Seguridad, Vitalidad y Equidad

Maximiliano Cristiá
Ingeniería de Software 1
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

2020

Resumen

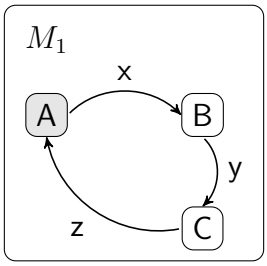
En este documento se presentan los siguientes conceptos y resultados: pasos de ejecución repetitivos (*stuttering steps*), el concepto de estado, el teorema de Alpern-Schneider, seguridad (*safety*), vitalidad (*liveness*), máquina cerrada (*closed machine*), equidad (*fairness*) y el teorema de Abadi-Lamport sobre máquinas cerradas.

El contenido de este documento está parcialmente basado en los siguientes trabajos [1, 2, 3, 4, 5].

1. Pasos de ejecución repetitivos

El concepto de pasos de ejecución repetitivos[1] (*stuttering steps*) está relacionado con la idea de dar la especificación de un sistema como la composición de las especificaciones de sub-sistemas o componentes.

Consideremos el sistema descripto por la siguiente máquina de estados (donde el estado en gris es el estado inicial).



Definimos la semántica de este sistema como el conjunto de todas las sucesiones infinitas de estados que surgen de ejecutar las transiciones. Si el sistema permanece infinito tiempo en un estado (porque la transición de salida no se da nunca) este se repite al infinito en la sucesión. A una sucesión infinita de estados la llamaremos *ejecución* o *comportamiento*.

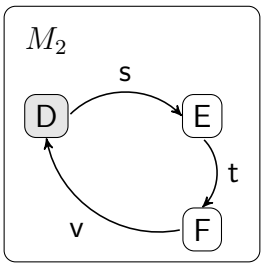
Por consiguiente, las siguientes son algunas de las ejecuciones de M_1 (donde $\widehat{\cdot}$ significa que $\cdot$ se

[1] En versiones anteriores de este apunte usaba el término 'pasos de ejecución intrascendentes'.

repite al infinito):

$$\langle \widehat{\mathsf{A}} \rangle \tag{1}$$
$$\langle \mathsf{A}, \widehat{\mathsf{B}} \rangle \tag{2}$$
$$\langle \mathsf{A}, \mathsf{B}, \widehat{\mathsf{C}} \rangle \tag{3}$$
$$\langle \mathsf{A}, \mathsf{B}, \mathsf{C}, \widehat{\mathsf{A}} \rangle \tag{4}$$
$$\langle \mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{A}, \widehat{\mathsf{B}} \rangle \tag{5}$$

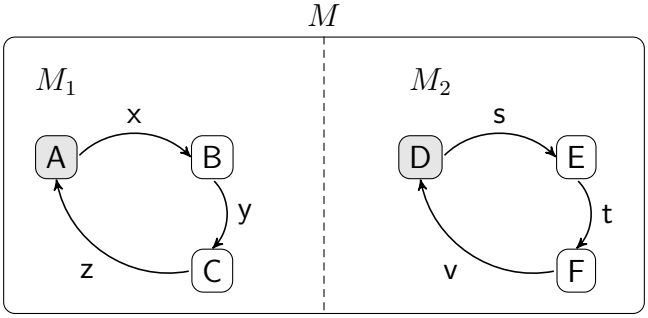
Ahora consideremos el sistema descripto por M_2:



algunas de cuyas ejecuciones son:

$$\langle \widehat{\mathsf{D}} \rangle \tag{6}$$
$$\langle \mathsf{D}, \widehat{\mathsf{E}} \rangle \tag{7}$$
$$\langle \mathsf{D}, \mathsf{E}, \widehat{\mathsf{F}} \rangle \tag{8}$$
$$\langle \mathsf{D}, \mathsf{E}, \mathsf{F}, \widehat{\mathsf{D}} \rangle \tag{9}$$
$$\langle \mathsf{D}, \mathsf{E}, \mathsf{F}, \mathsf{D}, \widehat{\mathsf{E}} \rangle \tag{10}$$

Finalmente, consideremos el sistema M que resulta de componer en paralelo M_1 y M_2.



