : \quad \sigma \notin P \;\Rightarrow\; \exists i \geq 0 : \forall$$

Una propiedad P es de safety
si toda traza infinita que se pueda
“aproximar” finitamente con trazas
de P, también está en P

o equivalentemente:

$$\forall\sigma : \quad \forall i \geq 0 : \exists\beta : \; \sigma[..i]\beta \in P \;\Rightarrow\; \sigma \in P$$

donde $\sigma[..i]$ denota el prefijo i-esimo de σ.

Formalización de propiedades safety

Ejemplos:

$$\big(\,(\texttt{make} + \varepsilon)\,(\texttt{ready} + \texttt{use})\,\big)^{\omega}$$

- Complemento:

$$(\texttt{make} + \texttt{ready} + \texttt{use})^{*}\,\texttt{make}\,\texttt{make}\,(\texttt{make} + \texttt{ready} + \texttt{use})^{\omega}$$

Formalización de propiedades safety

Ejemplos:

$$\Big(\,(\texttt{make} + \varepsilon)\,(\texttt{ready} + \texttt{use})\,\Big)^{\omega}$$

- Complemento:

Prefijos malos

$$(\texttt{make} + \texttt{ready} + \texttt{use})^{*}\,\texttt{make}\,\texttt{make}\,(\texttt{make} + \texttt{ready} + \texttt{use})^{\omega}$$

Formalización de propiedades safety

Ejemplos:

$$\left((\mathtt{make} + \varepsilon)\,(\mathtt{ready} + \mathtt{use}) \right)^{\omega}$$

- Complemento:

$$(\mathtt{make} + \mathtt{ready} + \mathtt{use})^{*}\,\mathtt{make}\,\mathtt{make}\,(\mathtt{make} + \mathtt{ready} + \mathtt{use})^{\omega}$$

Prefijos malos

$$\left((\mathtt{make} + \mathtt{ready})^{*}\,(\mathtt{use}\,\mathtt{ready}^{*}\,(\mathtt{use} + \varepsilon)\,\mathtt{ready}^{*}\mathtt{make})^{*}\right)^{\omega}$$

Sólo se puede usar hasta dos productos sin que antes se haga otro

Formalización de propiedades safety

Ejemplos:

$$\big(\,(\texttt{make} + \varepsilon)\,(\texttt{ready} + \texttt{use})\,\big)^\omega$$

- Complemento:

Prefijos malos

$$(\texttt{make} + \texttt{ready} + \texttt{use})^*\,\texttt{make}\,\texttt{make}\,(\texttt{make} + \texttt{ready} + \texttt{use})^\omega$$

$$\big((\texttt{make} + \texttt{ready})^*\,(\texttt{use}\,\texttt{ready}^*\,(\texttt{use} + \varepsilon)\,\texttt{ready}^*\texttt{make})^*\big)^\omega$$

- Complemento:

$$(\texttt{make} + \texttt{ready} + \texttt{use})^*\,(\texttt{use}\,\texttt{ready}^*\texttt{use}\,\texttt{ready}^*\texttt{use})\,(\texttt{make} + \texttt{ready} + \texttt{use})^\omega$$

Formalización de propiedades safety

Ejemplos:

$$\big(\,(\mathtt{make} + \varepsilon)\,(\mathtt{ready} + \mathtt{use})\,\big)^{\omega}$$

Prefijos malos

- Complemento:

$$(\mathtt{make} + \mathtt{ready} + \mathtt{use})^{*}\,\mathtt{make}\,\mathtt{make}\,(\mathtt{make} + \mathtt{ready} + \mathtt{use})^{\omega}$$

$$\big((\mathtt{make} + \mathtt{ready})^{*}\,(\mathtt{use}\,\mathtt{ready}^{*}\,(\mathtt{use} + \varepsilon)\,\mathtt{ready}^{*}\mathtt{make})^{*}\big)^{\omega}$$

Prefijos malos

- Complemento:

$$(\mathtt{make} + \mathtt{ready} + \mathtt{use})^{*}\,(\mathtt{use}\,\mathtt{ready}^{*}\mathtt{use}\,\mathtt{ready}^{*}\mathtt{use})\,(\mathtt{make} + \mathtt{ready} + \mathtt{use})^{\omega}$$

Formalización de propiedades liveness

Observar:

- Ningún fragmento de traza (finito) viola una propiedad de liveness: Si el evento que se esperaba que ocurriera aún no lo hizo, puede suceder más adelante.

