Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Ejercicios TLA

Ingeniería de Software I

Autores:
Arroyo, Joaquín
Bolzan, Francisco

7 de agosto de 2023

Índice

NOTA: Este pdf puede contener errores, no fue revisado a fondo, tenerlo en cuenta si va a ser utilizado. Las soluciones se realizaron desde un conocimiento prematuro sobre TLA y temas implicados, hasta con un conocimiento mas maduro, por esto se pueden encontrar errores y contradicciones entre ejercicios (Más que nada ejercicios de práctica).

1. Teoría

1.1. Ejercicio 1

Explique y ejemplifique las condiciones bajo las cuales equidad débil no es equivalente a equidad fuerte.

$WF(A)$ y $SF(A)$ no son equivalentes cuando existe algún evento externo tal que puede deshabilitar a A.

Dada una secuencia $VARIABLE\ q \in Seq\ VALOR$ y un evento $Add(p, v)$ (El proceso p agrega el valor v a q), donde:

$Add(p, v) \triangleq len(q) < N \land ...$

$WF(Add)$ y $SF(Add)$ no son equivalentes dado que para cualquier proceso p y valor v tal que se habilite $Add(p, v)$ en un momento dado este puede o bien deshabilitarse a si mismo, o bien ser deshabilitado por otro proceso $p' \neq p$ el cuál causa que q se quede sin espacio.

1.2. Ejercicio 2

Explique brevemente las ventajas de tener un lenguaje de especificación no tipado.

- Permite una mayor flexibilidad en la definición de variables y expresiones, lo que facilita la representación de modelos complejos sin la rigidez de los sistemas tipados.
- Al no preocuparse por la declaración de tipos, se puede centrarse en describir la lógica y el comportamiento del sistema de manera más concisa y clara, reduciendo la complejidad y facilitando la comprensión.
- Al evitar restricciones de tipos, se pueden expresar propiedades complejas y relaciones entre variables con mayor facilidad, lo que ayuda a describir sistemas de manera más completa.
- La ausencia de tipos puede simplificar el proceso de formalización y verificación de propiedades críticas del sistema.

1.3. Ejercicio 3

Explique y justifique el significado de la fórmula final de una especificación TLA.

En TLA las especificaciones tiene una forma final sintáctica restringida: $Init \land \Box[Next]_{vars} \land Fairness_{vars}$

En la especificación vale el inicio *Init*, vale siempre *Next* admitiendo pasos intrascendentes sobre *vars* y por último las leyes de equidad.

En nuestra especificación se parte desde un estado conocido el cuál vale, y de ahí en adelante los eventos posteriores tienen a su vez resultados esperados. Esto junto con las propiedades de equidad donde sean necesarias, garantizan las propiedades de *safety* en nuestra especificación.

Se espera que *Init* restrinja el estado inicial, *Next* restrinja los pasos que pueden ocurrir y *Fairness* describa solo lo que eventualmente debe suceder.

1.4. Ejercicio 4

Explique y ejemplifique la forma de tener una cantidad ilimitada de instancias de un módulo en TLA+. Luego explique el significado formal de una expresión de la forma $...H(i)!Action...$ donde $H(i)$ es una instancia del módulo H y *Action* es una de las acciones definidas en H.

La forma de tener una cantidad ilimitada de instancias de un módulo es a partir de la parametrización de instancias. Por ejemplo:

$H(i) \triangleq INSTANCE\ H$

El significado formal de una expresión de la forma $...H(i)!Action...$ donde $H(i)$ es una instancia del módulo H y $Action$ es una de las acciones definidas en H es la siguiente:

Para cualquier símbolo $Action$ definido en el módulo H y cualquier expresión i, esto define $H(i)!Action$ a la fórmula igual a $Action$ con los reemplazos de variables correspondientes.

1.5. Ejercicio 5

Explique la contribución del concepto de máquina cerrada a la teoría de especificaciones de sistemas concurrentes.

Explicar que es una máquina cerrada, el porqué queremos obtener una máquina cerrada al especificar.

1.6. Ejercicio 6

Explique los conceptos de vitalidad, seguridad y equidad según se estudiaron en clase.

Las propiedades de **seguridad** (safety) 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.

Generalmente esto se hace a partir de indicar las transiciones permitidas en el sistema, a partir de esto se indican los comportamientos permitido y se prohiben implicitamente todos los demas.

La definicion formal de seguridad es la siguiente: S es una propiedad de seguridad respecto de E sii se verifica lo siguiente: $e \notin S \; sii \; \exists\, n \in \mathbb{N} : \forall\, t \in E^{\infty} : e_n \circ t \notin S$ para toda $e \in E^{\infty}$.

Las propiedades de **vitalidad** (liveness) 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.

Una propiedad de vitalidad indica los estados que obligatoriamente deberan alcanzarse durante una ejecucion.

La definicion formal de vitalidad es la siguiente: L es una propiedad de vitalidad respecto de E sii se verifica lo siguiente: $\forall\, e \in E* : \exists\, t \in E^{\infty} e \circ t \in L$

Las propiedades de **equidad** (fairness) son una clase de propiedades de vitalidad que tienen una forma muy particular. Existen dos versiones de equidad: débil (weak fairness) y fuerte (strong fairness).

La transición de estados A verifica weak fairness sii: $WF(A) \mathrel{\widehat{=}} \diamond\Box enabled(A) \Rightarrow \Box \diamond A$.

La transición de estados A verifica strong fairness sii: $SF(A) \mathrel{\widehat{=}} \Box \diamond enabled(A) \Rightarrow \Box \diamond A$.

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.

1.7. Ejercicio 7

Describa formalmente con cierto detalle la semántica de una especificación TLA de la forma:

$$Init \wedge \Box[Op_1 \vee Op_2]_v \wedge WF_v(Op_1)$$

En TLA las especificaciones tiene una forma sintáctica restringida: $Init \wedge \Box[Next]_{vars} \wedge Fairness_{vars}$.
En este caso, $Next = Op_1 \vee Op_2$, $vars = v$ y $Fairness_{vars} = WF_v(Op_1)$.

- $Init$ es el estado inicial del sistema.
- $Op_1 \vee Op_2$ es la disyunción de las acciones permitidas en el sistema.

- $\Box[Op_1 \vee Op_2]_v$ representa que la disyunción entre Op_1 y Op_2 siempre está permitida en el sistema, y se admiten pasos intrascendentes sobre las variables v.
- $WF_v(Op_1) = \diamond\Box enabled(Op_1) \Rightarrow \Box \diamond Op_1$ donde $enabled(Op_1)$ significa que valen las condiciones previas de Op_1.

Se espera que *Init* restrinja el estado inicial, *Next* restrinja los pasos que pueden ocurrir y *Fairness* describa solo lo que eventualmente debe suceder.

1.8. Ejercicio 8

1. Respecto del lenguaje TLA escriba el significado formal de $\Box[A]_v$ con respecto a una ejecución, donde A es una acción y v es una variable.
2. Explique la razón por la cual Lamport sugiere escribir la vitalidad de un sistema con fórmulas de equidad.

1. $\Box[A]_v$ significa que la acción A esta siempre permitida en el sistema, y se admiten pasos intrascendentes sobre las variables v. Esto quiere decir, que puede haber un paso-A sin que se modifiquen todas las variables involucradas en el estado.
$\Box[A]_v = \forall\, i \in Nat : A(e(i), e(i+1)) \vee e(i)(v) = e(i+1)(v)$

2. Una propiedad de vitalidad no debería agregar restricciones a un sistema, para ellos están las propiedades de seguridad. Sin embargo, si se admiten propiedades de vitalidad arbitrarias, estas pueden agregar restricciones al sistema. Es por esto, que Lamport sugiere escribir la vitalidad de un sistema con fórmulas de equidad, para así evitar el conflicto explicado anteriormente.

1.9. Ejercicio 9

Explique la incidencia del teorema Alpern-Schneider en el lenguaje de especificación TLA.

El teorema dice que: 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).

(Explicar el porque enrealidad se usa fairness en lugar de liveness)

1.10. Ejercicio 10

Indique, justificando su respuesta, todos los conceptos teóricos importantes que utiliza Lamport en TLA+ para definir el lenguaje. Por ejemplo, el teorema de Alpern-Schneider.

Lógica temporal, Estado, Vitalidad, Seguridad, Máquina cerrada, Equidad, Teorema de Alpern-Schneider, Teorema de Abadi-Lamport.

1.11. Ejercicio 11

Explique y ejemplifique la diferencia entre *EXTENDS* e *INSTANCE* en TLA+.

EXTENDS permite utilizar todas las definiciones efectuadas en los módulos que se extienden. Excepto *BOOLEAN*, todos los demás "tipos"deben ser incluidos usando esta cláusula. Por ejemplo, *EXTENDS Naturals* extiendo el módulo de los números naturales.

En cambio *INSTANCE* se utiliza para instanciar un determinado módulo, y renombrar algunas (o todas) sus variables. Por ejemplo, *INSTANCE Timer WITH time* $\leftarrow t1$. Esto instancia el módulo *Timer* y renombra la variable *time* por $t1$. Además permite la instanciación de módulos infinitos con la instanciación parametrizada.

1.12. Ejercicio 12

Defina formalmente el concepto de estado y explique por qué en TLA los estados son estados del universo y cuál es su utilidad en las especificaciones.

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 $1, (2, 3), 4, a, 6, N$, etc.). Entonces el estado s se define como una función de *Var* en *Val*: $s : Var \to Val$.

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.

1.13. Ejercicio 13

Explique y ejemplifique el concepto de pasos intrascendentes.

El concepto de pasos de ejecución repetitivos (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.

