Exámenes teóricos y prácticos de TLA

2 de marzo de 2020

Índice

1. Ejercicios teóricos

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

   Solución Consideremos la ejecución $\langle x = 0, x = 1, x = 0, x = 1, \ldots \rangle$ y la acción $x = 0 \land x' = 2$. Si quisiéramos exigir la ejecución de dicha acción no bastaría con solicitar equidad débil pues ella aplica solamente si existe un momento en la ejecución a partir del cual la acción siempre esta habilitada, lo cual no ocurre.

   Por el contrario, equidad fuerte solo requiere que en cualquier momento de la ejecución, exista un punto a partir del cual la acción este habilitada infinitas veces, cosa que si ocurre.

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

   Solución A veces es necesario que una determinada variable pueda asumir valores de diferentes tipos. En un lenguaje tipado esto exige definir un nuevo tipo con múltiples constructores, lo cual conlleva que cada vez que se haga referencia a la variable deberá hacerse un análisis por casos. Esto torna la especificación mas larga y compleja.

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

   Solución La formula final de una especificación TLA suele tener la forma $Init \land \Box [Next]_{vars} \land Fairness_{vars}$. Dicha formula exige tres condiciones diferentes:

   a) El estado inicial de una ejecución válida debe cumplir con el predicado $Init$, pues $e \llbracket P \rrbracket \equiv e(0) \llbracket P \rrbracket$.

   b) Cualquier par de estados sucesivos debe verificar la acción $Next$, pues $e \llbracket \Box Next \rrbracket \equiv \forall i \in \mathbb{N} : e(i) \llbracket Next \rrbracket e(i+1)$.

   c) El sistema esta obligado a evolucionar si esta infinitamente habilitado para hacerlo, pues por definición equidad es $\Diamond\Box(ENABLED \langle T \rangle_v) \Rightarrow \Box\Diamond \langle T \rangle_v$ o bien $\Box\Diamond ENABLED \langle T \rangle_v \Rightarrow \Box\Diamond \langle T \rangle_v$.

Nótese que el subíndice $vars$ nos indica que también se admiten pasos intrascendentes, para poder combinar la especificación con otras.

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.

   Solución Puede lograrse por ejemplo definiendo un arreglo de la siguiente manera: $H = [i \in Nat : i \mapsto \text{INSTANCE } MODULO]$.

   COMPLETAR.

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

   Solución COMPLETAR.

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

   Solución

   - Una propiedad de seguridad es una propiedad que prohíbe pasos incorrectos. Hacemos esto generalmente indicando cuales pasos están permitidos. Por ejemplo $x = 11 \land x' = 0$ solo permite pasar de un estado donde x vale 11 al estado donde x vale 0.
   - Las propiedades de seguridad solo indican cuales son los pasos permitidos, pero no exigen que un sistema evolucione; un sistema podría satisfacer la especificación simplemente permaneciendo en el estado inicial. Las propiedades de vitalidad son las que exigen que ciertos estados sean alcanzado.
   - Las propiedades de equidad son un tipo especial de propiedades de vitalidad que no agrega restricciones a las posibles ejecuciones de un sistema.