Es decir:

Un conjunto de trazas $P \subseteq \Sigma^\omega$ es una propiedad de liveness si cumple

$$\forall \alpha \in \Sigma^* : \exists \beta \in \Sigma^\omega : \alpha\beta \in P$$

Es decir, una propiedad P es de liveness si toda palabra en su complemento **NO** tiene un "prefijo malo"

Formalización de propiedades liveness

Ejemplo:

$$(\texttt{make} + \texttt{ready})^* \, \texttt{use} \, (\texttt{make} + \texttt{ready} + \texttt{use})^\omega$$

En algún momento se puede consumir

Formalización de propiedades liveness

Ejemplo:

$$(\texttt{make} + \texttt{ready})^{*}\, \texttt{use}\, (\texttt{make} + \texttt{ready} + \texttt{use})^{\omega}$$

es equivalente a:

$$(\texttt{make} + \texttt{ready} + \texttt{use})^{*}\, \texttt{use}\, (\texttt{make} + \texttt{ready} + \texttt{use})^{\omega}$$

Formalización de propiedades liveness

Ejemplo:

$$(\texttt{make} + \texttt{ready})^{*}\,\texttt{use}\,(\texttt{make} + \texttt{ready} + \texttt{use})^{\omega}$$

Todos los posibles prefijos aparecen en la propiedad

es equivalente a:

$$(\texttt{make} + \texttt{ready} + \texttt{use})^{*}\,\texttt{use}\,(\texttt{make} + \texttt{ready} + \texttt{use})^{\omega}$$

Formalización de propiedades liveness

Ejemplo:

$$(\texttt{make} + \texttt{ready})^* \, \texttt{use} \, (\texttt{make} + \texttt{ready} + \texttt{use})^\omega$$

Todos los posibles prefijos aparecen en la propiedad

es equivalente a:

$$(\texttt{make} + \texttt{ready} + \texttt{use})^* \, \texttt{use} \, (\texttt{make} + \texttt{ready} + \texttt{use})^\omega$$

$$\left((\texttt{ready} + \texttt{use})^* \, \texttt{make}\right)^\omega$$

La producción se realiza indefinidamente (se producen infinitos "make" a lo largo de la ejecución)

Formalización de propiedades liveness

Ejemplo:

$$(\mathtt{make} + \mathtt{ready})^* \, \mathtt{use} \, (\mathtt{make} + \mathtt{ready} + \mathtt{use})^\omega$$

Todos los posibles prefijos aparecen en la propiedad

es equivalente a:

$$(\mathtt{make} + \mathtt{ready} + \mathtt{use})^* \, \mathtt{use} \, (\mathtt{make} + \mathtt{ready} + \mathtt{use})^\omega$$

$$\left((\mathtt{ready} + \mathtt{use})^* \, \mathtt{make}\right)^\omega$$

es equivalente a:

$$(\mathtt{ready} + \mathtt{use} + \mathtt{make})^* \left((\mathtt{ready} + \mathtt{use})^* \, \mathtt{make}\right)^\omega$$

Formalización de propiedades liveness

Ejemplo:

$$(\mathtt{make} + \mathtt{ready})^* \, \mathtt{use} \, (\mathtt{make} + \mathtt{ready} + \mathtt{use})^\omega$$

es equivalente a:

$$(\mathtt{make} + \mathtt{ready} + \mathtt{use})^* \, \mathtt{use} \, (\mathtt{make} + \mathtt{ready} + \mathtt{use})^\omega$$

Todos los posibles prefijos aparecen en la propiedad

$$\left((\mathtt{ready} + \mathtt{use})^* \, \mathtt{make}\right)^\omega$$

es equivalente a:

$$(\mathtt{ready} + \mathtt{use} + \mathtt{make})^* \left((\mathtt{ready} + \mathtt{use})^* \, \mathtt{make}\right)^\omega$$

Todos los posibles prefijos aparecen en la propiedad

Safety, liveness y todas las demás

Teorema [Alpern & Schneider 85]:

Toda propiedad puede escribirse como la intersección de una propiedad de safety y una de liveness.

Ejemplo: ***a* until *b***

algún evento ***b*** ocurre inmediatamente precedido por una serie ininterrumpida de eventos ***a*** (léase: "***a*** ocurre hasta que ocurre ***b***")

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

A la izquierda de la intersección se da la propiedad safety y a la derecha, la de liveness.