Supongamos que tenemos dos máquinas M_1 y M_2, se muestran algunas de sus ejecuciones:
(.! significa que . se repite hacia el infinito)

M_1:

1. $\langle A! \rangle$
2. $\langle A, B! \rangle$
3. $\langle A, B, C! \rangle$
4. $\langle A, B, C, A! \rangle$

M_2:

1. $\langle D! \rangle$
2. $\langle D, E! \rangle$
3. $\langle D, E, F! \rangle$
4. $\langle D, E, F, D! \rangle$

Consideremos el sistema M que resulta de componer M_1 y M_2, algunas de sus ejecuciones son:

1. $\langle (A, D)! \rangle$
2. $\langle (A, D), (B, D)! \rangle$
3. $\langle (A, D), (B, D), (B, E)! \rangle$
4. $\langle (A, D), (B, D), (B, E), (C, E)! \rangle$

y hagamos la proyección de la última ejecución sobre M_1:
$P_{M_1}(\langle (A, D), (B, D), (B, E), (C, E)! \rangle) = \langle A, B, B, C! \rangle$

El problema con esto es que esta no es una ejecución permitida 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, 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 son (donde $\cdot*$ significa que $\cdot$ puede repetirse un número finito de veces):

1. $\langle A! \rangle$
2. $\langle A*, B! \rangle$
3. $\langle A*, B*, C! \rangle$
4. $\langle A*, B*, C*, A! \rangle$

2. Práctica

2.1. Ejercicio 1

> Tomado en: 2007.03.09.
>
> Un brazo mecánico controlado digitalmente demora T_{ir} unidades de tiempo en ir de un extremo a otro pero el sistema debe detenerlo porque de lo contrario se dañaría el motor. Si durante su movimiento un operario pulsa un botón el sistema debe interrumpir el movimiento; si se vuelve a pulsar se lo debe reanudar, pero si no se pulsa el botón por segunda vez pasadas T_{det} unidades de tiempo desde la detención se lo debe mover en sentido contrario por el mismo tiempo que se lo movió hasta la detención. Cuando el brazo llega a cualquiera de los extremos debe permanecer allí a lo sumo T_{ext} unidades de tiempo luego de lo cual debe regresar (esto mismo vale para el estado inicial del sistema).
>
> Especifique en TLA$^+$el conocimiento de dominio y la especificación de los requerimientos anteriores. Escriba las designaciones correspondientes.

- T_i es el tiempo que demora el brazo en ir de un extremo al otro.
- T_d es el máximo que puede estar el brazo detenido.
- T_e es el máximo que puede estar el brazo en un extremo.
- Se oprime el botón $\approx$ Push. EC, S
- El brazo se empieza a mover $\approx$ StartMoving. EC, S
- El brazo llegó a un extremo $\approx$ Ext. MC, S
- Se frena el brazo con el botón $\approx$ Stop. EC, S
- El brazo estuvo T_d unidades de tiempo detenido, por lo que empieza a volver. *Combeack*. MC, S
- Se presiona el botón y se reanuda el movimiento del brazo $\approx$ Resume. EC, S

$$
\begin{array}{l}
\textit{MODULE} - \textit{DK\_Boton} \\
\textit{EXTENDS Naturals} \\
\textit{VARIABLES pressed} \\
\textit{BotonTypeInv} \mathrel{\widehat{=}} \textit{pressed} \in \{0, 1\} \\
\hline
\textit{BotonInit} \mathrel{\widehat{=}} \textit{pressed} = 0 \\
\textit{Push} \mathrel{\widehat{=}} \lor \textit{pressed} = 0 \land \textit{pressed}' = 1 \\
\qquad\qquad \lor \textit{pressed} = 1 \land \textit{pressed}' = 0 \\
\textit{BotonSpec} \mathrel{\widehat{=}} \textit{BotonInit} \land \Box[\textit{Push}]_{\textit{pressed}}
\end{array}
$$

MODULE − Brazo

EXTENDS Naturals

VARIABLES $position, goingto, lastgoingto, pressed, running, time, limit, now, state$

CONSTANTS T_d, T_e, T_i

ASSUME $T_d, T_e, T_i \in Nat \land T_d > 0 \land T_e > 0 \land T_i > 0$

$Positions \triangleq \{\overline{left}, \overline{right}\}$

$Directions \triangleq \{\overline{none}, \overline{left}, \overline{right}\}$

$States \triangleq \{\overline{ir}, \overline{ext}, \overline{det}, \overline{resume}, \overline{comeback}\}$

$Boton \triangleq INSTANCE\ DK\_Boton$

$Timer \triangleq INSTANCE\ SuperTimer$

$TypeInv \triangleq position \in Positions \land goingto, lastgointo \in Directions \land state \in States$
$\quad \land Boton!TypeInv \land Timer!TypeInv$

$vars1 = \langle position, goingto, lastgoingto, pressed, running, time, limit, state\rangle$

$vars2 = \langle position, goingto, lastgoingto, pressed, running, time, limit, state, now\rangle$

$Init \triangleq \land position = CHOOSE\ p : p \in \{\overline{left}, \overline{right}\}$
$\quad \land goingto = lastgoingto = \overline{none}$
$\quad \land state = \overline{ext}$
$\quad \land Boton!Init \land Timer!Init$

$SetTimeExt \triangleq state = \overline{ext} \land running = \overline{no} \land Timer!SetStart(T_e)$

$StartMoving \triangleq Timer!Timeout \land state = \overline{ext}$
$\quad \land (StartMoving1 \lor StartMoving2)$
$\quad \land state' = \overline{ir}$
$\quad \land UNCHANGED\langle position, goingto, lastgoingto, pressed\rangle$

$StartMoving1 \triangleq \land position = \overline{left}$
$\quad \land goingto' = lastgoingto' = \overline{right}$
$\quad \land UNCHANGED\langle position, pressed, running, time, limit, state\rangle$

$StartMoving2 \triangleq \land position = \overline{right}$
$\quad \land goingto' = lastgoingto' = \overline{left}$
$\quad \land UNCHANGED\langle position, pressed, running, time, limit, state\rangle$

$SetTimerIr \triangleq state = \overline{ir} \land running = \overline{no} \land Timer!SetStart(T_i)$

$Stop \triangleq \land Boton!Push \land state = \overline{ir}$
$\quad \land Timer!Stop$
$\quad \land goingto' = \overline{none}$
$\quad \land state' = \overline{det}$
$\quad \land UNCHANGED\langle position, lastgoingto\rangle$

$SetTimerDet \triangleq state = \overline{det} \land running = \overline{no} \land Timer!SetStart(T_d)$

$Resume \triangleq \land state = \overline{det}$
$\quad \land Boton!Push$
$\quad \land Timer!Stop$
$\quad \land state' = \overline{resume}$
$\quad \land UNCHANGED\langle position, goingto, lastgoingto\rangle$

$Comeback \triangleq \land state = \overline{det}$
$\quad \land Timer!Timeout$
$\quad \land state' = \overline{comeback}$
$\quad \land UNCHANGED\langle position, goingto, lastgoingto, pressed\rangle$

$DoResume \triangleq \land state = \overline{resume}$
$\quad \land Timer!SetStart(T_i - (now - time))$
$\quad \land goingto' = lastgoingto$
$\quad \land state' = \overline{ir}$
$\quad \land UNCHANGED\langle position, lastgoingto, pressed\rangle$

$DoComeback \triangleq \land state = \overline{comeback}$
$\quad \land Timer!SetStart(now - time)$
$\quad \land goingto' = IF\ lastgoingto = \overline{left}\ THEN\ \overline{right}\ ELSE\ \overline{left}$
$\quad \land UNCHANGED\langle position, lastgoingto, pressed, state\rangle$

$$
\begin{array}{l}
Ext \mathrel{\widehat{=}} \land(state = \overline{ir} \lor state = \overline{comeback}) \\
\qquad \land Timer!Timeout \\
\qquad \land position' = goingto \\
\qquad \land goingto' = \overline{none} \\
\qquad \land state' = \overline{ext} \\
\qquad \land UNCHANGED\langle lastgoingto, pressed\rangle \\
Next \mathrel{\widehat{=}} SetTimerExt \lor StartMoving \lor SetTimerIr \lor Stop \lor SetTimerDet \\
\qquad \lor Resume \lor Comeback \lor DoResume \lor DoComeback \lor Ext \\
Spec \mathrel{\widehat{=}} Init \land \Box[Next]_{vars1} \land \\
\qquad WF(SetTimerExt \lor SetTimerIr \lor SetTimerDet \lor DoResume \lor DoComeback \lor Ext)_{vars2}
\end{array}
$$

$$
THEOREM\ Spec \Rightarrow \Box TypeInv
$$

2.2. Ejercicio 2

Tomado en: 2008.08.08, 2010.07.30.

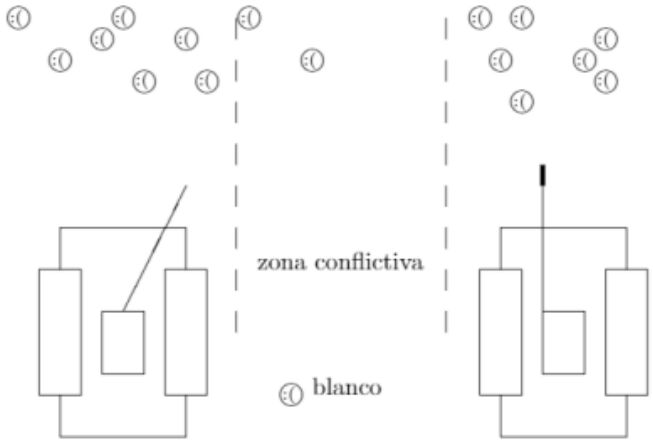
Se trata básicamente de que dos tanques en la misma zona de combate actúen coordinadamente en la selección y destrucción de blancos fijos. Cada tanque cuenta con un cañón, un sensor para detección de blancos por temperatura y una computadora de abordo. El programa de abordo, que consta de dos módulos funcionales A y B, se encarga de seleccionar blancos en base a los datos proporcionados por el sensor y a la cooperación con la computadora del otro tanque.