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

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

Solución

$$\begin{aligned}
& \langle s_0, s_1, \ldots \rangle \left[\!\left[Init \wedge \Box \left[Op_1 \vee Op_2\right]_v \wedge WF_v \left(Op_1\right) \right]\!\right] \\
\equiv & \langle s_0, s_1, \ldots \rangle \left[\!\left[Init \wedge \Box \left(Op_1 \vee Op_2 \vee v' = v\right) \wedge WF_v \left(Op_1\right) \right]\!\right] \\
\equiv & \wedge \langle s_0, s_1, \ldots \rangle \left[\!\left[Init \right]\!\right] \\
& \wedge \langle s_0, s_1, \ldots \rangle \left[\!\left[\Box \left(Op_1 \vee Op_2 \vee v' = v\right) \right]\!\right] \\
& \wedge \langle s_0, s_1, \ldots \rangle \left[\!\left[WF_v \left(Op_1\right) \right]\!\right] \\
\equiv & \wedge s_0 \left[\!\left[Init \right]\!\right] \\
& \wedge \forall i \in \mathbb{N} : s_i \left[\!\left[Op_1 \vee Op_2 \vee v' = v \right]\!\right] s_{i+1} \\
& \wedge \langle s_0, s_1, \ldots \rangle \left[\!\left[WF_v \left(Op_1\right) \right]\!\right] \\
\equiv & \wedge Init \left[\forall v \in Vars : v \leftarrow s_0 \left(v\right)\right] \\
& \wedge \forall i \in \mathbb{N} : s_i \left[\!\left[Op_1 \right]\!\right] s_{i+1} \vee s_i \left[\!\left[Op_2 \right]\!\right] s_{i+1} \vee s_i \left[\!\left[v' = v \right]\!\right] s_{i+1} \\
& \wedge \langle s_0, s_1, \ldots \rangle \left[\!\left[WF_v \left(Op_1\right) \right]\!\right] \\
\equiv & \wedge Init \left[\forall v \in Vars : v \leftarrow s_0 \left(v\right)\right] \\
& \wedge \forall i \in \mathbb{N} : \begin{array}{ll} \vee & Op_1 \left[\forall v \in Vars : v \leftarrow s_i \left(v\right), v' \leftarrow s_{i+1} \left(v\right)\right] \\ \vee & Op_2 \left[\forall v \in Vars : v \leftarrow s_i \left(v\right), v' \leftarrow s_{i+1} \left(v\right)\right] \\ \vee & s_i \left(v\right) = s_{i+1} \left(v\right) \end{array} \\
& \wedge \langle s_0, s_1, \ldots \rangle \left[\!\left[WF_v \left(Op_1\right) \right]\!\right]
\end{aligned}$$

8. Respecto del lenguaje TLA escriba el significado formal de $\Box \left[A\right]_v$ con respecto a una ejecución e, donde A es una acción y v es una variable.

Solución

$$\forall i \in \mathbb{N} : A \left(e \left(i\right), e \left(i+1\right)\right) \vee e \left(i\right) \left(v\right) = e \left(i+1\right) \left(v\right)$$

9. Explique la razón por la cual Lamport sugiere escribir la vitalidad de un sistema con fórmulas de equidad.

   Solución Una propiedad de vitalidad no debería agregar restricciones a un sistema, para ello deben utilizarse propiedades de seguridad. Sin embargo si se admiten propiedades de vitalidad arbitrarias es posible que estas agreguen restricciones.

   Por ejemplo un sistema con propiedades de seguridad que exigen estado inicial $x = 0$ y única transición $x' = x + 1$ admiten ejecuciones como $\langle x = 0 \rangle$ o $\langle x = 0, x = 1, \ldots \rangle$. Si ahora agregamos la propiedad de vitalidad «si en algún estado x vale 1 entonces eventualmente se debe alcanzar un estado donde x vale 0», entonces estamos agregando nuevas restricciones al sistema.

   Lamport sugiere escribir las propiedades de vitalidad como propiedades de equidad, para así evitar este conflicto.

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

    Solución El teorema de Alpern-Schneider dice que toda propiedad es el resultado de una propiedad de seguridad con otra de vitalidad. Es por eso que la formula final de una especificación suele tener la forma $Init \land \Box [Next]_{vars} \land Fairness_{vars}$.

11. 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.

    estado Sea Var el conjunto de todos los nombres posibles de variables y Val el conjunto de todos los valores que estas pueden llegar a tomar, luego un estado es una función $s : Var \to Val$.

    ejecución Secuencia infinita de estados.

    propiedad Conjunto de todas las ejecuciones que la satisfacen.

    sistema Conjunto de todas sus ejecuciones posibles.

seguridad Diremos que una propiedad P es de seguridad respecto del conjunto de estados E si y solo si para cualquier ejecución $e \in E^\infty$ ocurre:

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

vitalidad Diremos que una propiedad P es de vitalidad respecto del conjunto de estados E si y solo si:

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

equidad débil Sea T un predicado que depende de dos estados (transición) luego:

$$\begin{aligned} WF_v(T) \triangleq & \quad \Diamond\Box\left(ENABLED\left\langle T\right\rangle_v\right) \Rightarrow \Box\Diamond\left\langle T\right\rangle_v (a) \\ \equiv & \quad \Box\left(\Box ENABLED\left\langle T\right\rangle_v \Rightarrow \Diamond\left\langle T\right\rangle_v\right) (b) \\ \equiv & \quad \Box\Diamond\left(\neg ENABLED\left\langle T\right\rangle_v\right) \vee \Box\Diamond\left\langle T\right\rangle_v (c) \end{aligned}$$

a) Si en algún momento de una ejecución, T queda habilitada para siempre, entonces la transición T ocurre infinitas veces.