(¿Qué significa cada una de estas propiedades?)

Análisis de propiedades en FSP

Los distintos tipos de propiedades en FSP se verifican y/o modelan de manera distinta.

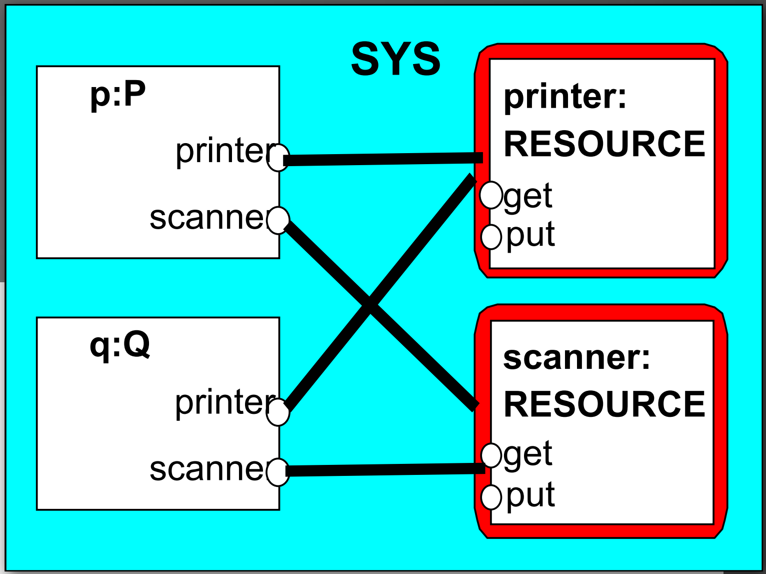
Deadlock

¿Cuáles son las cuatro condiciones necesarias para que ocurra deadlock?

Análisis de propiedades en FSP

Los distintos tipos de propiedades en FSP se verifican y/o modelan de manera distinta.

Deadlock



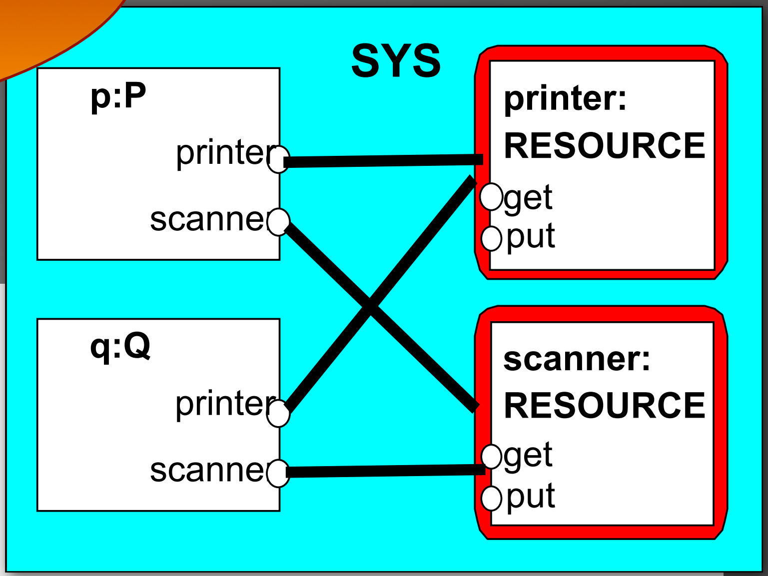

```
RESOURCE = (get -> put -> RESOURCE).

P = (printer.get -> scanner.get -> copy
                 -> printer.put -> scanner.put -> P).

Q = (scanner.get -> printer.get -> copy
                 -> printer.put -> scanner.put -> Q).

||SYS = (p:P || q:Q
        || {p,q}::printer:RESOURCE
        || {p,q}::scanner:RESOURCE
        ).
```

Análisis de propiedades en FSP

LTSA puede checkear deadlock. Lo hace mediante una búsqueda exahustiva.

