UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

VERIFICACIÓN DE ALGORITMOS CONCURRENTES

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:
SANDOVAL GRAJEDA ISRAEL

TUTOR:
DR. HÉCTOR BENÍTEZ PÉREZ
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y SISTEMAS
UNAM

MÉXICO, D. F. MARZO 2016

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Actualmente existen metodologías y procedimientos ampliamente conocidos acerca de la forma en que se puede atacar un problema por medio de programación, siempre teniendo en mente que los sistemas elaborados serán ejecutados en una máquina basada en el modelo de Von Neumann, cuya característica principal es la ejecución secuencial de las instrucciones.

El incremento de la velocidad de los procesadores de acuerdo a la ley de Moore fue una constante durante muchos años en los cuales la velocidad de ejecución de un programa aumentaba cuando se compraba un equipo con un procesador más moderno. Sin embargo en la década pasada los grandes fabricantes de microprocesadores cambiaron su enfoque con respecto al diseño de los microprocesadores y en lugar de aumentar los ciclos de reloj recurrieron a una arquitectura multicore, donde el trabajo de procesamiento se reparte pero los procesadores no son necesariamente mas rápidos.

Este cambio de arquitectura trajo como consecuencias que los programas de computadora no aumentaran su velocidad de ejecución y que los recursos de hardware fueran subutilizados. Sutter, en su artículo "the free lunch is over" [Sut05] hace un análisis de las repercusiones de las arquitecturas multicore, planteando la necesidad de recurrir a la Programación concurrente para elaborar sistemas de cómputo compuestos por una serie de procesos secuenciales ejecutándose de manera simultánea, interactuando y comunicándose para realizar una tarea en común para aprovechar al máximo la nueva arquitectura y lograr ejecuciones más rápidas que en entornos monoprocesador.

Sutter también plantea que construir sistemas concurrentes es una tarea difícil, entre otras cosas, porque implica cambiar los modelos mentales heredados de la programación secuencial, empezando con el análisis del problema.

Existen varios enfoques para realizar programación concurrente para entornos multicore, sin

embargo, aun no existe un consenso respecto a cual es el camino a elegir para crear una metodología que resuelva problemas con concurrencia e implementar las soluciones en algún lenguaje de programación cuya ejecución brinde los resultados esperados.

Como parte del enfoque utilizado en este documento se propone tomar como base el proceso de resolución de problemas con paralelismo planteado por Pancake y Bergmark[PB90] donde se describen y analizan los pasos necesarios para llegar a un programa concurrente escrito en algún lenguaje de programación que resuelva un problema dado a partir de representaciones de la solución con diferentes niveles de abstracción.

En el presente trabajo se propone un método para definir algoritmos basados en procesos secuenciales concurrentes como parte del proceso antes mencionado. Sin embargo, realizar esta labor no solo implica establecer instrucciones con pasos que dependen unos de otros, sino intentar definir un enfoque de pensamiento que permita a programadores realizar definiciones que les permitan llegar a dichos algoritmos aunque no tengan mas información que la especificación del problema.

1.1. Objetivo

Actualmente la programación para entornos multiprocesador es realizada por especialistas que han recopilado a través de su experiencia ciertas prácticas y procedimientos propios que les han ayudado a definir mejor sus soluciones y dar una mayor confiabilidad a sus sistemas, sin embargo, será una tarea difícil dar mantenimiento a ese tipo de sistemas si su creador no está disponible.

Siguiendo el proceso de resolución de problemas con paralelismo antes mencionado, la solución a un problema que se pretende atacar con programación concurrente puede plantearse en términos de un modelo matemático o por medio de una descripción gráfica de que pueda proporcionar las salidas correctas ante las entradas planteadas e ir transformando este modelo en otros mas abstractos hasta que pueda ser definido como un programa de computadora, de tal forma que cada paso estará documentado y pueda ser entendido por cualquier persona familiarizada con el proceso.

La descripción del sistema en términos algorítmicos es el paso intermedio entre la representación de alto nivel de una solución y un programa en algún lenguaje de programación. Aunque existen

enfoques que llegan a la implementación de una solución sin utilizar una representación algorítmica, su definición puede ser muy útil para realizar un análisis de dicha solución y además proporciona un mapeo sencillo hacia la elaboración de un programa de computadora.