b) Siempre se da el caso de que, si T esta habilitada por siempre, entonces la transición T eventualmente ocurrirá.

c) O bien T esta infinitamente a menudo deshabilitada, o bien la transición T ocurre infinitas veces.

equidad fuerte Sea T un predicado que depende de dos estados (transición) luego:

$$\begin{aligned} SF_v(T) \triangleq & \quad \Box\Diamond ENABLED\left\langle T\right\rangle_v \Rightarrow \Box\Diamond\left\langle T\right\rangle_v (a) \\ \equiv & \quad \Diamond\Box\left(\neg ENABLED\left\langle T\right\rangle_v\right) \vee \Box\Diamond\left\langle T\right\rangle_v (b) \end{aligned}$$

a) Si T esta infinitamente a menudo habilitada, entonces la transición T ocurre infinitas veces.

b) O bien T esta eventualmente deshabilitada para siempre, o bien la transición T ocurre infinitas veces.

máquina cerrada COMPLETAR.

- Teorema de Alpern-Schneider: Toda propiedad es el resultado de una propiedad de seguridad con otra de vitalidad.
- Teorema de Abadi-Lamport: Si las propiedades de vitalidad se escriben como propiedades de equidad, se obtienen maquinas cerradas.

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

Solución La sentencia EXTENDS $MODULO$ simplemente agrega las definiciones del modulo $MODULO$ al módulo actual. Por ejemplo si consideramos el siguiente modulo

MODULE UNO
VARIABLE $v1$
$UNOInit \triangleq v1 = 0$

entonces los siguientes dos módulos son iguales:

MODULE DOS
VARIABLES $v1, v2$
$UNOInit \triangleq v1 = 0$
$DOSInit \triangleq v2 = 10$

MODULE $TRES$
EXTENDS UNO
VARIABLE $v2$
$DOSInit \triangleq v2 = 10$

La sentencia INSTANCE $MODULO$ WITH $x \leftarrow a$ permite crear varias instancias del mismo modulo utilizando substitución. Por ejemplo en el siguiente modulo

MODULE $CUATRO$
VARIABLES $v1, v2, i1, i2$
$DOSInit \triangleq v2 = 10$
$i1 \triangleq$INSTANCE UNO WITH $v1 \leftarrow v2$
$i2 \triangleq$INSTANCE UNO

se agregan las variables $i1.v2$ y $i2.v1$, y las definiciones $i1!UNOInit \triangleq v2 = 0$ y $i1!UNOInit \triangleq v1 = 0$.

13. 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.

Solución Sea Var el conjunto de todos los nombres posibles de variables y Val el conjunto de todos los valores que estas pueden llegar a tomar, luego un estado es una función $s : Var \to Val$. Es decir, que si un sistema solo utiliza las variables $x_1, \ldots, x_n$, un estado asigna valores no solo a dichos identificadores, sino también a cualquier otro; es por ello que decimos que los estados son estados del universo.

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. Por ejemplo consideremos los sistemas M y N con estados iniciales $x = 0$ e $y = 0$ respectivamente y cuyas únicas transiciones son $x' = x + 1$ e $y' = y + 1$. Para el sistema que resulta de componer M y N, una ejecución posible esta dada por la siguiente sucesión de estados:

$$
\begin{aligned}
&s_0 : Var \to Val / s_0(x) = 0 \land s_0(y) = 0 \land s_0(z) = 77 \land \ldots \\
&s_1 : Var \to Val / s_0(x) = 1 \land s_0(y) = 1 \land s_0(z) = \text{""} \land \ldots \\
&\vdots
\end{aligned}
$$

Si hubiéramos restringido los estados solo a las variables intervinientes ($s : \{x\} \to Val$ para M y $s : \{y\} \to Val$ para N), entonces dicha ejecución no seria valida ni para M ni para N.