El módulo A comunica al módulo B la posición del blanco a partir de los datos crudos provistos por el sensor; la posición es el identificador universal para un blanco en la zona de combate. El módulo B se encarga de seleccionar los blancos en base a la posición y de mostrar a la tripulación los próximos blancos a destruir en el orden adecuado. La tripulación, leyendo estos datos, manejará el tanque y el cañón (es decir el sistema *no* conduce el tanque *ni* realiza los disparos).

Los dos tanques son jerárquicamente iguales; ninguno tiene prioridad sobre el otro para determinar la selección de blancos; ambos poseen una copia del módulo B en su sistema. En cada comunicación podrán determinar quién seleccionará un blanco en la zona conflictiva (ver la figura para mayores detalles).

La comunicación consta de dos mensajes. El iniciador de la comunicación le avisa al receptor que *no* puede encargarse del blanco en cuestión; el receptor puede aceptar el blanco o rechazarlo por tener más de *MAXTARGET* blancos seleccionados. Si el receptor acepta envía la respuesta correspondiente y luego lo ubica en su lista; si el receptor no acepta el blanco ambos lo reportan a la tripulación.

La funcionalidad se debe dividir en las siguientes acciones: recepción del blanco, clasificación del blanco (conflictivo o no), incorporación del blanco a la lista, actualización de la lista de blancos en pantalla, reporte de blanco no aceptado. La comunicación con el otro tanque es asíncrona.



Especifique en TLA$^+$el módulo funcional B del sistema que se describe arriba.

2.3. Ejercicio 3

Tomado en: 2010.02.12, 2015.07.10.

Cuando un cliente web solicita un archivo HTML a un servidor web la comunicación es asíncrona, es decir: un navegador solicita un archivo HTML, se corta la conexión y finalmente el servidor se comunica con el cliente, cuando puede satisfacer el pedido, para enviarle el archivo solicitado o el error apropiado.

También es asíncrona la comunicación cuando el cliente se autentica ante servidor: se envían los datos, se corta la conexión, el servidor verifica los datos, si son correctos registra al cliente y retorna el error apropiado.

Los archivos HTML que almacena el servidor pueden estar restringidos a ciertos clientes. Si este es el caso, el cliente que solicita uno de estos archivos debe estar autenticado y debe ser uno de los clientes autorizados a ver el archivo.

Por el contrario, cuando un cliente web desea enviarle un archivo al servidor la comunicación es asíncrona, aunque el servidor puede atender varios pedidos a la vez.

Un servidor web puede atender hasta M clientes simultáncamente.

Especifique en TLA$^+$el servidor mencionado en el problema anterior.

- Un usuario u pide hace loggin $\approx$ Loggin(u). EC, S
- Un usuario u pide descargar el archivo f $\approx$ Download(u, f). EC, S
- Se procesan los pedidos de loggin $\approx$ LogginReqts. MC, S
- Se procesan los pedidos de descargas $\approx$ DownloadReqts. MC, S
- Un usuario *uid* pide subir un archivo *fid* con contenido f y permitido para los usuarios $u \approx$ Upload(uid, fid, u, f). EC, S
- Un usuario u deja de subir archivos $\approx$ StopUpload(u). EC, S
- M es la cantidad maxima de usuarios que pueden subir archivos al mismo tiempo.
- Notify es la funcion subespecificada que notifica al usuario un error, o un mensaje con el archivo.
- Se inician la variable *users* vacia, y se supone que existe una accion que registra a los usuarios. Esto debido a que si se inicia la variable como *files*, las condiciones del estilo de $u : UID, u \in DOMAIN\ users$ no tienen sentido.

$MODULE - Servidor$

$EXTENDS\ Naturals, Sequences$

$CONSTANTS\ M, UID, FID, FILE, MSG, Notify(\_,\_)$

$ASSUME\ M \in Nat \land M > 0 \land$
$\quad \exists\, u : UID, m : MSG \cup FILE \bullet Notify(u, m) \in Boolean$

$VARIABLES\ users, files, logginReqts, downloadReqts, uploads$

$TypeInv \triangleq \land users \in [UID \to \{\overline{loggin}, \overline{loggout}\}]$
$\quad \land files \in [FID \to [u : Seq\ UID, f : FILE]]$
$\quad \land logginReqts \in Seq\ UID$
$\quad \land downloadReqts \in Seq\,[u : UID, f : FID]$
$\quad \land uploads \in Nat$

$NoFile \triangleq CHOOSE\, f : f \notin FILE$

$LogginErr \triangleq CHOOSE\, m : m \in MSG$

$DownloadErr \triangleq CHOOSE\, m : m \in MSG$

$UploadErr \triangleq CHOOSE\, m : m \in MSG$

$vars \triangleq \langle users, files, logginReqts, downloadReqts, uploads \rangle$

$Init \triangleq \land users = \emptyset$
$\quad \land files = [f \in FID \mapsto [u \mapsto \langle\rangle, f \mapsto NoFile]]$
$\quad \land logginReqts = \langle\rangle \land downloadReqts = \langle\rangle$
$\quad \land uploads = 0$

$Loggin(u) \triangleq logginReqts' = logginReqts \frown \langle u \rangle \land UNCHANGED \langle users, files, downloadReqts, uploads \rangle$

$Download(u', f') \triangleq \land downloadReqts' = downloadReqts \frown \langle [u \mapsto u', f \mapsto f'] \rangle$
$\quad \land UNCHANGED \langle users, files, logginReqts \rangle$

$LogginOk \triangleq \land head(logginReqts) \in DOMAIN\ users$
$\quad \land users' = [users\ EXCEPT\ ![head(logginReqts)] = \overline{loggin}]$
$\quad \land UNCHANGED \langle files, logginReqts, downloadReqts, uploads \rangle$

$LogginErr \triangleq \land head(logginReqts) \notin DOMAIN\ users$
$\quad \land Notify(head(logginReqts), LogginErr)$

$LogginReqts \triangleq \land logginReqts \neq \langle\rangle$
$\quad \land logginReqts' = tail(logginReqts)$
$\quad \land (LogginOk \lor LogginErr)$
$\quad \land UNCHANGED \langle users, files, downloadReqts, uploads \rangle$

$DownloadOk \triangleq \land head(downloadReqts).u \in DOMAIN\ users$
$\quad \land users[head(downloadReqts).u] = \overline{loggin}$
$\quad \land head(downloadReqts).f \in DOMAIN\ files$
$\quad \land head(downloadReqts).u \in files[head(downloadReqts).f].u$
$\quad \land Notify(head(downloadReqts).u, files[head(downloadReqts).f])$

$DownloadErr \triangleq (\lor head(downloadReqts).u \notin DOMAIN\ users$
$\quad \lor users[head(downloadReqts).u] \neq \overline{loggin}$
$\quad \lor head(downloadReqts).f \notin DOMAIN\ files$
$\quad \lor \bar{} head(downloadReqts).u \in files[head(downloadReqts).f].u)$
$\quad \land Notify(head(downloadReqts).u, DownloadErr)$

$DownloadReqts \triangleq \land downloadReqts \neq \langle\rangle$
$\quad \land downloadReqts' = tail(downloadReqts)$
$\quad \land (DownloadOk \lor DownloadErr)$
$\quad \land UNCHANGED \langle users, files, logginReqts, uploads \rangle$

$UploadOk(uid, fid, u', f') \triangleq \land uid \in DOMAIN\ users$
$\quad \land users[uid] = \overline{loggin}$
$\quad \land uploads < M$
$\quad \land uploads' = uploads + 1$
$\quad \land files' = [files\ EXCEPT\ files[fid] = [u \mapsto u', f \mapsto f']]$
$\quad \land UNCHANGED \langle users, logginReqts, downloadReqts \rangle$

$UploadErr(uid) \triangleq (\lor uid \notin DOMAIN\ users$
$\quad \lor users[uid] \neq \overline{loggin}$
$\quad \lor uploads = M)$
$\quad \land Notify(uid, UploadErr)$

$Upload(uid, fid, u', f') \triangleq UploadOk(uid, fid, u', f') \lor UploadErr(uid)$
$StopUpload(u) \triangleq uploads' = uploads - 1$
$Next \triangleq \exists\, uid : UID, fid : FID, u : Seq\ UID, f : FILE \bullet Loggin(uid) \lor$
$\quad Download(uid, fid) \lor Upload(uid, fid, u, f) \lor StopUpload(uid) \lor LogginReqts \lor DownloadReqts$
$Spec \triangleq Init \land \Box[Next]_{vars} \land SF(DownloadReqts \lor LogginReqts)_{vars}(?)$

THEOREM $Spec \Rightarrow \Box TypeInv$

2.4. Ejercicio 4

Tomado en: 2010.05.28.

Protocolo CSMA/CD. Para transmitir datos entre terminales de trabajo conectadas en red se debe hacer uso de algún protocolo. En algunas redes *broadcast* con un único bus la clave está en cómo asignar el uso de este cuando varias terminales compiten por él.

Uno de los protocolos que resuelven esta cuestión es el *Carrier Sense, Multiple Access with Collision Detection*, o simplemente CSMA/CD. Una breve descripción del funcionamiento de este protocolo es la siguiente.

Una terminal transmite al sistema (es decir a la implementación del protocolo) un mensaje que debe ser enviado por la red. El sistema, si el bus está disponible (esto es, no hay otra terminal transmitiendo), comienza a enviar su mensaje. Sin embargo, si detecta que el bus está ocupado, espera un tiempo aleatorio y vuelve a intentar transmitir el mensaje. Esto lo hará tantas veces como sea necesario hasta que pueda empezar a transmitir.

Aun tomando estas precauciones puede ocurrir que dos terminales usen el bus al mismo tiempo, lo que da lugar a una *colisión*. Cuando una colisión ocurre el bus comunica esta situación a todas las terminales. Esto implica que todas las terminales abortan inmediatamente las transmisiones y, nuevamente, eperan un tiempo aleatorio para empezar a transmitir de nuevo.

