Modelando propiedades de liveness en Event-B

Ramiro Garay

Agosto, 2025

Antes de empezar…

Sobre mí

Figure 1: Ramiro Garay

- Argentino, ex-estudiante de Ingeniería en Informática de la UNL, Santa Fe (2017-2022)

Sobre mí

Figure 1: Ramiro Garay

- Argentino, ex-estudiante de Ingeniería en Informática de la UNL, Santa Fe (2017-2022)
- Desarrollador a medio tiempo para MLabs, una consultora especializada en Blockchain (2022-)

Sobre mí

Figure 1: Ramiro Garay

- Argentino, ex-estudiante de Ingeniería en Informática de la UNL, Santa Fe (2017-2022)
- Desarrollador a medio tiempo para MLabs, una consultora especializada en Blockchain (2022-)
- Estudiante avanzado de Licenciatura en Computación (2025-)

Sobre mí

Figure 1: Ramiro Garay

- Argentino, ex-estudiante de Ingeniería en Informática de la UNL, Santa Fe (2017-2022)
- Desarrollador a medio tiempo para MLabs, una consultora especializada en Blockchain (2022-)
- Estudiante avanzado de Licenciatura en Computación (2025-)
- Actualmente becario PREXI en el LINS

Agenda

1. Intro. propiedades de liveness

Agenda

1. Intro. propiedades de liveness
2. *Lógica Temporal Lineal (LTL)*

Agenda

1. Intro. propiedades de liveness
2. *Lógica Temporal Lineal (LTL)*
3. **Demostración** de propiedades de liveness

Agenda

1. Intro. propiedades de liveness
2. *Lógica Temporal Lineal (LTL)*
3. **Demostración** de propiedades de liveness
4. **Verificación** de propiedades de liveness usando *model checking*

Propiedades de Liveness (Intro)

Una definición informal

"Son aquellas propiedades que nos garantizan que el sistema eventualmente va a hacer **algo**"

Una definición informal

"Son aquellas propiedades que nos garantizan que el sistema eventualmente va a hacer **algo**"

Son fundamentales, ya que permiten representar ciertos comportamientos dinámicos del sistema.

Ejemplo: ascensor (I)

Un ascensor debe cumplir propiedades de safety. Por ejemplo:

- El ascensor **nunca** se mueve con la puerta abierta

Ejemplo: ascensor (I)

Un ascensor debe cumplir propiedades de safety. Por ejemplo:

- El ascensor **nunca** se mueve con la puerta abierta
- El ascensor **nunca** cierra la puerta cuando un usuario es detectado en el umbral

Ejemplo: ascensor (II)

Pero también de liveness!

- Si el usuario pide el ascensor, **eventualmente** el mismo viaja hacia el piso del usuario.

Ejemplo: ascensor (II)

Pero también de liveness!

- Si el usuario pide el ascensor, **eventualmente** el mismo viaja hacia el piso del usuario.
- Si el usuario seleccionó un piso, **eventualmente** el ascensor va a llegar al piso pedido.

Ejemplo: ascensor (III)

Un ascensor sin propiedades de liveness puede ser muy seguro, pero también **inútil**.

Ejemplo: ascensor (III)

Un ascensor sin propiedades de liveness puede ser muy seguro, pero también **inútil**.

Ejemplo: Un ascensor que se mantiene cerrado e inmóvil satisface todas las propiedades de safety pero **ninguna** de liveness.

"Completitud" de propiedades de *safety* y *liveness*

- Usando propiedades de safety y liveness, uno puede especificar **completamente** un sistema. [1]

[1] B. Alpern y F. B. Schneider, «Defining liveness», Information Processing Letters, vol. 21, n.º 4, pp. 181-185, oct. 1985, doi: 10.1016/0020-0190(85)90056-0.

“Completitud” de propiedades de *safety* y *liveness*

- Usando propiedades de safety y liveness, uno puede especificar **completamente** un sistema. [1]
- O al revés: toda propiedad es de safety, liveness o una combinación de ambas.