algunas de cuyas ejecuciones son:

$$\langle \widehat{(\mathsf{A}, \mathsf{D})} \rangle \tag{11}$$
$$\langle (\mathsf{A}, \mathsf{D}), \widehat{(\mathsf{B}, \mathsf{D})} \rangle \tag{12}$$
$$\langle (\mathsf{A}, \mathsf{D}), (\mathsf{B}, \mathsf{D}), \widehat{(\mathsf{B}, \mathsf{E})} \rangle \tag{13}$$
$$\langle (\mathsf{A}, \mathsf{D}), (\mathsf{B}, \mathsf{D}), (\mathsf{B}, \mathsf{E}), \widehat{(\mathsf{C}, \mathsf{E})} \rangle \tag{14}$$

Consideremos ahora la proyección de la ejecución (14) a M_1:

$$\mathcal{P}_{M_1}(\langle (\mathsf{A}, \mathsf{D}), (\mathsf{B}, \mathsf{D}), (\mathsf{B}, \mathsf{E}), \widehat{(\mathsf{C}, \mathsf{E})} \rangle) = \langle \mathsf{A}, \mathsf{B}, \mathsf{B}, \widehat{\mathsf{C}} \rangle \tag{15}$$

El problema con (15) es que no es una de las ejecuciones admitidas por M_1 puesto que B se repite un número finito de veces en el medio de la ejecución (mientras que en todas las ejecuciones de M_1, si B se repite, lo hace infinitas veces y al final de la ejecución). Por lo tanto, el comportamiento de M contempla ejecuciones cuyas proyecciones resultan en ejecuciones que el componente correspondiente no contempla. En otras palabras, si tratamos de descubrir los componentes que componen M mirando sus ejecuciones, no llegaremos a las especificaciones de M_1 y M_2.

Los pasos repetitivos (*stuttering steps*) permiten evitar esta inconsistencia. Entonces, definimos una ejecución de un cierto sistema M como el conjunto de sucesiones infinitas de estados que surgen de ejecutar las transiciones pero donde se admiten *repeticiones finitas de estados* llamadas *pasos repetitivos*. O sea que para la máquina M_1 las ejecuciones (1)-(5) son (donde $\cdot^*$ significa que $\cdot$ puede repetirse un número finito de veces):

$$\langle \widehat{\mathsf{A}} \rangle \tag{16}$$

$$\langle \mathsf{A}^*, \widehat{\mathsf{B}} \rangle \tag{17}$$

$$\langle \mathsf{A}^*, \mathsf{B}^*, \widehat{\mathsf{C}} \rangle \tag{18}$$

$$\langle \mathsf{A}^*, \mathsf{B}^*, \mathsf{C}^*, \widehat{\mathsf{A}} \rangle \tag{19}$$

$$\langle \mathsf{A}^*, \mathsf{B}^*, \mathsf{C}^*, \mathsf{A}^*, \widehat{\mathsf{B}} \rangle \tag{20}$$

aunque normalmente los pasos repetitivos no se explicitan sino que se utiliza la notación usada en (1)-(5) donde se asume que cada estado puede repetirse un número finito de veces.

2. El concepto de estado

Supongamos que el sistema M_1 depende de la variable de estado x que toma valores en $\mathbb{N}$. La única transición permitida en M_1 es $T_{M_1} \triangleq x' = x + 1$ y el estado inicial de M_1 es $x = 0$. Entonces las ejecuciones de M_1 pueden caracterizarse de la siguiente forma:

$$\langle x = 0, x = 1, x = 2, \dots \rangle$$

Ahora supongamos que tenemos otro sistema M_2 que depende de la variable de estado y que toma valores en $\mathbb{N}$. La única transición permitida en M_2 es $T_{M_2} \triangleq y' = y + 2$ y el estado inicial de M_2 es $y = 0$. Entonces las ejecuciones de M_2 pueden caracterizarse de la siguiente forma:

$$\langle y = 0, y = 2, y = 4, \dots \rangle$$

Finalmente supongamos que el sistema M es el resultado de componer M_1 con M_2 de forma tal que la única transición permitida es $T_M \triangleq T_{M_1} \vee T_{M_2}$. Entonces una de las ejecuciones típicas de M puede caracterizarse de la siguiente forma:

$$\langle (x = 0, y = 0), (x = 1, y = -5), (x = 10, y = -3), \dots \rangle$$

puesto que entre $(x = 0, y = 0)$ y $(x = 1, y = -5)$ se ejecutó T_{M_1} la cual establece el valor de x pero no establece el valor de y por lo que el paso es perfectamente compatible con T_M; y entre $(x = 1, y = -5)$ y $(x = 10, y = -3)$ se ejecutó T_{M_2} la cual establece el valor de y pero no establece el valor de x por lo que el paso es perfectamente compatible con T_M. Claramente este tipo de comportamientos no son los que teníamos en mente cuando dimos la especificación de T_M.