Los mensajes que colisionan se pierden. Una vez que el mensaje ha sido transmitido, la terminal que inició la transmisión es notificada.

Se supone que todos los mensajes (sin importar tamaño) demoran exactamente λ unidades de tiempo en ser transmitidos.

En resumen, en cada terminal corre una implementación del protocolo y todas las terminales comparten el bus. La interfaz que provee el bus consta de: determinar si el bus está libre o no, enviar un mensaje, comunicar que un mensaje se transmitió, comunicar a todas las terminales que hay colisión.

Describa en TLA la especificación del protocolo CSMA/CD que se describe arriba.

- El Bus esta libre ≈ Free. EC, S
- El bus recibe un mensaje m de la terminal t y lo transmite ≈ Send(t, m). EC, S.
- El bus envió el mensaje correctamente ≈ Sent. EC, S
- El bus detectó una colisión ≈ Colission. EC, S
- Se broadcastea el mensaje de colisión ≈ BroadcastColission() (función subespecificada)
- Se avisa a la terminal t que su mensaje fue enviado con éxito ≈ Ack(t) (función subespecificada)
- Una terminal envía el mensaje m por el bus ≈ Send(m). EC, S
- Se aborta la transmisión de la terminal ≈ Abort. EC, S

MODULE − DK_Bus

EXTENDS Naturals
CONSTANTS DATA, λ, *BroadcastColission*(), *Ack*(_)
ASSUME $\lambda \in Nat \land \lambda > 0$
$\land \forall\, i \in Nat : Ack(i) \in Boolean$
$\land BroadcastColission() \in Boolean$
VARIABLES msg, *trm*, *blimit*, *now*, *btime*, *brunning*
$NoMsg \triangleq CHOOSE\ m : m \notin DATA$
$NoTrm \triangleq CHOOSE\ t : t \notin Nat$
$Timer \triangleq INSTANCE\ Timer$
$TypeInv \triangleq msg \in DATA \cup \{NoMsg\} \land trm \in Nat \cup \{NoTrm\} \land Timer!TypeInv$
$vars1 \triangleq \langle msg, trm, blimit, btime, brunning \rangle$
$vars2 \triangleq \langle msg, trm, blimit, btime, brunning, now \rangle$

$Init \triangleq msg = NoMsg \land trm = NoTrm \land Timer!Init \land Timer!Set(\lambda)$
$Free \triangleq msg = NoMsg$
$Send(t, m) \triangleq msg = NoMsg \land msg' = m \land trm' = t \land Timer!Start$
$Sent \triangleq Timer!Timeout \land msg' = NoMsg \land trm' = NoTrm \land Ack(trm)$
$Collision \triangleq msg' = NoMsg \land trm' = NoTrm \land Timer!Stop \land BroadcastCollision()$
$Next \triangleq Free \lor (\exists\, t : Nat, m : DATA \bullet Send(t, m)) \lor Sent \lor Collision$
$Spec \triangleq Init \land \Box[Next]_{vars1} \land WF(Sent)_{vars2}$

$THEOREM\ Spec \Rightarrow \Box TypeInv$

MODULE − Terminal

EXTENDS Naturals
CONSTANTS N_{random}, *DATA*
ASSUME $N_{random} \in Nat \land N_{random} > 0$
VARIABLES num, *msg*, *trm*, *time*, *limit*, *running*, *now*
btime, *blimit*, *brunning*
$Bus \triangleq INSTANCE\ DK\_Bus$
$Timer \triangleq INSTANCE\ Timer$
$TypeInv \triangleq num \in Nat \land Bus!TypeInv \land Timer!TypeInv$
$vars \triangleq \langle num, time, limit, running \rangle \frown Bus!vars1$

$Init(n) \triangleq num = n \land Bus!Init \land Timer!Init \land Timer!Set(N_{random})$
$Send1(m) \triangleq Bus!Free \land \overline{\ }running \land Bus!Send(num, m)$
$Send2 \triangleq \overline{\ }Bus!Free \land Timer!Start$
$Send(m) \triangleq Send1(m) \lor Send2$
$Abort \triangleq \overline{\ }running \land Timer!Start$
$Next \triangleq (\exists\, m : DATA \bullet Send(m)) \lor Abort$
$Spec \triangleq Init \land \Box[Next]_{vars}$

$THEOREM\ Spec \Rightarrow \Box TypeInv$

2.5. Ejercicio 5

Tomado en: 2010.08.06.

Varios procesos compiten por la utilización de un cierto recurso. Cada proceso, en el momento en que desea utilizar el recurso, pregunta al sistema si el recurso está libre o no. Si lo está el sistema lo reserva para ese proceso, le comunica esta decisión y luego el proceso puede ejecutar una de tres operaciones diferentes sobre el recurso.

Si no lo está, el sistema le comunica al proceso esta situación; el proceso espera una unidad de tiempo y vuelve a preguntar, si la respuesta es la misma, espera dos unidades de tiempo y reitera la pregunta; esto se repite hasta que el recurso esté libre.

Si un proceso recibe el OK para utilizar el recurso pero pasan más de T unidades de tiempo sin que lo use, el sistema lo libera para que lo utilice otro proceso. Luego de que un proceso

utiliza el recurso, no podrá volver a utilizarlo hasta que lo utilice otro proceso diferente, a menos que el primer proceso sea el único en el sistema.

Describa un modelo TLA para los requerimientos enunciados.

- El proceso pasa a requerir el recurso ≈ Req. EC, S
- Se le concede el recurso r al proceso ≈ Granted(r). EC, S
- Se le deniega el acceso al recurso al proceso ≈ Denied. EC, S
- Se le remueve el recurso al proceso ≈ Removed. EC, S
- El proceso aplica 1 de 3 operaciones disponibles al recurso ≈ Ops. EC, S
- El proceso p le pide al sistema el recurso ≈ PreReq(p). EC, S
- El sistema remueve el acceso del recurso al proceso actual ≈ Remove. MC, S
- El sistema chequea que el proceso que tiene el recurso esta activo ≈ Active. EC, S
- T es la unidad de tiempo que espera un proceso para volver a pedir el acceso al recurso (Cada vez que se denega el acceso, se suma T al limite actual).
- REC es la constante que representa al tipo de dato de los recursos.
- OP1, OP2, OP3 son las operaciones subespecificadas que se le pueden aplicar al recurso.
- PID es el tipo de dato que representa a los identificadores de los procesos.
- T_{idle} es el tiempo limite en el cual un proceso puede estar inactivo.
- La variable n del sistema representa a la cantidad de procesos vivos, esta se inicia en 0 pero se supone que aumenta y decrementa por una accion la cual no corresponde al problema.

$MODULE - DK\_Process$

$EXTENDS\ Naturals$

$CONSTANTS\ T, REC, OP1(\_), OP2(\_), OP3(\_)$

$ASSUME\ T \in Nat \land T > 0 \land$
$\quad \forall\, r \in REC \bullet OP1(r), OP2(r), OP3(r) \in REC$

$VARIABLES\ state, resource$

$NoRec \triangleq CHOOSE\ r : r \notin REC$

$TypeInv \triangleq state \in \{\overline{dead}, \overline{none}, \overline{req}, \overline{wait}, \overline{acq}\} \land resource \in REC \cup \{NoRec\}$

$Timer \triangleq INSTANCE\ Timer$

$vars = \langle state, resource \rangle$

$Init \triangleq state = \overline{dead} \land resource = NoRec \land Timer!Init \land Timer!Set(T)$
$\quad \land UNCHANGED\ resource$

$Req \triangleq \land state = \overline{none}$
$\quad \land state' = \overline{req}$
$\quad \land UNCHANGED\ resource$

$Granted(r) \triangleq \land state = \overline{req}$
$\quad \land state' = \overline{acq}$
$\quad \land Timer!Set(T)$
$\quad \land UNCHANGED\ resource$

$Denied \triangleq \land state = \overline{req}$
$\quad \land state' = \overline{wait}$
$\quad \land Timer!Start$
$\quad \land UNCHANGED\ resource$

$Ready \triangleq \land Timer!Timeout$
$\quad \land state' = \overline{none}$
$\quad \land Timer!Set(Timer!limit + T)$
$\quad \land UNCHANGED\ resource$

$Removed \triangleq \land state = \overline{acq}$
$\quad \land state' = \overline{none}$
$\quad \land resource' = NoRec$

$O1 \triangleq \land state = \overline{acq}$
$\quad \land resource' = OP1(resource)$
$\quad \land UNCHANGED\ state$

$O2 \triangleq \land state = \overline{acq}$
$\quad \land resource' = OP2(resource)$
$\quad \land UNCHANGED\ state$

$O2 \triangleq \land state = \overline{acq}$
$\quad \land resource' = OP2(resource)$
$\quad \land UNCHANGED\ state$

$Ops \triangleq O1 \lor O2 \lor O3$

$Next \triangleq Live \lor Kill \lor Req \lor Denied \lor Ready \lor Removed \lor Ops \lor (\exists\, r : REC \bullet Granted(r))$

$Spec \triangleq Init \land \Box[Next]_{vars}$

$THEOREM\ Spec \Rightarrow \Box TypeInv$

$MODULE - System$

$EXTENDS\ Naturals$

$CONSTANTS\ PID, REC, T_{idle}$

$ASSUME\ T_{idle} \in Nat \land T_{idle} > 0$

$VARIABLES\ current, last, resource, req, n$

$NoPID \triangleq CHOOSE\ p : p \notin PID$

$TypeInv \triangleq current, last \in PID \cup \{NoPID\} \land resource \in REC$
$\quad req \in Seq\ PID \land n \in Nat$

$Timer \triangleq INSTANCE\ SuperTimer$

$Procs(p) \triangleq INSTANCE\ DK\_Process$

$vars = \langle current, last, resource, req, n \rangle$

$Init \triangleq \land current = last = NoPID$
$\quad \land req = \langle\rangle \land n = 0$
$\quad \land resource = CHOOSE\ r : r \in REC$
$\quad \land Timer!Init \land Timer!Set(T_{idle})$
$\quad \land \forall\, p \in PID \bullet Procs(p)!Init$