El presente trabajo tiene como propósito abordar el problema de la programación concurrente definiendo un método para generar una representación algorítmica de una solución a un problema dado, utilizando como base el proceso de resolución de problemas con paralelismo.

1.2. Metas

Verificar características deseables o no deseables de sistemas concurrentes a través de su representación en términos de álgebra de procesos.

Representar soluciones a problemas dados en términos de álgebra de procesos para transformarlas en otras de tipo algorítmico, es decir, en términos de procesos secuenciales cuyas interacciones se verán reflejadas en sus definiciones.

Dar los primeros pasos en la formalización del proceso de resolución de problemas con paralelismo planteado por Pancake y Bergmark con herramientas conceptuales que permitan definir soluciones basadas en procesos concurrentes y llevar dichas soluciones a diferentes niveles de abstracción hasta obtener sistemas concurrentes para arquitecturas multicore.

1.3. Alcances

Cualquier característica de una solución puede ser verificada siempre y cuando pueda ser expresada en términos del álgebra de procesos utilizada en el presente trabajo.

La definición un algoritmo concurrente se realiza a partir un conjunto de algoritmos secuenciales cuya interacción y funcionamiento resuelven un problema dado, si embargo no se realizan especificaciones particulares acerca de su implementación en algún lenguaje de programación.

El proceso de problemas con paralelismo no termina con la definición de una representación de la solución que pueda ser programada en un lenguaje de programación. El presente trabajo solo

pretende abarcar los dos primeros pasos y sentar bases que puedan servir para la especificación de las acciones a seguir en las siguientes etapas.

1.4. Logros

Se han generado algoritmos basados en procesos concurrentes tomando como base el álgebra de procesos definida en el presente trabajo.

A través del uso de patrones de diseño para software en paralelo[OA10] y de los conceptos asociados al álgebra de procesos se logró una representación algebraica de soluciones a problemas dados.

Mediante un proceso de derivación algebraica se logró llegar a una representación algorítmica que garantiza una ejecución libre de deadlock, la cual fue obtenida a partir del análisis del proceso de derivación.

1.5. Organización

El presente trabajo se divide en las siguientes secciones: en el capítulo 2 se explicará la forma en que el álgebra de procesos ha evolucionado a través de su formalización con bases lógicas. Este análisis aporta elementos que fueron utilizados como base para llegar a las conclusiones presentadas.

Posteriormente en el capítulos 3 se presentarán algunos enfoques actuales de álgebra de procesos, así como la forma en que plantean la solución de problemas con procesos concurrentes. A continuación se planteará en el capítulo 4 el problema de la programación con procesos concurrentes de manera formal y se realizará un análisis del mismo.

En el capítulo 5 se detallará el proceso de solución utilizado para obtener representaciones algorítmicas a partir de un problema dado, delimitado por un marco de trabajo definido y en el capítulo 6 se hará un análisis al respecto. Finalmente, el capítulo 7 aportará las conclusiones y el posible trabajo futuro con respecto a este tema.

2. Evolución del álgebra de procesos

El propósito de este capítulo es realizar un análisis de distintos enfoques que han sido utilizados en la representación de sistemas concurrentes. Es importante mencionar que la palabra sistema bajo este contexto se utiliza en su forma mas amplia y que los modelos que sirvieron como base para el establecimiento de los conceptos utilizados en el presente trabajo van desde máquinas con dispositivos electrónicos y mecánicos hasta representaciones abstractas por medio de teoría de autómatas[Koz97].

2.1. Enfoques iniciales para sistemas concurrentes

Desde hace varias décadas se han publicado trabajos que analizan el comportamiento de procesos concurrentes así como sus interacciones, es decir, la forma en que se pueden comunicar y colaborar, dando origen a los modelos de comunicación por memoria compartida y paso de mensajes. A continuación se analizan los trabajos de algunos de los pioneros en el álgebra de procesos y los conceptos mas importantes que sentaron las bases para su formalización.