14. Enuncie la regla práctica que indica cuándo se debe usar equidad fuerte y no equidad débil.

Solución Siempre debe preferirse equidad débil por sobre equidad fuerte si no hay un motivo importante por el cual la segunda opción sea la mejor. Equidad débil y fuerte son son equivalentes para una acción, si siempre que esta está habilitada solo puede deshabilitarse al ser ejecutada.

2. Ejercicios prácticos

2.1. Brazo mecánico

2.1.1. Requerimientos

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 derención. 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).

2.1.2. Designaciones

2.1.3. Conocimiento de dominio

2.1.4. Especificación

2.2. Tanques

2.2.1. Requerimientos

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

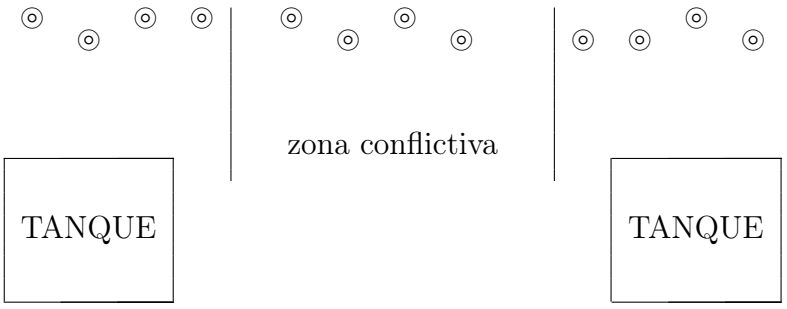
adecuado. La tripulación, leyendo estos datos, manejará el tanque y el cañón (es decir el sistema no conduce el tanque ni realiza los disparos).

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.



2.2.2. Designaciones

2.2.3. Conocimiento de dominio

2.2.4. Especificación

2.3. Servidor web

2.3.1. Requerimientos

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 simultáneamente.

2.3.2. Designaciones

2.3.3. Conocimiento de dominio

2.3.4. Especificación

2.4. Protocolo CSMA/CD

2.4.1. Requerimientos

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, esperan 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 libreo no, enviar un mensaje, comunicar que un mensaje se transmitió, comunicar a todas las terminales que hay colisión.

2.4.2. Designaciones

2.4.3. Conocimiento de dominio

2.4.4. Especificación

2.5. Recursos compartidos

2.5.1. Requerimientos

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, amenos que el primer proceso sea el único en el sistema.

2.5.2. Designaciones

2.5.3. Conocimiento de dominio

2.5.4. Especificación

2.6. Electroencefalogramas

2.6.1. Requerimientos

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.

2.6.2. Designaciones

2.6.3. Conocimiento de dominio

MODULE $DK\_ELECTRODOS$

EXTENDS $Naturals$
VARIABLE $valores$
$DKETypeInv \triangleq valores \in [1..10 \to Nat \cup \{\text{“none”}\}]$

$DKEInit \triangleq valores = [1..10 \mapsto \text{“none”}]$
$DKENext\,(i) \triangleq valores' = [valores \text{ EXCEPT } ![i] = CHOOSE\, x : x \in Nat \cup \{\text{“none”}\}]$
$DKESpec \triangleq DKEInit \wedge \Box\,[\exists i \in Nat : DKENext\,(i)]_{valores}$

THEOREM $DKESpec \Rightarrow \Box DKETypeInv$

MODULE $DK\_IMPRESORA$

EXTENDS $Naturals$
VARIABLES $imprimiendo, valor$
$IMPRESORATypeInv \triangleq valor \in Nat \wedge imprimiendo \in \{\text{“yes”}, \text{“no”}\}$

$IMPRESORAInit \triangleq imprimiendo = \text{“no”} \wedge valor \in Nat$
$Imprimir\,(x) \triangleq imprimiendo = \text{“no”} \wedge valor' = x \wedge imprimiendo' = \text{“yes”}$
$Detener \triangleq imprimiendo = \text{“yes”} \wedge imprimiendo' = \text{“no”}$
$IMPRESORANext \triangleq (\exists x \in Nat : Imprimir\,(x)) \vee Detener$
$IMPRESORASpec \triangleq IMPRESORAInit \wedge \Box\,[IMPRESORANext]_{\langle imprimiendo, valor\rangle}$