$PreReq(p) \triangleq \land Procs(p)!Req$
$\qquad \land req' = req \frown \langle p \rangle$
$\qquad \land UNCHANGED \langle current, last, resource, n \rangle$

$ReqOk1(p) \triangleq \land last \neq p$
$\qquad \land last' = current' = p$
$\qquad \land Procs(p)!Granted(r)$
$\qquad \land Timer!Start$
$\qquad \land UNCHANGED\ \langle resource, n \rangle$

$ReqOk2(p) \triangleq \land last = p$
$\qquad \land n = 1$
$\qquad \land Procs(p)!Granted(r)$
$\qquad \land Timer!Start$
$\qquad \land UNCHANGED \langle last, resource, n \rangle$

$ReqOk(p) \triangleq ReqOk1(p) \lor ReqOk2(p)$

$ReqDen(p) \triangleq \land p \neq NoPID$
$\qquad \land Procs(p)!Denied$

$Req \triangleq \land req \neq \langle\rangle$
$\quad \land (ReqOk(head(req)) \lor ReqDen(head(req))$
$\quad \land req' = tail(req)$

$Remove \triangleq \land Timer!Timeout$
$\qquad \land resource' = Procs(current).resource$
$\qquad \land Procs(current)!Removed$
$\qquad \land current' = NoPID$
$\qquad \land UNCHANGED \langle last, n \rangle$

$Active \triangleq \land Procs(current)!Ops$
$\qquad \land Timer!Restart$

$Next \triangleq Remove \lor Active \lor Req \lor (\exists\, p : PID \bullet PreReq(p))$

$Spec \triangleq Init \land \Box[Next]_{vars} \land WF(Req)_{vars}$

$THEOREM\ Spec \Rightarrow \Box TypeInv$

MODULE − System

$EXTENDS\ Naturals, Sequences$
$CONSTANTS\ PID, REC, T_{idle}, T_{wait}, OP1(\_), OP2(\_), OP3(\_)$
$ASSUME\ T_{idle}, T_{wait} \in Nat \land T_{idle} > 0 \land T_{wait} > 0 \land$
$\quad \forall\, r : REC \bullet OP1(r), OP2(r), OP3(r) \in REC$
$VARIABLES\ current, last, resource, processes, reqs, pset, running, time,$
$\quad limit, timers, start, now$
$NoPID \triangleq CHOOSE\ p : p \notin PID$
$States \triangleq \{\overline{dead}, \overline{active}, \overline{acq}, \overline{wait}\}$
$Running \triangleq \{\overline{no}, \overline{yes}\}$
$Timer \triangleq INSTANCE\ SuperTimer$
$Timers \triangleq INSTANCE\ Timers$
$TypeInv \triangleq current, last, pset \in PID \cup \{NoPID\} \land resource \in REC$
$\quad \land processes \in [PID \to States]$
$\quad \land reqs \in Seq\ PID$
$\quad \land Timer!TypeInv$
$\quad \land Timers!TypeInv$
$vars1 \triangleq \langle current, last, resource, processes, pset,$
$\quad reqs, running, time, limit, timers, start\rangle$
$vars2 \triangleq vars1 \frown \langle now\rangle$

$Init \triangleq \land current = last = pset = NoPID$
$\quad \land resource = CHOOSE\ r : r \in REC$
$\quad \land processes = [p \in PID \mapsto \overline{dead}]$
$\quad \land reqs = \langle\rangle$
$\quad \land Timer!Init \land Timer!Set(T_{idle})$
$\quad \land Timers!Init \land \forall\, p \in PID \bullet Timers!Set(p, T_{wait})$
$Active(p) \triangleq \land processes' = [processes\ EXCEPT\ ![p] = \overline{active}]$
$\quad \land UNCHANGED\langle current, last, resource, pset,$
$\quad\quad reqs, running, time, limit, timers, start, now\rangle$
$Kill(p) \triangleq \land processes[p] \neq \overline{acq}$
$\quad \land processes' = [processes\ EXCEPT\ ![p] = \overline{dead}]$
$\quad \land \bar{}(p\ \mathsf{in}\ reqs, p)$
$\quad \land UNCHANGED\langle current, last, resource, pset, running, time, limit, timers, start, now\rangle$
$Req(p) \triangleq \land processes[p] = \overline{active}$
$\quad \land reqs' = Append(reqs, p)$
$\quad \land UNCHANGED\langle current, last, resource, precesses, pset, running, time, limit, timers, start, now\rangle$
$Acq1(p) \triangleq \land current = NoPID$
$\quad \land p \neq last$
$\quad \land processes' = [processes\ EXCEPT\ ![p] = \overline{acq}]$
$\quad \land current' = last' = p$
$\quad \land UNCHANGED\langle resource, reqs, pset, running, time, limit, timers, start, now\rangle$
$Acq2(p) \triangleq \land current = NoPID$
$\quad \land p = last$
$\quad \land \#\{p : PID \mid processes[p] \neq \overline{dead}\} = 1$
$\quad \land processes' = [processes\ EXCEPT\ ![p] = \overline{acq}]$
$\quad \land current' = p$
$\quad \land UNCHANGED\langle last, resource, reqs, pset, running, time, limit, timers, start, now\rangle$
$AcqOk(p) \triangleq (Acq1(p) \lor Acq2(p)) \land Timer!Start \land Timers!Set(p, T_{wait})$
$AcqErr(p) \triangleq \land current \neq NoPID$
$\quad \land processes' = [processes\ EXCEPT\ ![p] = \overline{wait}]$
$\quad \land Timers!Start(p)$
$\quad \land UNCHANGED\langle current, last, resource, pset$
$\quad\quad reqs, timers\rangle$

$$
\begin{array}{l}
Acq \triangleq reqs \neq \langle\rangle \land \\
\quad LET\ p \triangleq head(reqs) \\
\quad IN \\
\quad \land (AcqOk(p) \lor AcqErr(p)) \\
\quad \land reqs' = tail(reqs) \\
\quad \land UNCHANGED \langle current, last, resource, pset, \\
\qquad\qquad running, time, limit, timers, start, now \rangle \\
StopWait \triangleq \exists\, p : PID \bullet \\
\qquad \land Timers!Timeout(p) \\
\qquad \land processes' = [processes\ EXCEPT\ ![p] = \overline{active}] \\
\qquad \land pset' = p \\
\qquad \land UNCHANGED \langle current, last, resource, \\
\qquad\qquad running, time, limit, timers, start, now \rangle \\
SetTimer \triangleq \land pset \neq NoPID \\
\qquad \land Timers!Set(pset, Timers[pset].l + T_{wait}) \\
\qquad \land pset' = NoPID \\
\qquad \land UNCHANGED \langle current, last, resource, processes, reqs, running, time, limit, start, now \rangle \\
Op1 \triangleq resource' = OP1(resource) \land UNCHANGED\ Remove(vars1, resource) \\
Op2 \triangleq resource' = OP2(resource) \land UNCHANGED\ Remove(vars1, resource) \\
Op3 \triangleq resource' = OP3(resource) \land UNCHANGED\ Remove(vars1, resource) \\
Ops \triangleq Op1 \lor Op2 \lor Op3 \\
CheckIdle \triangleq Ops \land Timer!Restart \\
Remove \triangleq \land Timer!Timeout \\
\qquad \land processes' = [processes\ EXCEPT\ ![current] = \overline{active}] \\
\qquad \land currrent' = NoPID \\
\qquad \land UNCHANGED \langle last, resource, reqs, pset, running, time, limit, timers, start, now \rangle \\
Next \triangleq (\exists\, p : PID \bullet Active(p) \lor Kill(p) \lor Req(p)) \lor Acq \\
\qquad \lor StopWait \lor SetTimer \lor Ops \lor CheckIdle \lor Remove \\
Spec \triangleq Init \land \Box[Next]_{vars} \land WF(Acq, SetTimer)_{vars2} \\
\hline
THEOREM\ Spec \Rightarrow \Box TypeInv
\end{array}
$$

2.6. Ejercicio 6

Tomado en: 2010.12.10.

Un sistema debe controlar un aparato para efectuar electroencefalogramas simples. El análisis consiste en estudiar el voltaje que tiene un conjunto de 10 electrodos que permiten conocer la actividad bioeléctrica cerebral (cada uno comunica un valor al sistema). Es necesario tomar 5 muestras por segundo, espaciadas uniformemente. En cada una de las 5 muestras se lee el valor de los 10 electrodos. Notar que si el cerebro del paciente no presenta actividad en las cercanías de un electrodo, este no emitirá señal alguna. Por lo tanto, el sistema no puede esperar indefinidamente por la señal del electrodo. Finalmente, el sistema debe enviar secuencialmente a una impresora el valor obtenido en cada electrodo (que haya retornado uno o nada en caso de que no se haya registrado ninguno).

Escriba las designaciones y modele en TLA el conocimiento de dominio y la especificación de los requerimientos que se enuncian arriba. Se premiará el uso correcto del carácter no tipado de TLA. Para las cuestiones temporales puede asumir la existencia de un temporizador como el que se mostró en clase.