2.1.1. Primeras representaciones

El enfoque de C.A.R. Hoare [Hoa78] llamado CSP plantea como objetivos el entendimiento de problemas de programación donde se realizan acciones simultáneas, no solo en computadoras sino en otras máquinas mas simples como las que dispensan golosinas al depositar algunas monedas, así como la aplicación de los conceptos en un lenguaje de programación definido para procesos concurrentes.

Un sistema como los antes mencionados eran modelados desde la perspectiva de la máquina y del usuario, es decir, ambos conforman al sistema y cada uno realiza tareas individuales hasta que necesitan tener alguna interacción para cumplir un propósito, de la misma forma en que se definen sistemas concurrentes actualmente.

Las entradas en este modelo son determinadas por un alfabeto, es decir, un conjunto de instrucciones previamente definidas, donde se puedan incluir aspectos fundamentales del sistema a modelar por medio de palabras, donde cada una representa un evento, es decir, las acciones que realiza el sistema o que son importantes para su funcionamiento. La definición rigurosa de este conjunto servirá como base para realizar una especificación algebraica de un sistema concurrente, es decir, a partir de la observación de los procesos se establece una representación algebraica de un sistema concurrente, donde el elemento base se llama acción.

Una acción provoca la evolución de un sistema hacia un estado nuevo y se denota como $x \rightarrow P$, donde x es un evento y P es un proceso. Esta definición permite especificar la ejecución de las acciones de un sistema y una secuencia de transiciones consecutivas que terminan con $\surd$ (success) es un indicador de una ejecución exitosa.

El comportamiento de un sistema bajo este enfoque se puede capturar a través de su traza o bien pueden ser representados de forma gráfica como diagramas de árbol pero con el enfoque de las máquinas de estado, como se muestra en la figura 2.1.

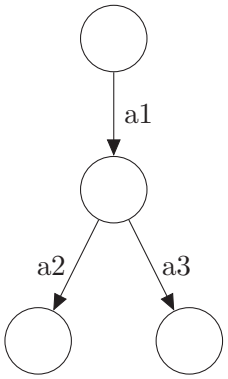

Figura 2.1.: Ejemplo del proceso $P = a1 \rightarrow (a2|a3)$

Los diagramas de árbol pueden ser útiles para entender los procesos pero tiene dos desventajas,

la primera es que un proceso puede tener mas de una representación gráfica, lo que hace difícil notar cuando dos procesos son equivalentes (limitación que se resuelve en el álgebra de procesos) y la segunda es que solo se pueden representar procesos con unos pocos pasos, ya que los diagramas tienden a crecer rápidamente, sin embargo, los diagramas son utilizados como elementos ilustrativos aunque la forma de representar procesos se hace por medios algebraicos.

Por otro lado, Robin Milner plantea otra forma de ver un sistema concurrente a través del enfoque llamado CSS [Mil82] definido como un tipo de cálculo parcialmente informal para sistemas concurrentes, el cual pretende ser flexible en su manipulación, rico en expresividad y bien fundamentado por un conjunto específico de ideas. Su objetivo principal es describir sistemas concurrentes y poder manipularlos a través de ideas simples que servirán para su implementación a través de cierta manipulación matemática, así como para verificar propiedades de este tipo de sistemas.

En este caso, el elemento principal que será representado se llama agente y su comportamiento es muy similar al de un autómata [Koz97] que acepta ciertos elemento. Milner llega también a las representaciones de árbol etiquetado, llamándolas RST como lo muestra la figura 2.2. En el ejemplo 2.2 los nodos se encuentran bajo β y γ. El símbolo $+$, es una operación binaria representada por un árbol con la raíz y dos ramas, en la figura 2.2 se puede ver bajo α. Si es necesario representar mas de dos sumandos se puede recurrir al símbolo algebraico de sumatoria (Σ). El símbolo λ, es la operación de transición unaria, como se ve en el ejemplo con la acción α y se representa como un árbol con la raíz y solo una hoja.