THEOREM $IMPRESORASpec \Rightarrow \Box IMPRESORATypeInv$

2.6.4. Especificación

MODULE $SISTEMA$

EXTENDS $TIMER$

VARIABLES $muestras, actual, estado$

$electrodos \triangleq$ INSTANCE $DK\_ELECTRODOS$

$impresora \triangleq$ INSTANCE $DK\_IMPRESORA$

$SISTEMATypeInv \triangleq estado \in \{\text{“none”}, \text{“muestreando”}, \text{“imprimiendo”}\}$

$$\begin{aligned} muestrasStart \triangleq\ & \wedge Timeout \wedge estado = \text{“none”} \\ & \wedge actual' = 0 \wedge estado' = \text{“muestreando”} \\ & \wedge \text{UNCHANGED } muestras \end{aligned}$$

$$\begin{aligned} siguienteMuestra \triangleq\ & \wedge estado = \text{“muestreando”} \\ & \wedge actual \neq 11 \\ & \wedge actual' = actual + 1 \\ & \wedge muestrear\,(actual) \\ & \wedge \text{UNCHANGED } estado \end{aligned}$$

$$muestrear\,(i) \triangleq muestras' = [muestras \text{ EXCEPT } ![i] = electrodos.valores\,[i]]$$

$$muestrasNext \triangleq muestraStart \vee siguienteMuestra \vee impresorStart$$

$$\begin{aligned} impresorStart \triangleq\ & \wedge actual = 11 \wedge estado = \text{“muestreando”} \\ & \wedge actual' = 0 \wedge estado' = \text{“imprimiendo”} \\ & \wedge \text{UNCHANGED } muestras \end{aligned}$$

$$\begin{aligned} siguienteImpresion \triangleq\ & \wedge estado = \text{“imprimiendo”} \\ & \wedge actual \neq 11 \\ & \wedge actual' = actual + 1 \\ & \wedge impresora.Imprimir\,(muestras\,[actual]) \\ & \wedge \text{UNCHANGED } \langle muestras, estado \rangle \end{aligned}$$

$$\begin{aligned} impresorStop \triangleq estado = \ & \wedge \text{“imprimiendo”} \wedge actual = 11 \\ & \wedge estado' = \text{“none”} \wedge Start \\ & \wedge \text{UNCHANGED } \langle actual, muestras \rangle \end{aligned}$$

$$impresorNext \triangleq siguienteImpresion \vee impresorStop$$

$$Set\&Start \triangleq running = \text{“}no\text{”} \wedge limit' = ? \wedge time' = now \wedge running' = \text{“}yes\text{”}$$

$$SISTEMAInit \triangleq muestras = [1..10 \mapsto \text{“none”}] \wedge estado = \text{“none”} \wedge Set\&Start$$

$$SISTEMANext \triangleq muestrasNext \vee impresorNext$$

$$\begin{aligned} SISTEMASpec \triangleq\ & \wedge SISTEMAInit \\ & \wedge \Box\,[SISTEMANext]_{\langle muestras, actual, estado \rangle} \\ & \wedge WF_{\langle muestras, actual, estado \rangle}\,(SISTEMANext) \end{aligned}$$

THEOREM $SISTEMASpec \Rightarrow \Box SISTEMATypeInv$

2.7. Sistema de memoria

2.7.1. Requerimientos

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.

2.7.2. Designaciones

2.7.3. Conocimiento de dominio

2.7.4. Especificación

2.8. Buffers

2.8.1. Requerimientos

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 *buffers*, 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.

2.8.2. Designaciones

2.8.3. Conocimiento de dominio

2.8.4. Especificación

2.9. Caldera

2.9.1. Requerimientos

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 disminuyese 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.

2.9.2. Designaciones

2.9.3. Conocimiento de dominio

2.9.4. Especificación

2.10. Celular

2.10.1. Requerimientos

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

El celular se enciende luego de presionar el botón de «cortar» dos segundos. Mientras no se este realizando ninguna llamada se puede presionar números del teclado y estos aparecerán en el display. Si se presiona el botón 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 botón de «cortar» se podrá volver a marcar números, 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.