[1] B. Alpern y F. B. Schneider, «Defining liveness», Information Processing Letters, vol. 21, n.º 4, pp. 181-185, oct. 1985, doi: 10.1016/0020-0190(85)90056-0.

Ejemplos de propiedades de liveness

1. *Starvation freedom*: un proceso hace progreso infinitamente seguido

Ejemplos de propiedades de liveness

1. *Starvation freedom*: un proceso hace progreso infinitamente seguido
2. *Termination*: el proceso finaliza

Ejemplos de propiedades de liveness

1. *Starvation freedom*: un proceso hace progreso infinitamente seguido
2. *Termination*: el proceso finaliza
3. *Guaranteed service*: cada solicitud es satisfecha eventualmente

Un poco de LTL

Hacia una definición formal

Para definir formalmente propiedades de liveness es necesario hablar de *tiempo*.

Hacia una definición formal

Para definir formalmente propiedades de liveness es necesario hablar de *tiempo*.

La Lógica Temporal Lineal (LTL) es una extensión de la *lógica de 1er orden* que incluye **operadores temporales**.

Hacia una definición formal

Para definir formalmente propiedades de liveness es necesario hablar de *tiempo*.

La Lógica Temporal Lineal (LTL) es una extensión de la *lógica de 1er orden* que incluye **operadores temporales**.

Los operadores temporales de LTL nos permite expresar cosas como **siempre**, **después**, **eventualmente**, etc.

Hacia una definición formal

Al igual que la lógica de 1er orden, LTL se puede analizar desde dos puntos de vista:

Hacia una definición formal

Al igual que la lógica de 1er orden, LTL se puede analizar desde dos puntos de vista:

1. La *sintaxis* (cómo se construyen las fórmulas lógicas)

Hacia una definición formal

Al igual que la lógica de 1er orden, LTL se puede analizar desde dos puntos de vista:

1. La *sintaxis* (cómo se construyen las fórmulas lógicas)
2. La *semántica* (cuándo se satisfacen las fórmulas)

Hacia una definición formal

Al igual que la lógica de 1er orden, LTL se puede analizar desde dos puntos de vista:

1. La *sintaxis* (cómo se construyen las fórmulas lógicas)
2. La *semántica* (cuándo se satisfacen las fórmulas)

Hacia una definición formal

Al igual que la lógica de 1er orden, LTL se puede analizar desde dos puntos de vista:

1. La *sintaxis* (cómo se construyen las fórmulas lógicas)
2. La *semántica* (cuándo se satisfacen las fórmulas)

Empezamos por la semántica.

Estructura de Kripke

La semántica de LTL requiere de una estructura auxiliar llamada **estructura de Kripke**.

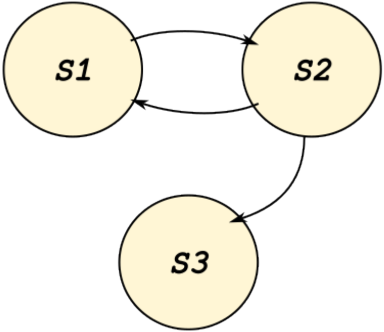


Figure 2: Máquina de estados

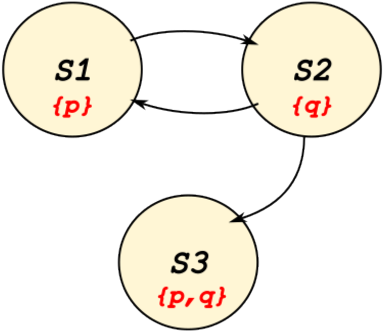


Figure 3: Estructura de Kripke

Donde $p(s)$ y $q(s)$ son predicados definidos sobre el estado de la máquina

Traza de una máquina

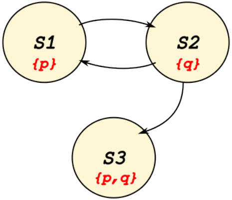
Una máquina de estados tiene asociada un conjunto de **trazas**.