La especificación de T_M que en realidad queremos es esta:

$$T_M \triangleq T_{M_1} \wedge y' = y \vee T_{M_2} \wedge x' = x$$

pero el problema es que ni $T_{M_1} \wedge y' = y$ es una especificación para la máquina M_1 ni $T_{M_2} \wedge x' = x$ lo es para M_2 dado que M_1 no predica sobre y ni M_2 sobre x.

Por este motivo Lamport define el concepto de estado de la siguiente forma. Sea *Var* el conjunto de todos los nombres posibles de variables y sea *Val* la colección de todos los valores que las variables pueden tomar (notar que algunos elementos de *Val* son 1, $(2, 3)$, $\{4, a, 6\}$, $\mathbb{N}$, etc.). Entonces el estado s se define como una función de *Var* en *Val*:

$$s : Var \to Val$$

Es decir, que si s es un estado del sistema M_1 o de M_2 o de M, tanto $s(x)$ como $s(y)$ están definidas. Por lo tanto, si $e \triangleq \langle e(0), e(1), e(2), \dots \rangle$ es una ejecución, lo es de M_1 sí y solo sí:

$$e(0)(x) = 0 \wedge e(1)(x) = 1 \wedge e(2)(x) = 2 \wedge \dots \tag{21}$$

y lo es de M_2 sí y solo sí:

$$e(0)(y) = 0 \wedge e(1)(y) = 2 \wedge e(2)(y) = 4 \wedge \dots \tag{22}$$

Definir el concepto de estado de esta forma surge de la necesidad de poder especificar un sistema como la composición de las especificaciones de sub-sistemas o componentes. De esta forma, cualquier estado depende potencialmente de todas las variables posibles solo que cada componente especifica ciertas condiciones para algunas de las variables.

Podemos decir que al definir el concepto de estado de esta forma estamos definiendo *el estado de un universo discreto.*

3. El teorema de Alpern-Schneider

De aquí en más consideraremos que una *propiedad* de un sistema concurrente es un conjunto de ejecuciones. De la misma forma, como ya vimos, un sistema concurrente queda caracterizado por el conjunto de ejecuciones que admite. En consecuencia decimos que un sistema M verifica una propiedad P sí y solo sí $M \subseteq P$ (es decir, si el conjunto de ejecuciones que caracteriza a M está incluido o es igual al conjunto de ejecuciones que define a la propiedad P).

Las siguientes son algunos ejemplos de propiedades que se pueden expresar como conjuntos de ejecuciones: ausencia de abrazo mortal (*deadlock freedom*), corrección parcial (si un programa termina, verifica su especificación[2]), exclusión mutua, primero en llegar primero en ser atendido, ausencia de inanición (*starvation freedom*), terminación (o sea parte de la corrección total) y garantía de servicio.

En 1985 Bruce Alpern y Fred Schneider, en su artículo[3] "Defining Liveness", postulan y demuestran el siguiente teorema.

Teorema 1 *Toda propiedad P es el resultado de la intersección de una propiedad de seguridad (*safety*) con una propiedad de vitalidad (*liveness*).*

Como los sistemas también están caracterizados como conjuntos de ejecuciones, el teorema implica que todo sistema puede expresarse como la intersección de una propiedad de seguridad (*safety*) con una propiedad de vitalidad (*liveness*). Por lo tanto, a continuación veremos con cierto detalle las propiedades de seguridad y vitalidad.

[2]Corrección total implica, además, que el programa termina.