2.10.2. Designaciones

2.10.3. Conocimiento de dominio

2.10.4. Especificación

2.11. Router

2.11.1. Requerimientos

Un *router* posee N interfaces de red. El sistema operativo del *router* debe pasar los datos que llegan por una interfaz a otra, según la configuración del equipo (una tabla que indica los pares de interfaces (a, b), tales que se deben comunicar los datos que llegan a a únicamente hacia b). Por cuestiones de eficiencia, los datos primero se copian en un *buffer*, único para todo el sistema, con capacidad para M datos. Si el *buffer* no tiene espacio el dato se pierde. No debe permitirse que se pierdan infinitos datos que llegan a través de una misma interfaz. Cada interfaz de red genera una interrupción cada vez que

recibe un dato; el *firmware* de la interfaz almacena el dato en cierta porción de la memoria del equipo destinada para esa interfaz.

Asuma que por cuestiones de eficiencia, el sistema debe primero recibir una interrupción y mas adelante ejecutar la operación (interna) asociada.

2.11.2. Designaciones

2.11.3. Conocimiento de dominio

MODULE $DK\_INTERFAZ$

EXTENDS $Naturals$
VARIABLES $memoria, router$
CONSTANT id, N
ASSUME $N \in Nat \land id \in 1..N$
$DKITypeInv \triangleq memoria \in Nat$

$DKIInit \triangleq memoria = 0$

$$DKISend(x) \triangleq \begin{array}{l} \land router[id].status = \text{“free”} \\ \land router' = [router \text{ EXCEPT } ![id] = [value \mapsto x, status \mapsto \text{“sending”}]] \\ \land \text{UNCHANGED } memoria \end{array}$$

$$DKIReceive \triangleq \begin{array}{l} \land router[id].status = \text{“receiving”} \\ \land router' = [router \text{ EXCEPT } ![id].status = \text{“free”}] \\ \land memoria' = router[id].value \end{array}$$

$DKINext \triangleq DKIReceive \lor \exists i \in Nat : DKISend(i)$

$DKISpec \triangleq DKIInit \land \Box[DKINext]_{\langle memoria, router \rangle} \land WF_{\langle memoria, router \rangle}(DKINext)$

THEOREM $DKISpec \Rightarrow \Box DKITypeInv$

2.11.4. Especificación

MODULE $ROUTER$

EXTENDS $NATURALS, SEQUENCES$

VARIABLES $interfaces, buffer, router$

CONSTANTS $N, M, table$

ASUME $N, M \in Nat \land table \in [Nat \to Nat]$

$ROUTERTypeInv \triangleq buffer \in Seq([val : Nat, dst : Nat])$

$interfacesInit \triangleq interfaces = [1..N \mapsto \text{INSTANCE } DK\_INTERFAZ]$

$bufferInit \triangleq buffer = \langle\rangle$

$routerInit \triangleq router = [1..N \mapsto [value \mapsto 0, status \mapsto \text{“free”}]]$

$$
\begin{aligned}
Send(i) \triangleq\ & \land router[i].status = \text{“sending”} \\
& \land Len(buffer) < M \\
& \land buffer' = Append(buffer, [val \mapsto router[i].value, dst \mapsto table[i]]) \\
& \land router' = [router \text{ EXCEPT } ![i].status = \text{“free”}] \\
& \land \text{UNCHANGED } interfaces
\end{aligned}
$$

$$
\begin{aligned}
Receive \triangleq\ & \land Len(buffer) \neq 0 \\
& \land router[Head(buffer).dst].status = \text{“free”} \\
& \land buffer' = Tail(buffer) \\
& \land router' = [router \text{ EXCEPT } ![Head(buffer).dst] = [status \mapsto \text{“receiving”}, value \mapsto Head(buffer).val]] \\
& \land \text{UNCHANGED } interfaces
\end{aligned}
$$

$routerNext \triangleq Receive \lor \exists i \in Nat : Send(i)$

$ROUTERInit \triangleq interfacesInit \land bufferInit \land routerInit \land \forall i \in 1..N : interfaces[i]!DKIInit$

$ROUTERNext \triangleq routerNext \lor \exists i \in i..N : interfaces[i]!DKINext$

$$
\begin{aligned}
ROUTERSpec \triangleq\ & \land ROUTERInit \\
& \land \Box[ROUTERNext]_{\langle interfaces, buffer, router\rangle} \\
& \land WF_{\langle interfaces, buffer, router\rangle}(ROUTERNext)
\end{aligned}
$$

THEOREM $ROUTERSpec \Rightarrow \Box ROUTERTypeInv$