CTAD CON PASO DE MENSAJES

Parking con CSP:

C-TAD: Parking
OPERACIONES:
ACCION: entrar
ACCION: salir

DOMINIO
TIPO: Parking = $\mathbb{N}$
DONDE: CAP = $\mathbb{N}$
INVARIANTE: $0 \leq self \leq CAP$
INICIAL: self = 0

CPRE: self < CAP
entrar()
POST: self = $self^{PRE}$+1

CPRE: cierto
salir()
POST: self = $self^{PRE} - 1$

Almacén de un dato con CSP:

C-TAD: Almacen1Dato
OPERACIONES:
ACCION: almacenar: Tipo_Dato[e]
ACCION: extraer: Tipo_Dato[s]

DOMINIO
TIPO: Almacen1Dato = (Dato: Tipo_Dato x HayDato: B)
INVARIANTE: cierto
INICIAL: ¬self.HayDato

CPRE: ¬self.HayDato
almacenar(e)
POST: self.Dato = e ∧ self.HayDato

CPRE: self.HayDato
extraer(s)
POST: s = $self^{pre}$.Dato ∧ ¬self.HayDato

Dada la siguiente especificación formal del recurso compartido de Lectores/Escritores. Se pide: Completar la implementación de este recurso mediante paso de mensajes:

TIPO: $LE = (l : \mathbb{Z} \times e : \mathbb{Z})$
INVARIANTE: $\forall r \in LE \bullet\ r.e \geq 0 \land r.l \geq 0 \land r.e \leq 1 \land$
$((r.e > 0 \Rightarrow r.l = 0) \land (r.l > 0 \Rightarrow r.e = 0))$
INICIAL: $\text{self} = (0, 0)$

CPRE: $\text{self}.e = 0$
inicioLeer()
POST: $\text{self} = (\text{self}^{pre}.l + 1, \text{self}^{pre}.e)$
CPRE: $\text{self}.e = 0 \land \text{self}.l = 0$
inicioEscribir()
POST: $\text{self} = (\text{self}^{pre}.l, \text{self}^{pre}.e + 1)$
CPRE: *cierto*
finLeer()
POST: $\text{self} = (\text{self}^{pre}.l - 1, \text{self}^{pre}.e)$
CPRE: *cierto*
finEscribir()
POST: $\text{self} = (\text{self}^{pre}.l, \text{self}^{pre}.e - 1)$

Se pide implementar el siguiente CTAD usando como mecanismo de sincronización el paso de mensajes:

C-TAD MultiCont

OPERACIONES
ACCIÓN inc: N[e]
ACCIÓN dec: N[e]

SEMÁNTICA
DOMINIO:
TIPO: MultiCont = N
INVARIANTE: $0 \leq self \wedge self \leq N$

INICIAL: self = 0

PRE: $n > 0 \wedge n < N/2$
CPRE: $self + n \leq N$
inc(n)
POST: $self = self^{PRE} + n$

PRE: $n > 0 \wedge n < N/2$
CPRE: $n \leq self$
dec(n)
POST: $self = self^{PRE} - n$

Dada la siguiente especificación formal de un recurso compartido *Peligro*. Se pide: Completar la implementación de este recurso mediante paso de mensajes:

C-TAD *Peligro*
OPERACIONES
ACCIÓN *avisarPeligro*: $\mathbb{B}[e]$
ACCIÓN *entrar*:
ACCIÓN *salir*:

SEMÁNTICA
DOMINIO:
TIPO: $Peligro = (p : \mathbb{B} \times o : \mathbb{N})$
INICIAL: $\text{self} = (false, 0)$

INVARIANTE: $\text{self}.o \leq 5$

CPRE: Cierto
***avisarPeligro*(x)**
POST: $\text{self}.p = x \wedge \text{self}.o = \text{self}^{pre}.o$

CPRE: $\neg\text{self}.p \wedge \text{self}.o < 5$
***entrar*()**
POST: $\neg\text{self}.p \wedge \text{self}.o = \text{self}^{pre}.o + 1$

CPRE: $\text{self}.o > 0$
***salir*()**
POST: $\text{self}.p = \text{self}^{pre}.p \wedge \text{self}.o = \text{self}^{pre}.o - 1$

A continuación mostramos la especificación formal de un recurso gestor *Misil*. Se pide: Completar la implementación de este recurso mediante paso de mensajes. NOTA: Podéis usar el método Math.abs(x) para calcular el valor absoluto de un número, |x|:

C-TAD Misil
OPERACIONES
ACCIÓN notificar: $\mathbb{Z}[e]$
ACCIÓN detectarDesviacion: $TUmbral[e] \times \mathbb{Z}[s]$

SEMÁNTICA
DOMINIO:
TIPO: $TUmbral = [0..100]$
TIPO: $Misil = \mathbb{Z}$

INICIAL: self $= 0$

CPRE: *Cierto*
notificar(desv)
POST: self $= desv$

CPRE: $|$ self $|> umbral$
detectarDesviacion(umbral,d)
POST: $self = \text{self}^{pre} \wedge d = \text{self}^{pre}$

El siguiente recurso compartido forma parte de un algoritmo paralelo de ordenación por mezcla. Permite mezclar dos secuencias ordenadas de números enteros para formar una única secuencia ordenada. En este recurso interactúan solo tres procesos: dos productores (izquierdo y derecho) que van pasando números de sus secuencias de uno en uno y un consumidor que va extrayendo los números en orden.

C-TAD: OrdMezcla

OPERACIONES:
ACCIÓN: insertar: Lado[e] x Z [e]
ACCIÓN: extraerMenor: Z[s]

SEMÁNTICA:
DOMINIO:
TIPO: OrdMezcla = { haydato: Lado → B x dato: Lado → Z }
TIPO: Lado = Izda | Dcha

INICIAL: $\forall i \in Lado \cdot \neg$self.hayDato(i)

CPRE: ¬self.hayDato(l)
insertar(l, d)
POST: $self^{PRE}$ = (hay, dat) ∧ self = <hay ⊕ {l → Cierto} ∧ dat ⊕ {l → d}>

CPRE: self.hayDato(Izda) ∧ self.hayDato(Dcha)
extraerMenor(min)
POST: $self^{PRE}$ = (hay, dat) ∧
(dat(Izda) ≤ dat(Dcha) ∧ min ⇒ dat(Izda) ∧ self = < hay ⊕ {Izda → Falso}, dat>) ∧
(dat(Dcha) ≤ dat(Izda) ∧ min ⇒ dat(Dcha) ∧ self = < hay ⊕ {Dcha → Falso}, dat>)

La operación insertar(lado, dato) inserta dato en el lado correspondiente, bloqueando si ese hueco no está disponible. Cuando hay datos de ambas secuencias la operación extraerMenor tomará el menor de ambos y permitirá que se añada un nuevo dato de la secuencia correspondiente. Por concisión, no hemos considerado el problema de la terminación de las secuencias.

Se pide: Completar la implementación de este recurso compartido mediante paso de mensajes.