Los distintos … se verifican y/o modelan de m…

Deadlock




```
RESOURCE = (get -> put -> RESOURCE).

P = (printer.get -> scanner.get -> copy
                 -> printer.put -> scanner.put -> P).

Q = (scanner.get -> printer.get -> copy
                 -> printer.put -> scanner.put -> Q).

||SYS = (p:P || q:Q
        || {p,q}::printer:RESOURCE
        || {p,q}::scanner:RESOURCE
        ).
```

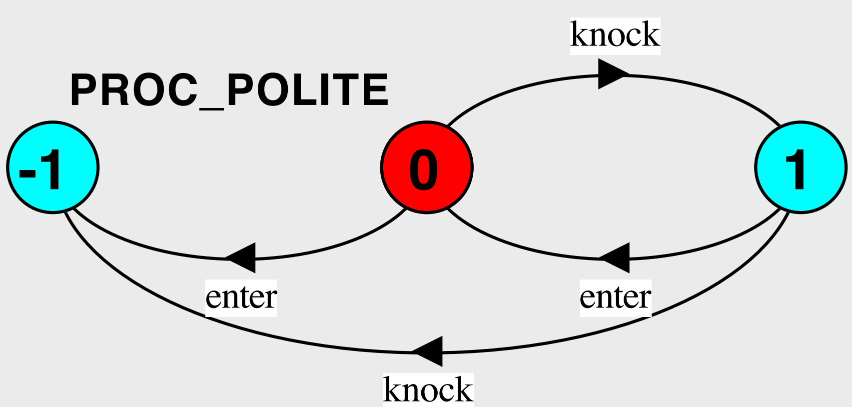
Análisis de propiedades en FSP

Safety

En FSP podemos expresar las propiedades de safety mediante procesos haciendo uso del estado de **ERROR** para indicar cuáles son las trazas que violan esta propiedad (i.e. los prefijos malos).

Ejemplo: Una persona educada golpea sólo una vez antes de entrar y no entra sin golpear.

```
PROC_POLITE = (knock->(enter->PROC_POLITE
                      |knock->ERROR
                      )
              |enter->ERROR
              ).
```



Toda ejecución que lleva a ERROR es un "prefijo malo"

Análisis de propiedades en FSP

Safety (cont.)

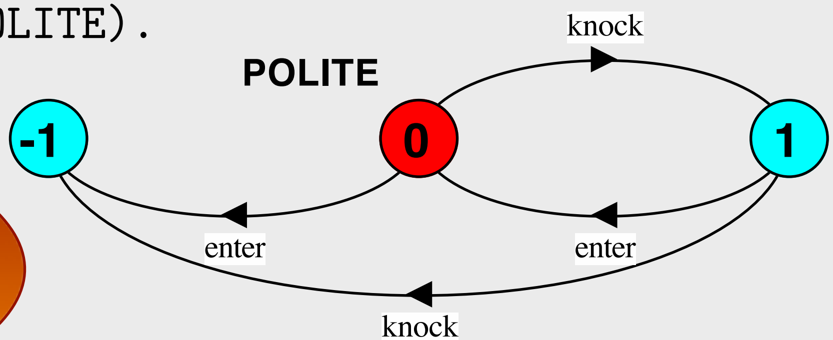
FSP provee una construcción que nos permite obviar el agregado “a mano” de los estados de error:

```
property N = P
```

indica que los eventos correspondientes al alfabeto de **`P`** deben ocurrir respetando el patrón de ocurrencias definido por **`P`** (toda ocurrencia no definida por **`P`** lleva al estado de **`ERROR`**).

En el ejemplo anterior:

```
property POLITE = (knock -> enter -> POLITE).
```



La declaración de un proceso **`P`** como **`property`** determina que la omisión de una acción en un estado de **`P`** interpretado como **proceso** induce una transición errónea en **`P`** como **propiedad**.

Análisis de propiedades en FSP

Safety (cont.)

Ejemplo

```
const Max = 3
range Int = 0..Max

SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int]    = (up->SEMA[v+1]
                 |when(v>0) down->SEMA[v-1]
                 ).

LOOP = (mutex.down->enter->exit->mutex.up->LOOP).

||SEMADEMO = (p[1..3]:LOOP
             || {p[1..3]}::mutex:SEMAPHORE(1)).
```

Análisis de propiedades en FSP

Safety (cont.)

Ejemplo

```
const Max = 3
range Int = 0..Max

SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int]    = (up->SEMA[v+1]
                 |when(v>0) down->SEMA[v-1]
                 ).

LOOP = (mutex.down->enter->exit->mutex.up->LOOP).

||SEMADEMO = (p[1..3]:LOOP
             || {p[1..3]}::mutex:SEMAPHORE(1)).

property MUTEX
= (p[i:1..3].enter->p[i].exit->MUTEX).

||CHECK = (SEMADEMO || MUTEX).
```

Propiedad de exclusión mututua

Análisis de propiedades en FSP

Safety (cont.)