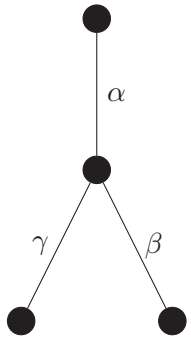

Figura 2.2.: Ejemplo del proceso $\alpha(\beta NIL + \gamma NIL)$

2.1.2. Conceptualización de la concurrencia

Existen dos conceptos fundamentales que definen la representación de procesos concurrentes, por un lado, las operaciones entre los elementos que componen al sistema y por otro la forma en que es posible "ver" el comportamiento del mismo, por ejemplo a través de su traza, sin embargo, aún hace falta entender como se puede representar este comportamiento para procesos ejecutándose simultáneamente.

En el enfoque de Hoare, pueden existir procesos de diferente naturaleza (como un hombre y una máquina) realizando tareas de forma simultánea pero pensando en que tarde o temprano habrá una interacción entre ambos. Para representar dos procesos que pueden interactuar, se hace utilizando el símbolo "||", y la única restricción asociada a este operador es que ambos procesos deben tener el mismo alfabeto, aunque de no ser así se puede realizar la unión los alfabetos de cada proceso y ese será el alfabeto del sistema, o bien se pueden empatar los alfabetos por medio de cambios de símbolos, con el propósito de tener una representación general de un sistema con procesos concurrentes y la definición de sus eventos asociados.

Milner analiza el comportamiento de un sistema a partir de experimentos que son una forma diferente de ver los agentes, lo cual resulta muy útil cuando se pretende determinar si dos pueden realizar las mismas acciones. Los procesos concurrentes se construyen al tomar dos aceptores o agentes (S y T) y se conectan para que puedan interactuar. Conceptualmente significa que un autómata puede "ver" las acciones del otro y viceversa a través transiciones etiquetadas como se ve en la figura 2.3.

2.1.3. Comunicación entre procesos

Posterior a la representación, es importante especificar la forma en que pueden establecer una comunicación. Hoare define dos elementos fundamentales, el canal de comunicación (c) y el mensaje a transmitir (v), cuya especificación general es $c.v$, pero debido a que el primero es unidireccional, un evento completo de comunicación se define mediante la especificación de dos, uno correspondiente a la entrada del mensaje por un canal ($c?v$) y otro a la salida del mismo mensaje por el mismo canal($c!v$). En este caso, el mecanismo de comunicación se establece por medio una variable

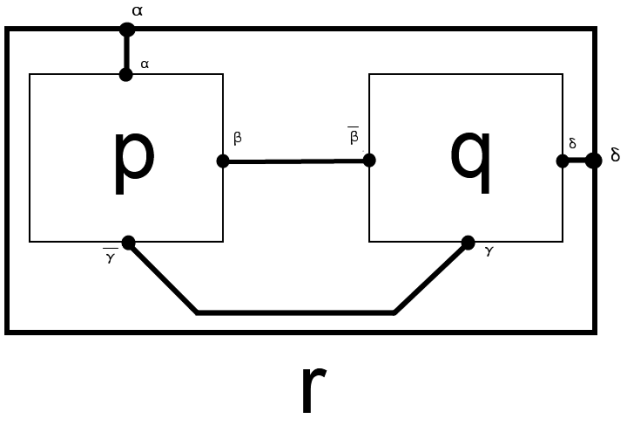

Figura 2.3.: Ejemplo de $r = p||q$

compartida que servirá como medio de intercambio de datos.

Este modelo de comunicación debe cumplir que cualquier especificación del estilo $c.v$ debe encontrarse en el alfabeto de los procesos que requieran comunicarse y que los eventos de envío y recepción se encuentren disponibles para que el sistema no caiga en **deadlock**, es decir, ya se sabe que el sistema puede llegar a un estado del cual no pueda salir si el proceso de comunicación falla.

Milner define su mecanismo de comunicación con reglas similares pero a través de una conexión (estableciendo un canal de transmisión) entre los agentes y definiendo el mismo evento de comunicación en ambos elementos pero marcando al receptor con un guión sobre el nombre del canal, por ejemplo, la comunicación de un dato v a través del canal c se representa por medio de los elementos $c.v$ y $\overline{c}.v$ o como se puede observar en la figura 2.3 $\gamma.v$ y $\overline{\gamma}.v$.

2.1.4. Representación de procesos infinitos

Existen problemas cuya solución concurrente será un sistema que se ejecute de forma indefinida, por lo tanto es necesario establecer un medio para representar procesos que son pensados para que su ejecución sea infinita.