[3]`https://www.cs.cornell.edu/fbs/publications/DefLiveness.pdf`

3.1. Seguridad

Las propiedades de seguridad[4] postulan que *nada malo puede pasar* durante una ejecución del sistema. Toda propiedad de seguridad *proscribe* o *prohíbe* algo 'malo'. Algunos ejemplos de propiedades de seguridad son: ausencia de abrazo mortal, corrección parcial, exclusión mutua y primero en llegar primero en ser atendido. Entones:

- En exclusión mutua lo 'malo' es dos procesos ejecutando en la sección crítica al mismo tiempo;
- En ausencia de abrazo mortal lo 'malo' es que haya abrazo mortal;
- En corrección parcial lo 'malo' es que el programa termine en un estado que no satisface la post-condición habiendo comenzado a ejecutar en un estado que satisfacía la pre-condición;
- En primero en llegar primero en ser atendido, lo 'malo' es atender un pedido antes que otro que llegó primero.

Usualmente la forma más fácil de dar una propiedad de seguridad es indicando solo las transiciones de estado que están permitidas. O sea que una propiedad de seguridad normalmente se da indicando solo los comportamientos permitidos y de esta forma, implícitamente, se prohíben todos los demás. En este sentido, una propiedad de seguridad *no* indica que el sistema hará algo; solo indica lo que tiene *permitido* hacer. Hasta el momento, todo lo que hemos hecho en este curso es especificar propiedades de seguridad sin saberlo (por ejemplo, una especificación Z o un modelo Statecharts, normalmente, indican *solo* la propiedad de seguridad que se espera verifique el sistema que se está especificando). Lamport dice que normalmente el 90 % del esfuerzo de especificación está dedicado a propiedades de seguridad.

Los siguientes son algunos ejemplos de propiedades de seguridad tomados del cruce ferroviario visto en Statecharts:

- Un tren puede ingresar a la zona de peligro solo cuando la vía (en dicha zona) está vacía (EC-U-DK).
- El sensor Y se activa solo luego de que se haya activado el sensor Z correspondiente (EC-S-DK).
- La alarma se enciende y se apaga de forma alternada (MC-S-DK).
- La barrera no sube mientras pasa el tren (MC-S-R).

El primero en formalizar el concepto de seguridad fue Lamport. Sin embargo, antes de dar la definición formal de seguridad introducimos la siguiente notación.

Notación Sea E un conjunto de estados. E^* denota el conjunto de todas las sucesiones finitas de elementos de E (llamadas *ejecuciones parciales*), y E^∞ el de todas las sucesiones infinitas (o sea las *ejecuciones*). Si $e \in E^\infty$ y $n \in \mathbb{N}$, entonces e_n denota el prefijo de e hasta n y $e(n)$ el n-ésimo elemento de la sucesión (notar que el primer elemento es $e(0)$). Es decir, $e_n = \langle e(0), e(1), \ldots, e(n)\rangle$. Si e y t son dos sucesiones de estados ($e \in E^*$), $e \circ t$ denota la concatenación de e con t.

Definición 1 *S es una propiedad de seguridad respecto de E sí y solo sí se verifica lo siguiente:*

$$e \notin S \iff \exists n \in \mathbb{N} : \forall t \in E^\infty : e_n \circ t \notin S$$

para toda $e \in E^\infty$.

[4]Como habrán observado, traducimos *safety* como *seguridad*. Notar que *security* también se traduce como *seguridad*. El problema con esta traducción es que *security* y *safety* son dos conceptos técnicamente muy diferentes. Por lo tanto, la traducción puede resultar confusa y dar lugar a errores si no se aclara con cierta precisión el contexto en el cual se la usa. En todo este apunte *seguridad* refiere a *safety*.

Observar que esta definición define una propiedad de seguridad por las ejecuciones que *no* pertenecen a ella. Es decir, la propiedad de seguridad es el complemento (respecto de E^∞) de las ejecuciones que verifican la Definición 1. Precisamente, e no pertenece a S cuando en un punto (n) lo 'malo' prohibido por S tuvo lugar y en consecuencia esto no se puede remediar (lo 'malo' ocurrió y ya no se puede volver atrás). En otras palabras, haga lo que haga el sistema luego de n (simbolizado por t) no podrá remediar el hecho de que lo 'malo' ya ocurrió en e. Por ejemplo, si S es "primero en llegar primero en ser atendido" y en e hay un pedido que es servido primero que otro que llegó antes, entonces hay un punto en e donde esto ocurrió y cualquier cosa que siga en e no podrá ocultar este hecho, por lo tanto $e \notin S$.