- Se leen los valores de los 10 electrodos ≈ Read. EC, S
- El sistema comienza a leer los valores de los 10 electrodos ≈ Read. MC, S
- El sistema esta leyendo los valores de los 10 electrodos ≈ Reading. MC, S
- El sistema deja de leer los valores de los 10 electrodos ≈ StopReading. MC, S

- El sistema manda a imprimir los valores que recolecto de los 10 electrodos ≈ Printing. MC, S

MODULE − DK_Electrodos

EXTENDS Naturals
CONSTANTS VOLT
VARIABLES electrodes
$NoVolt \triangleq CHOOSE\ x : x \notin VOLT$
$TypeInv \triangleq electrodes \in [\{0\,..\,9\} \to VOLT \cup \{NoVolt\}]$

$Init \triangleq electrodes = [n \in \{0\,..\,9\} \mapsto NoVolt]$
$Read \triangleq electrodes' = [n \in \{0\,..\,9\} \mapsto CHOOSE\ x : x \in VOLT \cup \{NoVolt\}]$
$Spec \triangleq Init \land \Box[Read]_{electrodes}$

THEOREM $Spec \Rightarrow \Box TypeInv$

MODULE − System

EXTENDS Naturals
CONSTANTS $VOLT, Print(\_)$
ASSUME $\forall\, x \bullet Print(x) \in Boolean$
VARIABLES $samples, printing, electrodes, running, limit, time, now$
$NoVolt \triangleq CHOOSE\ x : x \notin VOLT$
$TypeInv \triangleq samples \in Seq\ VOLT \cup \{NoVolt\} \land printing \in Boolean$
$\land electrodes \in [\{0\,..\,9\} \to VOLT \cup \{NoVolt\}]$
$\land running \in yes, no \land limit, time, now \in Nat$
$Electrodes \triangleq INSTANCE\ DK\_Electrodos$
$Timer \triangleq INSTANCE\ Timer$
$vars = \langle samples, printing, electrodes, running, limit, time \rangle$

$Init \triangleq samples = \langle\rangle \land printing = FALSE \land Electrodes!Init \land Timer!Init.Timer!Set(1)$
$Read \triangleq \neg printing \land Timer!Start$
$Reading \triangleq Timer!running = \overline{yes} \land Electrodes!Read \land$
$samples' = samples \frown \langle \forall\, x : \{0\,..\,9\} \bullet Electrodes!electrodes[x] \rangle$
$StopReading \triangleq Timer!Timeout \land printing' = TRUE$
$Printing1 \triangleq samples = \langle\rangle \land printing' = FALSE$
$Printing2 \triangleq samples \neq \langle\rangle \land Print(head(samples)) \land samples' = tail(samples)$
$Printing \triangleq printing \land (Printing1 \lor Printing2)$
$Next \triangleq Read \lor Reading \lor StopReading \lor Printing$
$Spec \triangleq Init \land \Box[Next]_{vars} \land WF(Reading \lor StopReading \lor Printing)$

THEOREM $Spec \Rightarrow \Box TypeInv$

CONSULTAS

- Caso CHOOSE en record, genera distintos valores o uno solo?
- Caso samples' en Reading
- En Printing1 (done printing) deberíamos llamar nuevamente a Read?

2.7. Ejercicio 7

Tomado en: 2011.02.25.

Un sistema de memoria consiste en cierta cantidad de procesadores que se comunican con la memoria física a través de cierta interfaz. Esta interfaz posee una operación por medio de la cual un procesador puede requerir a la memoria una lectura o escritura, y otra operación por medio de la cual la memoria envía cierto valor a un procesador. La interfaz está dada, no debe ser programada; se debe programar el funcionamiento de la memoria física.

Los procesadores pueden escribir un valor en una celda de memoria o solicitar el valor almacenado en una celda. Cada procesador efectúa un pedido a la vez y espera la respuesta de la memoria antes de hacer el siguiente pedido. La respuesta a un pedido de lectura es el valor almacenado en la celda solicitada y la respuesta a un pedido de escritura es un código especial que indica que la operación a concluido.

Claramente, ni la interfaz ni la memoria física pueden controlar cuándo un procesador hará una solicitud. Por lo tanto, se espera que el sistema esté preparado para recibir pedidos en cualquier momento y que utilice los períodos ociosos para completar las operaciones.

Se espera que todo pedido efectuado por algún procesador eventualmente reciba una respuesta proveniente de la memoria.

Escriba las designaciones y modele en TLA el conocimiento de dominio y la especificación de los requerimientos que se enuncian arriba.

2.8. Ejercicio 8

Tomado en: 2011.09.09.

Un proceso, B, debe almacenar elementos en dos *buffers* de la misma capacidad finita y conocida, N. Por otro lado, existen procesos (llamados productores) que envían datos a B para que este los almacene en los /emphbuffers, y existen procesos (llamados consumidores) que le piden a B los datos que tiene almacenados (lo que hace que los *buffers* se vayan vaciando).

Cuando un consumidor quiere un dato que B tiene, el proceso le debe indicar el *buffer* del cual lo quiere; en cambio los productores no pueden seleccionar el *buffer*.

Obviamente B no puede poner elementos en un *buffer* lleno y no puede sacar elementos de un *buffer* vacío. Lo que sí debe hacer B es balancear el uso de los *buffers*: cuando almacena datos lo debe hacer en el *buffer* más vacío y si un *buffer* se está vaciando más rápido que el otro, debe pasar elementos del último al primero.

Describa en TLA$^+$la especificación y el conocimiento de dominio de los requerimientos anteriores. Designe los términos básicos de su modelo. Debe poner especial cuidado en determinar cuáles de las operaciones que implícitamente se describen son internas y cuáles son externas.

- N es el tamaño máximo de los buffers.
- El dato v se agrega a uno de los dos buffers, de manera balanceada $\approx$ Add(v). EC, S
- Se consume un dato del buffer $n \approx$ Consume(n). EC, S
- Se balancean los buffers $\approx$ Balance. MC, S
- Se dejan de balancear los buffers $\approx$ StopBalance. MC, S

$$
\begin{array}{l}
\textit{MODULE} - \textit{DK\_Process} \\
\textit{EXTENDS } \textit{Naturals}, \textit{Sequences} \\
\textit{CONSTANT } \textit{DATA}, N, \textit{NULLDATA} \\
\textit{ASSUME } N \in \textit{Nat} \land N > 0 \land \textit{NULLDATA} \in \textit{DATA} \\
\textit{VARIABLES } b1, b2, \textit{dataOut}, \textit{balancing} \\
\textit{TypeInv} \triangleq b1, b2 \in \textit{Seq DATA} \land \textit{dataOut} \in \textit{DATA} \land \textit{balancing} \in \textit{Boolean} \\
\textit{vars} \triangleq \langle b1, b2, \textit{dataOut}, \textit{balancing} \rangle \\
\hline
\textit{Init} \triangleq b1 = \langle\rangle \land b2 = \langle\rangle \land \textit{dataOut} = \textit{NULLDATA} \land \textit{balancing} = \textit{FALSE} \\
\textit{Add1}(v) \triangleq \textit{Len } b1 = \textit{Len } b2 \land \textit{Len } b1 < N \land b1' = \langle v \rangle \land \textit{UNCHANGED } \langle b2, \textit{dataOut}, \textit{balancing} \rangle \\
\textit{Add2}(v) \triangleq \textit{Len } b1 = 0 \land \textit{Len } b2 > 0 \land b1' = \langle v \rangle \land \textit{UNCHANGED } \langle b2, \textit{dataOut}, \textit{balancing} \rangle \\
\textit{Add3}(v) \triangleq \textit{Len } b1 > 0 \land \textit{Len } b2 = 0 \land b2' = \langle v \rangle \land \textit{UNCHANGED } \langle b1, \textit{dataOut}, \textit{balancing} \rangle \\
\textit{Add4}(v) \triangleq \textit{Len } b1 > 0 \land \textit{Len } b2 > 0 \land \\
\qquad \textit{Len } b1 > \textit{Len } b2 \land b2' = b2 \frown \langle v \rangle \land \textit{UNCHANGED } \langle b1, \textit{dataOut}, \textit{balancing} \rangle \\
\textit{Add5}(v) \triangleq \textit{Len } b1 > 0 \land \textit{Len } b2 > 0 \land \\
\qquad \textit{Len } b2 > \textit{Len } b1 \land b1' = b1 \frown \langle v \rangle \land \textit{UNCHANGED } \langle b2, \textit{dataOut}, \textit{balancing} \rangle \\
\textit{Add}(v) \triangleq \neg \textit{balancing} \land (\textit{Add1}(v) \lor \textit{Add2}(v) \lor \textit{Add3}(v) \lor \textit{Add4}(v) \lor \textit{Add5}(v)) \\
\textit{Consume}(1) \triangleq \neg \textit{balancing} \land b1' = \textit{tail}(b1) \land \textit{dataOut}' = \textit{head}(b1) \land \textit{UNCHANGED } \langle b2, \textit{balancing} \rangle \\
\textit{Consume}(2) \triangleq \neg \textit{balancing} \land b2' = \textit{tail}(b2) \land \textit{dataOut}' = \textit{head}(b2) \land \textit{UNCHANGED } \langle b1, \textit{balancing} \rangle \\
\textit{Balance}1 \triangleq \textit{Len } b1 - \textit{Len } b2 > 1 \land b1' = \textit{tail}(b1) \land \textit{balancing}' = \textit{TRUE} \land b2' = b2 \frown \textit{head}(b1) \land \\
\qquad \textit{UNCHANGED dataOut} \\
\textit{Balance}2 \triangleq \textit{Len } b2 - \textit{Len } b1 > 1 \land b2' = \textit{tail}(b2) \land \textit{balancing}' = \textit{TRUE} \land b1' = b1 \frown \textit{head}(b2) \land \\
\qquad \textit{UNCHANGED dataOut} \\
\textit{Balance} \triangleq \textit{Balance}1 \lor \textit{Balance}2 \\
\textit{StopBalance} \triangleq \textit{balancing} \land \|\textit{Len } b1 - \textit{Len } b2\| \leq 1 \land \textit{balancing}' = \textit{FALSE} \land \textit{UNCHANGED } \langle b1, b2, \textit{dataOut} \rangle \\
\textit{Next} \triangleq \exists\, v \in \textit{DATA} \bullet \textit{Add}(v) \land \exists\, n \in (1 \,..\, 2) \bullet \textit{Consume}(n) \land \textit{Balance} \land \textit{StopBalance} \\
\textit{Spec} \triangleq \textit{TypeInv} \land \Box[\textit{Next}]_{\textit{vars}} \land \textit{WF}(\textit{Balance} \lor \textit{StopBalance})_{\textit{vars}} \\
\hline
\textit{THEOREM Spec} \Rightarrow \Box \textit{TypeInv}
\end{array}
$$

2.9. Ejercicio 9

Una caldera consiste en un tanque de agua y un quemador. El quemador calienta el agua y el agua caliente viaja por tuberías para calefaccionar un edificio. El tanque se llena con agua de la red pero fundamentalmente se retroalimenta con el agua que circula por las tuberías del edificio (es decir, el agua sale muy caliente del tanque, viaja por las tuberías, pierde calor en el trayecto y vuelve a ingresar al tanque). Un termómetro mide la temperatura del agua en el tanque y un barómetro la presión.