Hoare plantea una definición recursiva a través de ecuaciones de tipo $P = (x \rightarrow P)$, donde P es una variable recursiva y dicha ecuación representa un sistema donde se ejecutará el evento x y

regresará al estado P.

Este tipo de ecuaciones funcionará adecuadamente si su lado derecho tiene al menos un evento prefijo para todas las ocurrencias recursivas del nombre del proceso, en cuyo caso se dice que es ***custodiada***. Este concepto fue planteado previamente por Dijkstra[Dij97] y constituye una de las aportaciones mas importantes para sistemas concurrentes con comportamiento infinito ya que gracias a esta característica es posible definir estados finitos de un sistema y los medios para llegar a ellos.

Un proceso se define a partir de varias ecuaciones recursivas tomando solo en cuenta que el lado derecho de todas las ecuaciones debe ser custodiado y que cada elemento desconocido debe aparecer al menos una vez del lado izquierdo de las ecuaciones.

Este resultado es significativo ya que permite tener certeza acerca del siguiente estado en el que se encontrará el sistema, aunque también existen casos donde se tiene mas de un evento prefijo que puede ser ejecutado en un momento dado, cambiando el comportamiento del sistema dependiendo del evento ejecutado. Este comportamiento se puede escribir de la siguiente forma: $(x \rightarrow P|y \rightarrow Q)$, donde se asume que se tiene un programa que puede recibir los eventos x ó y. Si $x \neq y$ la elección entre x ó y es determinada por el evento que suceda primero, es decir, se brinda una caracterización para el no determinismo.

A continuación se presenta un ejemplo donde se define un proceso con varias ecuaciones recursivas y se utiliza el operador "|".

$$\begin{aligned} P &= (x \rightarrow Q|y \rightarrow R) \\ Q &= (w \rightarrow Q|y \rightarrow R|x \rightarrow Q) \\ R &= (z \rightarrow R|x \rightarrow Q|y \rightarrow R) \end{aligned}$$

Si se utilizan valores indexados en los procesos se puede especificar un conjunto infinito de ecuaciones:

$$\begin{aligned} P_0 &= (x \rightarrow P_1|y \rightarrow P_0) \\ P_{n+1} &= (x \rightarrow P_{n+2}|z \rightarrow P_n) \end{aligned}$$

En CSS también es posible extender la representación algebraica de forma recursiva para definir comportamiento infinito como en el caso de CSP. Las ecuaciones recursivas 2.1 representan el RTS de la figura 2.4. Este sistema tiene solución única ya que este árbol puede desarrollarse de arriba hacia abajo y a cualquier nivel de profundidad observando un patrón que se repite, sin embargo se debe ser cuidadoso, ya que existen algunos casos donde tipo de ecuaciones puede tener mas de una solución, sin embargo, en este enfoque aún no existen restricciones definidas bajo las cuales este fenómeno no suceda.

$$\begin{aligned} s_0 &= \alpha s_1 \\ s_1 &= \beta s_2 + \gamma s_3 \\ s_3 &= NIL \\ s_4 &= \delta s_0 \end{aligned} \tag{2.1}$$

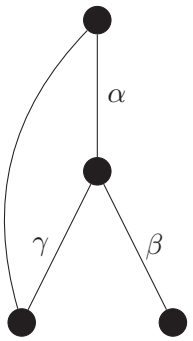

Figura 2.4.: Ejemplo de un proceso infinito

2.1.5. Comportamiento interno de los sistemas concurrentes

A través de las custodias es posible representar la ejecución no determinista de un sistema compuesto de procesos concurrentes, lo cual es útil para mantener un alto nivel de abstracción en la descripción del comportamiento del sistema pero también dificulta su implementación desde su planteamiento conceptual hasta el modelado de su comportamiento ya que funciona de manera muy distinta a un proceso secuencial, cuyo comportamiento siempre es determinista.