Una traza es una sucesión (posiblemente infinita!) de estados. Estos se obtienen de ejecutar la máquina **respetando las restricciones**.

Traza de una máquina

Una máquina de estados tiene asociada un conjunto de **trazas**.

Una traza es una sucesión (posiblemente infinita!) de estados. Estos se obtienen de ejecutar la máquina **respetando las restricciones**.

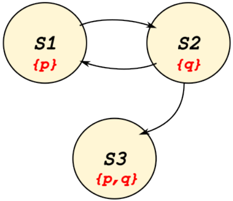


- $\Omega = s_1, s_2, s_1, s_2, s_1, s_2 ...$

Traza de una máquina

Una máquina de estados tiene asociada un conjunto de **trazas**.

Una traza es una sucesión (posiblemente infinita!) de estados. Estos se obtienen de ejecutar la máquina **respetando las restricciones**.



- $\Omega = s_1, s_2, s_1, s_2, s_1, s_2 ...$
- $\Gamma = s_1, s_2, s_1, s_2, s_3$

Traza de una máquina

Por lo tanto, la estructura de Kripke también tiene una traza.

Traza de una máquina

Por lo tanto, la estructura de Kripke también tiene una traza.

Pero además, como tenemos proposiciones, podemos hablar de cómo las proposiciones de nuestra estructura cambian con el tiempo.

Traza de una máquina

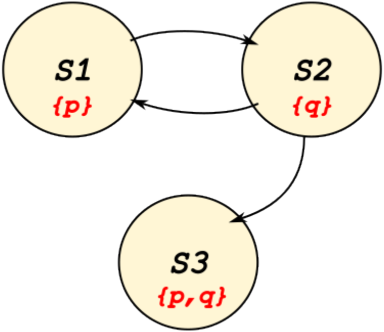
Por lo tanto, la estructura de Kripke también tiene una traza.

Pero además, como tenemos proposiciones, podemos hablar de cómo las proposiciones de nuestra estructura cambian con el tiempo.

Esto es muy **potente**.

Estructura de Kripke (III)

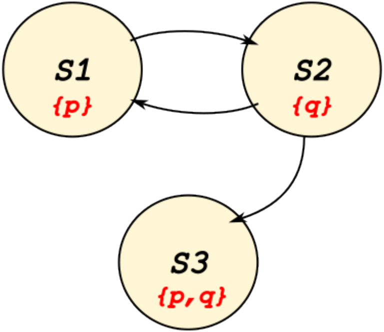
La estructura de Kripke tiene asociada un conjunto de **trazas**. Por ejemplo:



- $\Omega = s_1, s_2, s_1, s_2, s_1, s_2 ...$

Estructura de Kripke (III)

La estructura de Kripke tiene asociada un conjunto de **trazas**. Por ejemplo:



- $\Omega = s_1, s_2, s_1, s_2, s_1, s_2 ...$
- Una *traza* es una sucesión de estados generada por la maquina **respetando** las restricciones de la misma.

LTL: Ejemplos (I)

Las trazas pueden satisfacer o no **una fórmula LTL**

$$(\Omega \vdash \phi)$$

LTL: Ejemplos (I)

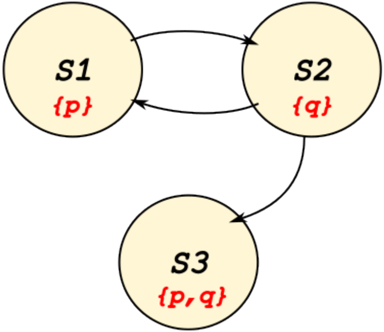
Las trazas pueden satisfacer o no **una fórmula LTL**

$$(\Omega \vdash \phi)$$

Analizamos la traza:

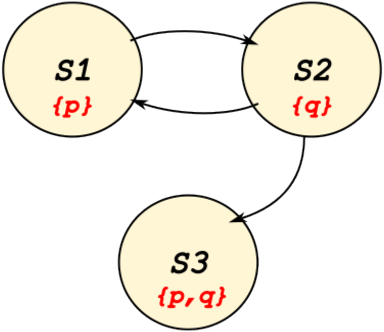
$$\Omega = s_1, s_2, s_1, s_2, s_1, s_2 ...$$

LTL: Ejemplos (II)



- *Variables lógicas / proposiciones con conectivas lógicas*

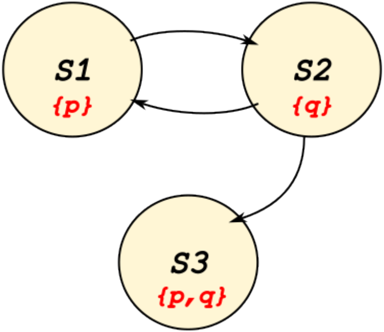
LTL: Ejemplos (II)



- *Variables lógicas / proposiciones con conectivas lógicas*
  -

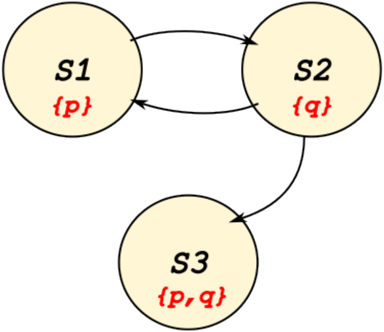
$$\Omega \vdash p$$

LTL: Ejemplos (II)



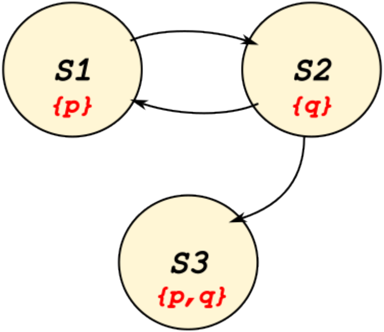
- *Variables lógicas / proposiciones con conectivas lógicas*
  - $$\Omega \vdash p$$
  - $$\Omega \vdash \neg q$$

LTL: Ejemplos (II)



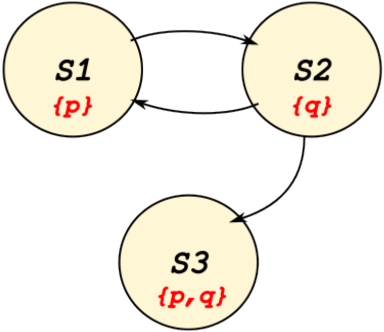
- *Variables lógicas / proposiciones con conectivas lógicas*
  -
    $$\Omega \vdash p$$
  -
    $$\Omega \vdash \neg q$$
- ***Operadores temporales***

LTL: Ejemplos (II)



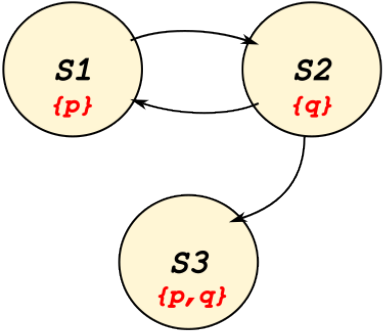
- *Variables lógicas / proposiciones con conectivas lógicas*
  - $$\Omega \vdash p$$
  - $$\Omega \vdash \neg q$$
- ***Operadores temporales***
  - $$\Omega \vdash \circ q$$

LTL: Ejemplos (II)



- *Variables lógicas / proposiciones con conectivas lógicas*
  - $$\Omega \vdash p$$
  - $$\Omega \vdash \neg q$$
- ***Operadores temporales***
  - $$\Omega \vdash \circ q$$
  - $$\Omega \vdash \Box(p \vee q)$$

LTL: Ejemplos (II)