A raíz de la existencia de un punto preciso a partir del cual la ejecución no cumple con S, Alpern y Schneider dicen que las propiedades de seguridad son *discretas* en el sentido de que hay un punto preciso donde se puede ver la violación de la propiedad. Además, de la Definición 1 surge que una propiedad de seguridad jamás puede estipular que algo eventualmente ocurrirá, solo que algo nunca ocurrirá.

3.2. Vitalidad

Las propiedades de vitalidad estipulan que algo 'bueno' ocurrirá durante la ejecución del sistema. Toda propiedad de vitalidad asegura que algo eventualmente ocurrirá en el sistema. Algunos ejemplos de propiedades de vitalidad son: ausencia de inanición, terminación y garantía de servicio. Entonces:

- En ausencia de inanición (que establece que un proceso progresa infinitamente a menudo) lo 'bueno' es que el proceso progresa;
- En terminación lo 'bueno' es ejecutar la última instrucción;
- En garantía de servicio lo 'bueno' es que el proceso es atendido.

Una propiedad de vitalidad dice que ninguna ejecución parcial es irremediable: siempre existe la posibilidad de que más adelante la ejecución cumpla con la propiedad. Es decir que una propiedad de vitalidad indica los estados que obligatoriamente deberán alcanzarse durante una ejecución.

Los siguientes son algunos ejemplos de propiedades de vitalidad tomados del cruce ferroviario visto en Statecharts:

- Un tren que entra en la zona de peligro eventualmente la abandona (EC-U-DK).
- Llega un momento en que ha pasado demasiado tiempo desde la última señal del sensor Y (EC-S-DK).
- Una vez que el tren pasó se debe levantar la barrera (MC-S-R).

Jackson señala que las propiedades de vitalidad del tipo MC-S-DK no parecen ser posibles en ningún sistema (aunque sí son posibles las de seguridad de este tipo).

Ejercicio 1 *Piense en una justificación para esta conjetura de Jackson.*

A continuación formalizamos el concepto de vitalidad.

Definición 2 *L es una propiedad de vitalidad respecto de E sí y solo sí se verifica lo siguiente:*

$$\forall e \in E^* : \exists t \in E^\infty : e \circ t \in L$$

Es decir que L es una propiedad de vitalidad si toda ejecución parcial puede ser extendida a una ejecución donde lo 'bueno' estipulado por L ocurre. En otras palabras, un sistema verifica L de vitalidad si toda ejecución parcial del sistema puede ser extendida a una ejecución donde lo estipulado por L se da.

Notar que al contrario de las propiedades de seguridad, no necesariamente hay un punto preciso donde la propiedad de vitalidad se haga verdadera. Más aun, ciertas propiedades de vitalidad requieren que muchos estados la verifiquen (por ejemplo en ausencia de inanición el proceso debe avanzar infinitamente a menudo por lo que debe haber infinitos estados donde cierta condición se cumple). Es decir que las propiedades de vitalidad no son discretas según la idea comentada en la sección anterior. En este sentido 'bueno' y 'malo' parecen ser conceptos fundamentalmente diferentes. Además, una propiedad de vitalidad no estipula que algo 'bueno' *siempre* ocurre, solo indica que eso eventualmente ocurrirá.

En general hay un acuerdo universal sobre la definición de seguridad; no así con la definición de vitalidad (por lo menos al momento en que Alpern y Schneider publicaron su famoso artículo). Entonces cabe preguntarse, ¿es razonable la definición de vitalidad? Supongamos que L es de vitalidad pero no verifica la Definición 2. Entonces existe una ejecución parcial e tal que:

$$\forall t \in E^{\infty} : e \circ t \notin L$$

lo cual sugiere que e es algo prohibido por L por lo que L es, al menos en parte, de seguridad (comparar con la Definición 1). Es decir que, al relajar la Definición 2 se obtienen propiedades que prohíben cosas lo que es el dominio de las propiedades de seguridad. Por lo tanto, parecería darse que si una propiedad no verifica la definición dada es en parte de seguridad. Entonces se considera que la Definición 2 incluye a todas las propiedades (puramente) de vitalidad.