LTSA puede checkear safety. También lo hace mediante búsqueda exhaustiva.

Ejemplo

```
const Max = 3
range Int = 0..Max

SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int]    = (up->SEMA[v+1]
                 |when(v>0) down->SEMA[v-1]
                 ).

LOOP = (mutex.down->enter->exit->mutex.up->LO

||SEMADEMO = (p[1..3]:LOOP
             || {p[1..3]}::mutex:SEMAPHORE(1)).

property MUTEX
= (p[i:1..3].enter->p[i].exit->MUTEX).

||CHECK = (SEMADEMO || MUTEX).
```

Verificar cambiando la inicialización del semáforo a 2.

Propiedad de exclusión mututua

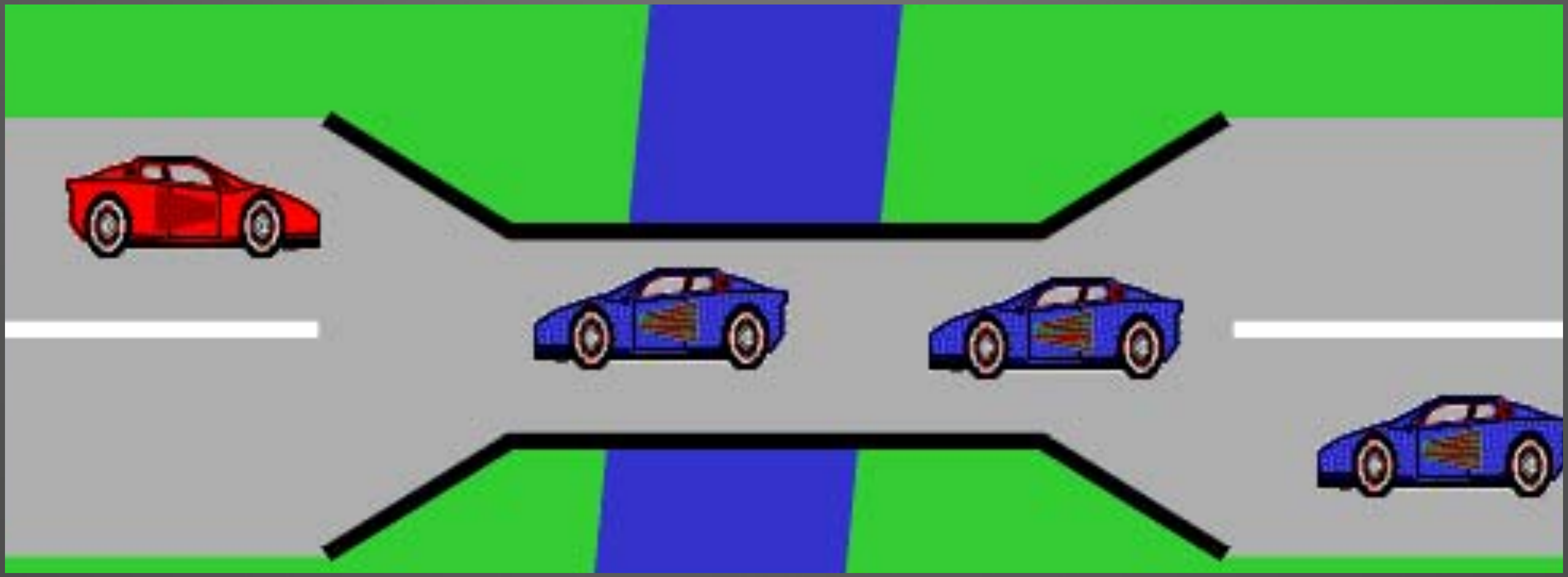
Análisis de propiedades en FSP

Safety (cont.)

Ejemplo: Un puente de una sola vía

Un puente sobre un río tiene el ancho suficiente para permitir tráfico en sólo una dirección. Por consiguiente, los autos sólo pueden cruzar el puente concurrentemente si viajan en la misma dirección.

Cuando dos autos que viajan en direcciones opuestas ingresan al puente al mismo tiempo ocurre una violación de seguridad.