El quemador, las válvulas de entrada de agua de la red y las de entrada o salida de las cañerías pueden ser controladas digitalmente. El termómetro y el barómetro son sensores electrónicos activos.

El sistema debe controlar que la presión y la temperatura del agua en el tanque se mantengan dentro de ciertos parámetros. Cuando la temperatura sube (baja) se debe bajar (subir) el quemador; cuando la presión aumenta se debe liberar el agua a las cañerías, pero si disminuye se debe suministrar agua de la red.

Liste las designaciones y confeccione la tabla de control y visibilidad para los fenómenos de interés del problema anterior. Luego modele en TLA el conocimiento de dominio y la especificación del software de control que se mencionan.

- El sensor sensa el valor $v \approx$ Sens(v). EC, S
- Se setea el estado s al quemador $\approx$ Set(s). EC, S
- Se abre la valvula $\approx$ Open. EC, S
- Se cierra la valvula $\approx$ Close. EC, S

- El sistema chequea la temperatura del agua $\approx$ CheckTemp. MC, S
- El sistema chequea la presion del agua $\approx$ CheckPre. MC, S
- TermLim es el valor a partir del cual la temperatura es 'alta'.
- TermMax es la temperatura maxima que sensa el termometro.
- BarLim es el valor a partir del cual la presion es 'alta'.
- BarMax es la presion maxima que sensa el barometro.

MODULE − DK_Sens

EXTENDS Naturals
Variables val
$TypeInv \mathrel{\widehat{=}} val \in Nat$

$Init \mathrel{\widehat{=}} val = 0$
$Sens(v) \mathrel{\widehat{=}} val' = v$
$Next \mathrel{\widehat{=}} \exists\, v : Nat \bullet Sens(v)$
$Spec \mathrel{\widehat{=}} Init \wedge \Box[Next]_{val}$

THEOREM $Spec \Rightarrow \Box TypeInv$

MODULE − DK_Quemador

Variables state
$QStates \mathrel{\widehat{=}} \{\overline{low}, \overline{high}\}$
$TypeInv \mathrel{\widehat{=}} state \in QStates$

$Init \mathrel{\widehat{=}} state = \overline{low}$
$Set(s) \mathrel{\widehat{=}} state \neq s \wedge state' = s$
$Next \mathrel{\widehat{=}} \exists\, s : QStates \bullet Set(s)$
$Spec \mathrel{\widehat{=}} Init \wedge \Box[Next]_{state}$

THEOREM $Spec \Rightarrow TypeInv$

MODULE − DK_Valvula

Variables state
$TypeInv \mathrel{\widehat{=}} state \in \{\overline{closed}, \overline{open}\}$

$Init \mathrel{\widehat{=}} state = \overline{closed}$
$Open \mathrel{\widehat{=}} state = \overline{closed} \wedge state' = \overline{open}$
$Close \mathrel{\widehat{=}} state = \overline{open} \wedge state' = \overline{closed}$
$Next \mathrel{\widehat{=}} Open \vee Close$
$Spec \mathrel{\widehat{=}} Init \wedge \Box[Next]_{state}$

THEOREM $Spec \Rightarrow TypeInv$

$$
\begin{array}{l}
\textit{MODULE} - \textit{Caldera} \\
\textit{EXTENDS Naturals} \\
\textit{CONSTANTS } \textit{TermLim}, \textit{TermMax}, \textit{BarLim}, \textit{BarMax} \\
\textit{ASSUME } \textit{TermLim}, \textit{TermMax}, \textit{BarLim}, \textit{BarMax} \in \textit{Nat} \land \\
\qquad 0 < \textit{TermLim} < \textit{TermMax} \land 0 < \textit{BarLim} < \textit{BarMax} \\
\textit{VStates} \triangleq \{\overline{\textit{closed}}, \overline{\textit{open}}\} \\
\textit{QStates} \triangleq \{\overline{\textit{low}}, \overline{\textit{high}}\} \\
\textit{VARIABLES } \textit{term}, \textit{bar}, \textit{quem}, \textit{valv} \\
\hline
\textit{Quem} \triangleq \textit{INSTANCE DK\_Quemador WITH state} \leftarrow \textit{quem} \\
\textit{Valv} \triangleq \textit{INSTANCE DK\_Valvula WITH state} \leftarrow \textit{valv} \\
\textit{TypeInv} \triangleq \textit{term} \in (0 \,..\, \textit{TermMax}) \land \textit{bar} \in (0 \,..\, \textit{BarMax}) \land \textbf{(DUDA 1)} \\
\qquad \textit{Quem!TypeInv} \land \textit{Valv!TypeInv} \\
\textit{Term} \triangleq \textit{INSTANCE DK\_Sens WITH val} \leftarrow \textit{term} \\
\textit{Bar} \triangleq \textit{INSTANCE DK\_Sens WITH val} \leftarrow \textit{bar } \textit{vars} \triangleq \langle \textit{term}, \textit{bar}, \textit{quem}, \textit{valv} \rangle \\
\hline
\textit{Init} \triangleq \textit{Term!Init} \land \textit{Bar!Init} \land \textit{Quem!Init} \land \textit{Valv!Init} \\
\textit{CheckTemp1} \triangleq \land \textit{term} < \textit{TermLim} \\
\qquad\qquad \land \textit{Quem!Set}(\overline{\textit{high}}) \\
\textit{CheckTemp2} \triangleq \land \textit{term} \geq \textit{TermLim} \\
\qquad\qquad \land \textit{Quem!Set}(\overline{\textit{low}}) \\
\textit{CheckTemp} \triangleq \textit{CheckTemp1} \lor \textit{CheckTemp2} \\
\textit{CheckPre1} \triangleq \land \textit{bar} < \textit{BarLim} \\
\qquad\qquad \land \textit{Valv!Open} \\
\textit{CheckPre2} \triangleq \land \textit{bar} \geq \textit{BarLim} \\
\qquad\qquad \land \textit{Valv!Close} \\
\textit{CheckPre} \triangleq \textit{CheckPre1} \lor \textit{CheckPre2} \\
\textit{Next} \triangleq \textit{CheckTemp} \lor \textit{CheckPre} \\
\textit{Spec} \triangleq \textit{Init} \land \Box[\textit{Next}]_{\textit{vars}} \land \textit{SF}(\textit{CheckTemp} \lor \textit{CheckPree})_{\textit{vars}} \\
\hline
\textit{THEOREM Spec} \Rightarrow \Box \textit{TypeInv}
\end{array}
$$

1. Es correcto restringir el dominio de los sensores al instanciarlos? Por ejemplo, que me impide que el sensor sense algo mayor a TermMax, dado que en el DK el valor que sensa es Nat.

2.10. Ejercicio 10

Tomado en: 2017.08.4.

(Segun lo recuerdo)

Un celular consiste de una bocina que puede ser prendida o apagada, un display y un teclado numerico con dos botones de "cortar" y "enviar".

El celular se enciende luego de presionar el boton de "cortar" dos segundos. Mientras no se este realizando ninguna llamada se puede presionar numeros del teclado y estos apareceran en el display. Si se presiona el boton de "enviar" se llama al numero marcado, si se presiona "cortar" se borra el display.

Dada una llamada entrante sonara la bocina intermitentemente de la siguiente forma: 1 segundo de sonido seguido de 1 sonido de silencio. Si se presiona el boton de "cortar" se podra volver a marcar numeros, si se presiona "enviar" se entra en la llamada. No se modelaran las llamadas.

Por ultimo, si el numero de una llamada entrante se encuentra en la agenda se mostrara el nombre del mismo.

- Se enciende la bocina $\approx$ Bon.
- Se apaga la bocina $\approx$ Boff.
- Se escribe secuencia s en el display $\approx$ DReplace(s).

- Se borra en display ≈ Ddelete.
- Se presiona boton i ≈ Press(i).
- Se deja de presionar el boton i ≈ Release(i). (Duda, quizas hay que hacerlo con un timer interno)
- Se pulsa el boton i ≈ Pulse(i).
- Comienza el encendido del celular ≈ Startup.
- Se aborta el encendido del celular ≈ Abort.
- Se enciende el celular ≈ On.
- Se borra el contenido de la pantalla ≈ Delete.
- Se corta una llamada iniciada ≈ Hangup.
- Se corta una llamada entrante ≈ HangupIncoming.
- Se recibe una llamada del numero n ≈ IncomingCall(n).
- Se alterna la bocina del celular al recibir una llamada ≈ Alternate.
- Se acepta una llamada entrante ≈ Accept.
- Se escribe el caracter i en la pantalla ≈ Write(i).