3.3. El concepto de máquina cerrada

Según el teorema de Alpern-Schneider la especificación de un sistema debería estar dada por una propiedad de seguridad y otra de vitalidad, donde la propiedad de seguridad restringe el comportamiento del sistema a aquellas ejecuciones que son seguras. Sin embargo, consideremos el siguiente sistema.

1. El estado del sistema está dado por la variable x que toma valores en $\mathbb{N}$.
2. El estado inicial es $x = 0$.
3. Propiedad de seguridad: la única transición posible es $x' = x + 1$,
4. Propiedad de vitalidad: si en algún estado x vale 1 entonces eventualmente se debe alcanzar un estado donde x valga 0.

Por consiguiente, la única ejecución posible de este sistema es $\langle \widehat{x = 0} \rangle$. Es decir, el sistema permanecerá siempre en el estado inicial puesto que si la transición permitida se ejecutara, x valdría 1 pero no habría forma de cumplir con la propiedad de vitalidad. Entonces la única forma de cumplir con las dos propiedades es permaneciendo en el estado inicial.

Este ejemplo demuestra que no necesariamente todas las restricciones están dadas por la propiedad de seguridad. Según Abadi y Lamport este problema se origina al permitir propiedades de vitalidad arbitrarias. Por lo tanto ellos estudiaron el problema de encontrar propiedades de vitalidad que no agregaran restricciones al sistema que no estuvieran dadas en su propiedad de seguridad. El primer paso fue definir el concepto de *máquina cerrada* para el cual es preciso definir primero la *clausura* de una propiedad.

Definición 3 *Si A es una propiedad cualquiera, $\mathcal{C}(A)$, denominada clausura de A, es la menor propiedad de seguridad que contiene a A.*

Definición 4 *Si S es una propiedad de seguridad y L es una propiedad cualquiera, entonces el par (S, L) es una máquina cerrada sí y solo sí $\mathcal{C}(S \cap L) = S$.*

Observar que si (S, L) es máquina cerrada todas las restricciones del sistema definido por $S \cap L$ están dadas por S ya que no existe S' de seguridad tal que $S \cap L \subseteq S' \subset S$. Si tal S' existiera querría decir que L agrega restricciones.

Abadi y Lamport demostraron que si las propiedades de vitalidad se escriben como propiedades de equidad entonces se obtienen máquinas cerradas.

4. Propiedades de equidad

Las propiedades de equidad (*fairness*) son una clase de propiedades de vitalidad que tienen una forma muy particular. Para definir formalmente las propiedades de equidad necesitamos introducir algunos conceptos simples de lógica temporal.

Sea P un predicado de estado (es decir un predicado que depende de las variables de estado), entonces $\Box P$ (que se lee 'siempre P') es un predicado de la lógica temporal que depende de una ejecución:

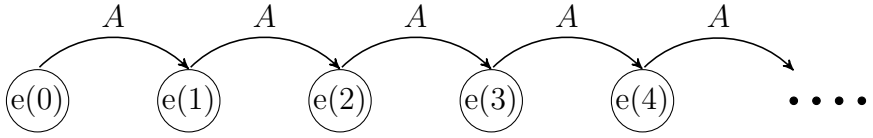
$$\Box P \text{ es verdadero en } e \in E^\infty \iff \forall i \in \mathbb{N} : P(e(i))$$

Esta es una forma temporal de expresar que P es un invariante de estado en la ejecución e.

Sea A un predicado que depende de dos estados (es decir A es una transición de estados), entonces $\Box A$ es un predicado de la lógica temporal que depende de una ejecución:

$$\Box A \text{ es verdadero en } e \in E^\infty \iff \forall i \in \mathbb{N} : A(e(i), e(i+1))$$

En este caso el sistema pasa del estado $e(i)$ al $e(i+1)$ (para cualquier i) siempre debido a A. Cuando el sistema pasa del estado $e(i)$ al $e(i+1)$ debido a A decimos que se da un paso-A (A-*step*). Gráficamente $\Box A$ se puede representar de la siguiente forma:



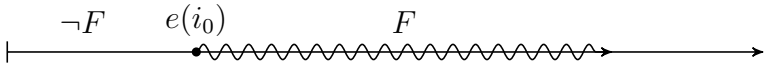
Si F es un predicado cualquiera, $\Diamond F$ se define como $\neg\Box\neg F$ y se lee 'eventualmente F'. Precisamente:

$$\Diamond F \text{ es verdadero en } e \in E^\infty \iff \exists i \in \mathbb{N} : F(e(i))$$