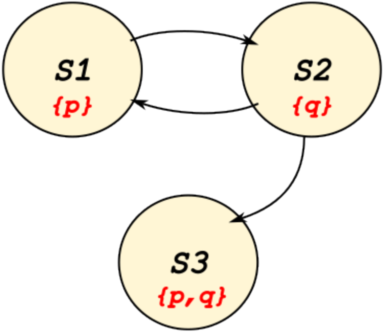


- *Variables lógicas / proposiciones con conectivas lógicas*
  - $$\Omega \vdash p$$
  - $$\Omega \vdash \neg q$$
- ***Operadores temporales***
  - $$\Omega \vdash \circ q$$
  - $$\Omega \vdash \Box(p \lor q)$$
  - ...

LTL: Ejemplos (III)

Otro ejemplo (**finito**):

$$\Gamma = s_1, s_2, s_1, s_2, s_3$$



Γ satisface las mismas propiedades que antes, pero también:

- $$\Gamma \vdash \Diamond(p \wedge q)$$

Propiedades de Liveness (en LTL)

Existencia de P (Definición)

"***Siempre*** *es cierto que,* ***eventualmente*** *P es verdadero*"

Existencia de P (Definición)

"***Siempre** es cierto que, **eventualmente** P es verdadero*"

En LTL:

$\Box\Diamond P$

Existencia de P (Demostración)

Por medio de dos propiedades auxiliares:

1. **Convergencia en $\neg P$**
2. $\neg P$ **es libre de deadlocks**

Existencia de P (Demostración)

Por medio de dos propiedades auxiliares:

1. **Convergencia en** $\neg P$
2. $\neg P$ **es libre de deadlocks**

$$\frac{\begin{array}{l} M \vdash \downarrow \neg P \\ M \vdash \circlearrowleft \neg P \end{array}}{M \vdash \Box \Diamond P} \quad \mathbf{LIVE}_{\Box \Diamond}$$

Persistencia de P (Definición)

"***Eventualmente*** *es cierto que,* ***siempre*** *P es verdadero*"

Persistencia de P (Definición)

"***Eventualmente*** *es cierto que,* ***siempre*** *P es verdadero*"

En LTL:

$\Diamond\Box P$

Persistencia de P (Demostración)

Por medio de dos propiedades auxiliares:

1. **Divergencia en** P
2. $\neg P$ **es libre de deadlocks**

Persistencia de P (Demostración)

Por medio de dos propiedades auxiliares:

1. **Divergencia en** P
2. $\neg P$ **es libre de deadlocks**

$$\frac{\begin{array}{l} \mathsf{M} \vdash \nearrow P \\ \mathsf{M} \vdash \circlearrowleft \neg P \end{array}}{\mathsf{M} \vdash \Diamond \Box P}\ \mathbf{LIVE}_{\Diamond\Box}$$

Progreso de P_1 a P_2 (Definición)

"***Siempre*** *es cierto que, si* P_1 *es verdadero,* ***eventualmente*** P_2 *lo va a ser*"

Progreso de P_1 a P_2 (Definición)

"***Siempre*** *es cierto que, si* P_1 *es verdadero,* ***eventualmente*** P_2 *lo va a ser*"

En LTL:

$\Box(P_1 \implies \Diamond P_2)$

Progreso de P_1 a P_2 (Demostración)

Por medio de varias propiedades auxiliares (no tan simples).

$$\frac{\begin{array}{l}\mathsf{M} \vdash \Box(P_1 \land \neg P_2 \Rightarrow P_3) \\ \mathsf{M} \vdash \underline{\Box(P_3 \Rightarrow (P_3\, \mathcal{U}\, P_2))}\end{array}}{\mathsf{M} \vdash \Box(P_1 \Rightarrow \Diamond\, P_2)} \;\; \mathbf{LIVE_{progress}}$$

$$\frac{\begin{array}{l}\mathsf{M} \vdash (P_1 \land \neg P_2) \curvearrowright (P_1 \lor P_2) \\ \mathsf{M} \vdash \Box\Diamond(\neg P_1 \lor P_2)\end{array}}{\mathsf{M} \vdash \underline{\Box(P_1 \Rightarrow (P_1\, \mathcal{U}\, P_2))}} \;\; \mathbf{Until}$$

Propiedades auxiliares

Convergencia de un evento (I)

Esta propiedad de liveness **sí** se puede representar en el lenguaje de Event-B.