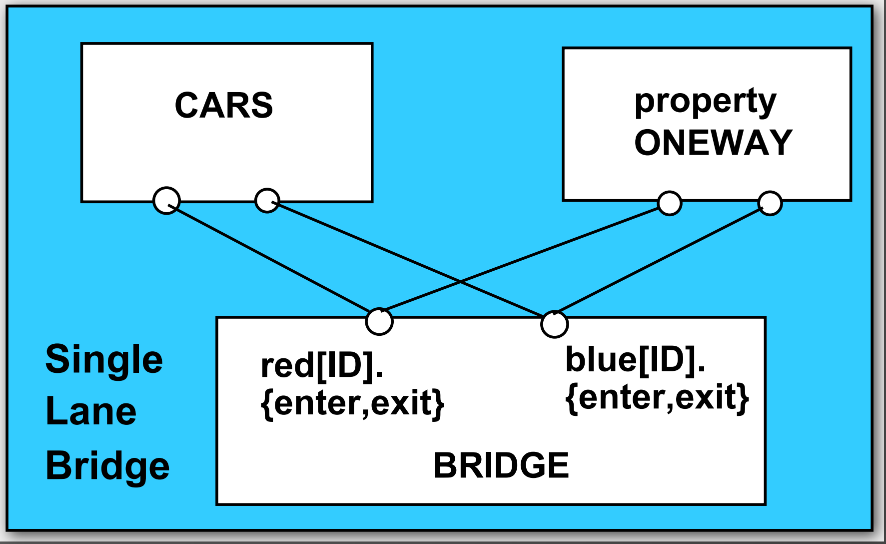


Ejemplo: Un puente de una sola vía

```
const N = 3 // number of each type of car
range T = 0..N // type of car count
range ID= 1..N // car identities

CAR = (enter->exit->CAR).

/* cars may not overtake each other */
NOPASS1   = C[1],
C[i:ID]   = ([i].enter -> C[i%N+1]).

NOPASS2   = C[1],
C[i:ID]   = ([i].exit -> C[i%N+1]).

||CONVOY = ([ID]:CAR || NOPASS1 || NOPASS2).

||CARS = (red:CONVOY || blue:CONVOY).
```

```
BRIDGE = BRIDGE[0][0],
BRIDGE[nr:T][nb:T] =
        (when (nb==0)
          red[ID].enter  -> BRIDGE[nr+1][nb]
         |red[ID].exit    -> BRIDGE[nr-1][nb]
         |when (nr==0)
          blue[ID].enter -> BRIDGE[nr][nb+1]
         |blue[ID].exit   -> BRIDGE[nr][nb-1]
        ).

||SingleLaneBridge = (CARS || BRIDGE || ONEWAY ).
```

```
property ONEWAY = ( red[ID].enter  -> RED[1]
                  | blue[ID].enter -> BLUE[1]
                  ),
RED[i:ID] = (red[ID].enter -> RED[i+1]
            |when(i==1)red[ID].exit  -> ONEWAY
            |when( i>1)red[ID].exit  -> RED[i-1]
            ),
BLUE[i:ID] = (blue[ID].enter -> BLUE[i+1]
             |when(i==1)blue[ID].exit  -> ONEWAY
             |when( i>1)blue[ID].exit  -> BLUE[i-1]
             ).
```



Análisis de propiedades en FSP

Liveness

LTSA solo puede manejar un conjunto reducido de propiedades de liveness:

```
progress N = { a1, a2, ...}
```

Esto indica que, en toda ejecución fuertemente equitativa (strongly fair), alguna de la acciones **a1**, **a2**,... debe ejecutarse infinitas veces

Ejemplo:

```
COIN = ( toss -> heads -> COIN
       | toss -> tails -> COIN).
progress HEADS = {heads}
```

Análisis de prop

LTSA puede checkear progreso. En este caso, el algoritmo se basa en la búsqueda de ciclos que atrapen las acciones en **`N`**

Liveness

LTSA solo puede manejar un con[...] reduc[...] liveness:

```
progress N = { a1, a2, ...}
```

Esto indica que, en toda ejecución fuertemente equitativa (strongly fair), alguna de la acciones **`a1`**, **`a2`**,... debe ejecutarse infinitas veces

Ejemplo:

```
COIN = ( toss -> heads -> COIN
       | toss -> tails -> COIN).
progress HEADS = {heads}
```

Análisis de prop

Liveness

LTSA solo puede manejar un con   reduc
liveness:

```
progress N = { a1, a2, ...}
```

Esto indica que, en toda ejecución fuer
alguna de la acciones **a1**, **a2**,... debe ejecu

Ejemplo:

```
COIN = ( toss -> heads -> COIN
       | toss -> tails -> COIN).
progress HEADS = {heads}
```

LTSA puede checkear progreso. En este caso, el algoritmo se basa en la búsqueda de ciclos que atrapen las acciones en **N**

Explicación: LTSA verifica bajo la suposición de **strong fairness**. Sin esta suposición, la propiedad es falsa.