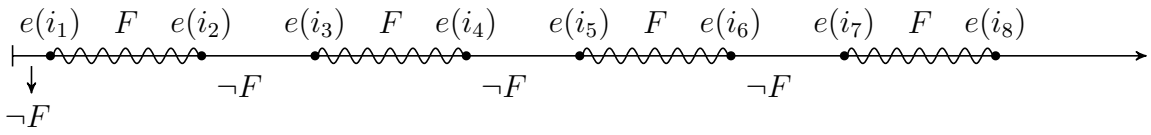
o $F(e(i), e(i+1))$ si F es una transición.

Hay dos combinaciones de $\Box$ y $\Diamond$ que son importantes para definir las propiedades de equidad:

- $\Diamond\Box F$ se lee 'eventualmente siempre F' y es verdadero si existe un estado a partir del cual F es siempre verdadero. Gráficamente:



- $\Box\Diamond F$ se lee 'infinitamente a menudo F' y es verdadero si existen infinitos estados donde F es verdadero (por ejemplo todos los estados con índice par o todos los estados con índice terminado en 0, etc.). Gráficamente se puede expresar de la siguiente forma donde se asume que los sectores donde F vale se repiten al infinito:



Si A es una transición de estados, el predicado $enabled(A)$ es verdadero cuando la precondición de A se satisface (es decir, cuando es posible ejecutar la transición A o cuando A está *habilitada* para ejecutar).

Con todos estos elementos podemos dar la definición de las propiedades de equidad. Existen dos versiones de equidad: débil (*weak fairness*) y fuerte (*strong fairness*).

Definición 5 *La transición de estados A verifica equidad débil sí y solo sí:*

$$WF(A) \triangleq \Diamond\Box enabled(A) \implies \Box\Diamond A$$

Definición 6 *La transición de estados A verifica equidad fuerte sí y solo sí:*

$$SF(A) \triangleq \Box\Diamond enabled(A) \implies \Box\Diamond A$$

Observar que la diferencia entre las dos fórmulas se da en el antecedente: en WF se pide que a partir de cierto punto A esté siempre habilitada mientras que en SF es suficiente con que A esté habilitada en infinitos estados aunque puede no estarlo de forma continua. En cualquiera de los dos casos se exige que A sea ejecutada infinitas veces.

Finalmente enunciamos el teorema de Abadi-Lamport.

Teorema 2 *Si S es una propiedad de seguridad y L es una conjunción numerable de fórmulas $WF(A_i)$ o $SF(A_i)$ tales que cada A_i $(i \in I)$ puede ejecutarse satisfaciendo S, entonces (S, L) es máquina cerrada.*

Una forma de cumplir con la condición "A_i puede ejecutarse satisfaciendo S", es definiendo S como $\bigvee_{i \in I} A_i$. Es decir, S es la disyunción de todas las transiciones permitidas, lo que además implica que S es de seguridad.

Referencias

[1] B. Alpern and F. B. Schneider, "Defining livennes," *Information Processing Letters*, vol. 21, no. 7, pp. 181–185, Oct. 1985.

[2] M. Abadi, B. Alpern, K. R. Apt, N. Francez, S. Katz, L. Lamport, and F. B. Schneider, "Preserving liveness: Comments on "safety and liveness from a methodological point of view"," *Inf. Process. Lett.*, vol. 40, no. 3, pp. 141–142, 1991. [Online]. Available: https://doi.org/10.1016/0020-0190(91)90168-H

[3] L. Lamport, "The temporal logic of actions," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 3, pp. 872–923, 1994. [Online]. Available: http://doi.acm.org/10.1145/177492.177726

[4] M. Abadi and L. Lamport, "Composing specifications," *ACM Trans. Program. Lang. Syst.*, vol. 15, no. 1, pp. 73–132, 1993. [Online]. Available: http://doi.acm.org/10.1145/151646.151649

[5] ——, "The existence of refinement mappings," *Theor. Comput. Sci.*, vol. 82, no. 2, pp. 253–284, 1991. [Online]. Available: https://doi.org/10.1016/0304-3975(91)90224-P