Module − DK_Bocina

$$
\begin{array}{l}
\textit{VARIABLES } state \\
\textit{TypeInv} \mathrel{\widehat{=}} state \in \{\overline{off}, \overline{on}\} \\
\hline
\textit{Init} \mathrel{\widehat{=}} state = \overline{off} \\
\textit{BOn} \mathrel{\widehat{=}} state' = \overline{on} \\
\textit{BOff} \mathrel{\widehat{=}} state' = \overline{off} \\
\textit{Next} \mathrel{\widehat{=}} On \lor Off \\
\textit{Spec} \mathrel{\widehat{=}} Init \land \Box[Next]_{state} \\
\hline
\textit{THEOREM } Spec \Rightarrow \Box TypeInv
\end{array}
$$

Module − DKDisplay

$$
\begin{array}{l}
\textit{EXTENDS } Naturals, Sequences \\
\textit{CONSTANTS } CHARACTER \\
\textit{VARIABLES } content \\
\textit{ASSUME } (0 \,..\, 9) \subseteq CHARACTER \\
\textit{TypeInv} \mathrel{\widehat{=}} content \in Seq\ CHARACTER \\
\hline
\textit{Init} \mathrel{\widehat{=}} content = \langle\rangle \\
\textit{DReplace}(s) \mathrel{\widehat{=}} content' = s \\
\textit{DDelete} \mathrel{\widehat{=}} content' = \langle\rangle \\
\textit{Next} \mathrel{\widehat{=}} \exists\, s : Seq\ CHARACTER \bullet \\
\qquad\qquad DReplace(s) \lor DDelete \\
\textit{Spec} \mathrel{\widehat{=}} Init \land \Box[Next]_{content} \\
\hline
\textit{THEOREM } Spec \Rightarrow \Box TypeInv
\end{array}
$$

Module − DK_Teclado

EXTENDS Naturals
VARIABLES register
$TypeInv \mathrel{\widehat{=}} register \in [(0\,..\,11) \to \{0,1\}]$

$Init \mathrel{\widehat{=}} register = [n \in (0\,..\,11) \mapsto 0]$
$Press(i) \mathrel{\widehat{=}} register[i] = 0 \land register' = [register\ EXCEPT\ ![i] = 1]$
$Release(i) \mathrel{\widehat{=}} register[i] = 1 \land register' = [register\ EXCEPT\ ![i] = 0]$
$Pulse(i) \mathrel{\widehat{=}} Press(i).Release(i)$
$Next \mathrel{\widehat{=}} \exists\, i : (0\,..\,12) \bullet Press(i) \lor Release(i) \lor Pulse(i)$
$Spec \mathrel{\widehat{=}} Init \land \Box[Next]_{register}$

$THEOREM\ Spec \Rightarrow \Box TypeInv$

Module − Celular

$$
\begin{array}{l}
\textit{EXTENDS } \textit{Naturals}, \textit{Sequences} \\
\textit{CONSTANTS } \textit{CHARACTER} \\
\textit{ASSUME } (0\,..\,9) \subseteq \textit{CHARACTER} \\
\textit{VARIABLES } \textit{state}, \textit{incall}, \textit{incomingcall}, \textit{contacts}, \textit{buffer} \\
\textit{TypeInv} \mathrel{\widehat{=}} \land \textit{state}, \textit{incall}, \textit{incomingcall} \in \{\overline{\textit{off}}, \overline{\textit{on}}\} \\
\qquad\qquad \land \textit{contacts} \in [\textit{Seq Nat} \to \textit{Seq CHARACTER}] \\
\qquad\qquad \land \textit{buffer} \in \textit{Seq Nat} \\
\textit{bocina} \mathrel{\widehat{=}} \textit{INSTANCE DKBocina} \\
\textit{display} \mathrel{\widehat{=}} \textit{INSTANCE DKDisplay} \\
\textit{teclado} \mathrel{\widehat{=}} \textit{INSTANCE DKTeclado} \\
\textit{timer} \mathrel{\widehat{=}} \textit{INSTANCE Timer} \\
\textit{vars} = \langle \textit{state}, \textit{incall}, \textit{incomingcall}, \textit{contacts}, \textit{buffer} \rangle
\end{array}
$$

$$
\begin{array}{l}
\textit{Init} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{off}} \land \textit{incall} = \overline{\textit{off}} \land \textit{incomingcall} = \overline{\textit{off}} \land \textit{buffer} = \langle\rangle \\
\qquad \land \textit{bocina}!\textit{Init} \land \textit{display}!\textit{Init} \land \textit{teclado}!\textit{Init} \\
\qquad \land \textit{timer}!\textit{Init} \land \textit{timer}!\textit{Set}(2) \\
\textit{Startup} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{off}} \land \textit{teclado}!\textit{Press}(10) \land \textit{timer}!\textit{Start} \\
\textit{Abort} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{off}} \land \textit{teclado}!\textit{Release}(10) \land \textit{timer}!\textit{Stop}.\textit{timer}!\textit{Set}(2) \\
\textit{On} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{off}} \land \textit{timer}!\textit{Timeout} \\
\qquad \land \textit{state}' = \overline{\textit{on}} \\
\qquad \land \textit{UNCHANGED} \langle \textit{incall}, \textit{incomingcall}, \textit{contacts}, \textit{buffer} \rangle \\
\textit{Delete} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{on}} \land \textit{incall} = \overline{\textit{off}} \land \textit{incomingcall} = \overline{\textit{off}} \land \textit{teclado}!\textit{Pulse}(11) \\
\qquad\quad \land \textit{buffer}' = \langle\rangle \land \textit{display}!\textit{DDelete} \\
\qquad\quad \textit{UNCHANGED} \langle \textit{state}, \textit{incall}, \textit{incomingcall}, \textit{contacts} \rangle \\
\textit{Hangup} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{on}} \land \textit{incall} = \overline{\textit{on}} \land \textit{teclado}!\textit{Pulse}(11) \\
\qquad\quad \land \textit{incall}' = \overline{\textit{off}} \\
\qquad\quad \land \textit{UNCHANGED} \langle \textit{state}, \textit{incomingcall}, \textit{contacts}, \textit{buffer} \rangle \\
\textit{HangupIncoming} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{on}} \land \textit{incomingcall} = \overline{\textit{on}} \land \textit{teclado}!\textit{Pulse}(11) \\
\qquad\qquad\qquad \land \textit{incomingcall}' = \overline{\textit{off}} \\
\qquad\qquad\qquad \land\ \textit{bocina}!\textit{Off} \land \textit{timer}!\textit{Stop} \\
\qquad\qquad\qquad \land \textit{UNCHANGED} \langle \textit{state}, \textit{incall}, \textit{contacts}, \textit{buffer} \rangle \\
\textit{IncomingCall}(n) \mathrel{\widehat{=}} \textit{state} = \overline{\textit{on}} \land \textit{incomingcall} = \overline{\textit{off}} \land \textit{incall} = \overline{\textit{off}} \\
\qquad\qquad\qquad \land \textit{incomingcall}' = \overline{\textit{on}} \land \textit{display}!\textit{Replace}(\textit{IF } n \in \textit{DOM}(\textit{contacts}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{THEN } \textit{contacts}(n) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{ELSE } n) \\
\qquad\qquad\qquad\ \land \textit{timer}!\textit{Set}(1).\textit{timer}!\textit{Start} \land \textit{bocina}!\textit{On} \\
\qquad\qquad\qquad\ \land \textit{UNCHANGED} \langle \textit{state}, \textit{incall}, \textit{contacts}, \textit{buffer} \rangle \\
\textit{Alternate} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{on}} \land \textit{incomingcall} = \overline{\textit{on}} \land \textit{timer}!\textit{Timeout} \land \\
\qquad\qquad ((\textit{bocina}!\textit{state} = \overline{\textit{on}} \land \textit{bocina}!\textit{Off}) \lor \\
\qquad\qquad (\textit{bocina}!\textit{state} = \overline{\textit{off}} \land \textit{bocina}!\textit{On})) \\
\qquad\qquad \land \textit{timer}!\textit{Set}(1).\textit{timer}!\textit{Start} \\
\textit{Accept} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{on}} \land \textit{incomingcall} = \overline{\textit{on}} \land \\
\qquad\quad \textit{incomingcall}' = \overline{\textit{off}} \land \textit{incall}' = \overline{\textit{on}} \\
\qquad\quad \textit{teclado}!\textit{Pulse}(10) \land \textit{bocina}!\textit{Off} \land \textit{timer}!\textit{Stop} \\
\textit{Write}(i) \mathrel{\widehat{=}} \textit{state} = \overline{\textit{on}} \land \textit{incall} = \overline{\textit{off}} \land \textit{incomingcall} = \overline{\textit{off}} \\
\qquad\quad \land \textit{teclado}!\textit{Pulse}(i) \land \textit{buffer}' = \textit{buffer} \frown \langle i \rangle \\
\qquad\quad \land \textit{display}!\textit{DReplace}(\textit{buffer}') \land \textit{UNCHANGED} \langle \textit{state}, \textit{incall}, \textit{incomingcall}, \textit{contacts} \rangle \\
\textit{Call} \mathrel{\widehat{=}} \textit{state} = \overline{\textit{on}} \land \textit{incall} = \overline{\textit{off}} \land \textit{incomingcall} = \overline{\textit{off}} \land \textit{teclado}!\textit{Pulse}(10) \land \\
\qquad \textit{incall}' = \overline{\textit{on}} \land \textit{buffer}' = \langle\rangle \land \textit{display}!\textit{DDelete} \\
\qquad \land \textit{UNCHANGED} \langle \textit{state}, \textit{incomingcall}, \textit{contacts} \rangle \\
\textit{Next} \mathrel{\widehat{=}} \forall\, n : \textit{Seq Nat}, i \in \{0\,..\,9\} \bullet \\
\qquad \textit{Startup} \lor \textit{Abort} \lor \textit{On} \lor \textit{Delete} \lor \textit{Hangup} \lor \\
\qquad \textit{HangupIncoming} \lor \textit{IncomingCall}(n) \lor \\
\qquad \textit{Alternate} \lor \textit{Accept} \lor \textit{Write}(i) \lor \textit{Call} \\
\textit{Spec} \mathrel{\widehat{=}} \textit{Init} \land \Box[\textit{Next}]_{\textit{vars}} \land \textit{WF}_{\textit{vars}}(\textit{On} \lor \textit{Alternate})
\end{array}
$$

$$\textit{THEOREM } \textit{Spec} \Rightarrow \textit{TypeInv}$$