“*Si un evento convergente está activado, entonces eventualmente va a dejar de estarlo*”

Para marcar un evento como convergente, se lo marca con la palabra clave `convergent`.

Adicionalmente, al modelo se le debe agregar una *variante*.

Convergencia de un evento (II)

La variante es un número que satisface las siguientes condiciones:

1. Cuando el evento está activo, **la variante es un número natural**.
2. Cuando el evento se ejecuta, **la variante disminuye**.

Convergencia de un evento (II)

La variante es un número que satisface las siguientes condiciones:

1. Cuando el evento está activo, **la variante es un número natural**.
2. Cuando el evento se ejecuta, **la variante disminuye**.

Intuitivamente, esto implica que cuando la variante deje de ser natural, el evento **ya no va a estar activo** (*Modus tollens* en proposición 1).

Convergencia de un evento (II)

La variante es un número que satisface las siguientes condiciones:

1. Cuando el evento está activo, **la variante es un número natural**.
2. Cuando el evento se ejecuta, **la variante disminuye**.

Intuitivamente, esto implica que cuando la variante deje de ser natural, el evento **ya no va a estar activo** (*Modus tollens* en proposición 1).

Así mísmo, la variantes **debe** dejar de ser natural, ya que el evento disminuye la variante con cada ejecución.

Convergencia de un evento (III)

Cuando varios eventos son convergentes, la elección de la variante se complica.

¿Por qué? Porque Event-B permite **solo una variante por modelo**.

La solución para este problema es combinar las variantes de cada evento convergente en una única *variante lexicográfica*.

Convergencia de un evento (III)

Cuando varios eventos son convergentes, la elección de la variante se complica.

¿Por qué? Porque Event-B permite **solo una variante por modelo**.

La solución para este problema es combinar las variantes de cada evento convergente en una única *variante lexicográfica*.

(TODO: mostrar Variante en modelo PingPongEnd)

Convergencia en P

TODO

Divergencia

TODO

Transiciona de P_1 a P_2

TODO

Verificación en ProB

“Model check” (I)

Esta funcionalidad explora lo máximo posible el espacio de estados del modelo para encontrar *violaciones de invariantes/teoremas* y *deadlocks*.

Es útil para verificar que el modelo cumple con las variantes **antes de demostrarlo**.

(TODO: Mostrar espacio de estados generado por ProB)

“Model check” (II)

Hay dos casos donde el model check no es exhaustivo:

- **No se exploró el espacio de estados completo**
- **No se exploraron todos los eventos posibles**

Ambos se pueden remediar aumentando los valores de las constantes `MAX_INITIALIZATIONS` y `MAX_OPERATIONS` y **acotando las constantes del modelo** (fundamental).

“Model check” (III)

El *model checking* nos permite **sólo verificar**, no demostrar.

En el mejor de los casos (cuando el chequeo es exhaustivo), nos permite **demostrar** las propiedades deseadas en un modelo más pequeño que el “real”.

LTL checking (I)

Esta funcionalidad nos permite escribir fórmulas LTL que son verificadas por ProB.

ProB soporta todos los operadores temporales e incluso algunos operadores específico a B/Event-B que facilitan la escritura de propiedades útiles.

LTL checking (II)

Las propiedades de liveness se pueden escribir del siguiente modo:

```
G F ({is_pinging = 1}))
G F ({is_pinging = 0}))
F G ({runs_counter = RUNS_LIMIT})
G (e(ping) => F (e(pong)))
```

Donde `e(<evento>)` es la *guarda del evento en cuestión* (i.e: el evento está activado).

Bibliografía

Bibliografía