LTSA la verifica verdadera

Análisis de prop

LTSA puede checkear progreso. En este caso, el algoritmo se basa en la búsqueda de ciclos que atrapen las acciones en `N`

Liveness

LTSA solo puede manejar un con... redu... liveness:

```
progress N = { a1, a2, ...}
```

Explicación: LTSA verifica bajo la suposición de **strong fairness**. Sin esta suposición, la propiedad es falsa.

Esto indica que, en toda ejecución fuer... alguna de la acciones **a1**, **a2**,... debe ejecu...

Ejemplo:

```
COIN = ( toss -> heads -> COIN
       | toss -> tails -> COIN).
progress HEADS = {heads}
```

LTSA la verifica verdadera

Ejemplo: para el puente de una sola vía

```
progress BLUECROSS = {blue[ID].enter}
progress REDCROSS =  {red[ID].enter}
```

Análisis de prop

LTSA puede checkear progreso. En este caso, el algoritmo se basa en la búsqueda de ciclos que atrapen las acciones en **`N`**

Liveness

LTSA solo puede manejar un con educ
liveness:

```
progress N = { a1, a2, ...}
```

Explicación: LTSA verifica bajo la suposición de **strong fairness**. Sin esta suposición, la propiedad es falsa.

Esto indica que, en toda ejecución fuer
alguna de la acciones **`a1`**, **`a2`**,... debe ejecu

Ejemplo:

```
COIN = ( toss -> heads -> COIN
       | toss -> tails -> COIN).
progress HEADS = {heads}
```

LTSA la verifica verdadera

Ejemplo: para el puente de una sola vía

LTSA las verifica correctas!! :-(

```
progress BLUECROSS = {blue[ID].enter}
progress REDCROSS =  {red[ID].enter}
```

Prioridades entre acciones

LTSA asume en general que la elección es equitativa (fair), por eso no reporta error de progreso en el ejemplo de la moneda o en el puente de una sola vía.

Sin embargo, sabemos que es posible que los autos azules (o los rojos) acaparen el puente haciendo que los rojos (o los azules) esperen por siempre.

Entonces, para detectar problemas de progreso debemos imponer poltícas de scheduling en las acciones que modele la situación en la que el puente (o cualquier otro sistema en general) es sometido a usos extremos.

Operador de alta prioridad: `||C = (P||Q)<<{`a_1`,...,`a_n`}` especifica una composición en la cual las acciones a_1`,...,`a_n tienen mayor prioridad que cualquier otra acción en `P||Q`, incluyendo `tau`.

Operador de baja prioridad: `||C = (P||Q)>>{`a_1`,...,`a_n`}` especifica una composición en la cual las acciones a_1`,...,`a_n tienen menos prioridad que cualquier otra acción en `P||Q`, incluyendo `tau`.

Análisis de propiedades en FSP

Liveness (cont.)

Ejemplo: Siguiendo con el puente de una sola vía deseamos modelar que el puente se encuentra congestionado.

Para ello daremos menor prioridad a los eventos de salida del puente. De esta manera se prioriza a la entrada sobre la salida forzando la congestión del puente:

```
||CongestedBridge =
      SingleLaneBridge >> {red[ID].exit,blue[ID].exit}.
```

Al verificar las propiedades de progreso, LTSA nos indicará ahora una situación de error.

Análisis de propiedades en FSP

Liveness (cont.)

Ejemplo: Siguiendo con el puente de
modelar que el puente se encu

Para ello daremos menor pri
puente. De esta manera se pr
forzando la congestión del puent

Otros model checkers permiten elegir si se desea realizar la verificación bajo **fairness** o no.

```
||CongestedBridge =
      SingleLaneBridge >> {red[ID].exit,blue[ID].exit}.
```

Al verificar las propiedades de progreso, LTSA nos indicará ahora una situación de error.