Cuando se define el alfabeto para los procesos que conforman un sistema con procesos concurren-

tes en CSP, se elige la menor cantidad de elementos que reflejen el comportamiento de un sistema, sin embargo, algunas veces es útil agregar algunos eventos que reflejan comportamientos internos que se desean controlar pero que no son relevantes cuando se construye la traza. El concepto de ocultación fue desarrollado para realizar la especificación de dichos comportamientos internos sin dejar rastros de estas acciones en la traza del sistema.

En CSS es mucho mas fácil apreciar este tipo de acciones internas al conectar dos aceptores, por ejemplo, p y q formando uno nuevo llamado r, el cual tiene las acciones de ambos, excepto las que se conectaron, quedando ocultas. Es decir, se plantea un modelo similar al de Hoare pero ahora los elementos que integran el sistema son del mismo tipo e inclusive se definen también acciones internas que se representan dentro del modelo con la letra τ.

2.2. Verificación de sistemas concurrentes

Una vez que se define un sistema concurrente es útil saber si la definición planteada cumple con los requisitos establecidos al inicio o si al menos se puede garantizar la concurrencia, es decir, no caer en un estado de deadlock.

La notación algebraica utilizada por Hoare es capaz de describir varios elementos diferentes, es decir está dotada de la suficiente expresividad para representar diferentes elementos conceptuales:

- Procesos concurrentes.
- Algunas leyes generales que ayudan a establecer relaciones entre procesos.
- Especificaciones de características deseables especificación de forma algebraica.

En CSP es posible asociar un proceso P con una ecuación que represente alguna priopiedad deseable o no deseable del sistema, llamada especificación S y cuando P cumple con ella se dice que P *satisface* S. Esto implica todas las trazas generadas por P cumplen las especificaciones de S, es decir, cualquier ejecución del programa será correcta.

La forma que plantea Hoare para corroborar esta relación es a través *demostraciones* con respecto a las trazas, para lo cual define operadores utilizados en lógica de predicados (como and, or y $\Rightarrow$),

sin embargo, este enfoque no cuenta con una herramienta específica para evitar que un sistema modelado no caiga en deadlock y detectarlo es un proceso que consiste en analizar un número representativo de casos descritos en términos del problema o realizar un programa que analice todos lo casos cuando esto sea posible.

Una característica peculiar en este enfoque es el manejo del tiempo ya que ignora la duración exacta de una acción, ya que su definición solo coloca su duración en términos del tiempo que transcurre entre eventos, lo cual simplifica el diseño de soluciones y aunque en un sistema con procesos concurrentes sus acciones pueden traslaparse en el tiempo si el inicio de una precede el final de la otra, para el autor el desempeño debe ser tratado como un problema de implementación y no de diseño.

En el caso de CSS cuando se conectan dos o mas aceptores puede ser que el sistema formado pueda pasar a un estado en que ya no sea válido ningún λ-experimento, es decir, el sistema se encuentra en deadlock, sin embargo, en este enfoque si es posible establecer restricciones sobre ciertas transiciones que pueden llevar al aceptor este estado, con la notación $r\backslash\lambda$, donde r es un proceso definido (un aceptor) y λ es la acción que se quiere evitar, es decir, se define una encapsulación o aislamiento de operaciones dentro de un conjunto.

Para dar formalidad a este enfoque, se determina su sintaxis y se establece una semántica de tipo operacional para la especificación de los procesos concurrentes. Entre los elementos sintácticos definidos se encuentran variables, símbolos constantes, las acciones y expresiones de comportamiento del sistema (funciones), así como el comportamiento nulo (NIL), composiciones $(A|B)$, restricciones $(A\backslash\lambda)$ y re-etiquetamientos de símbolos.

La semántica se define a partir de reglas de inferencia utilizando como base acciones atómicas $p \xrightarrow{\lambda v} p'$ y su uso permite identificar cuando dos aceptores son equivalentes. Esto se logra por medio de una relación de equivalencia[Gri04] llamada *equivalencia de observación* definida a través de un conjunto de transformaciones sucesivas, denotadas como s, sobre dos agentes, por ejemplo p y q, esperando que cada resultado p' de aplicar cada uno de los experimentos s sobre p debe ser equivalente a cada resultado q' al aplicar el mismo experimento sobre q y viceversa,es decir, $p \approx_k q$ si $p' \approx_{k-1} q'$, donde k representa el k-ésimo λ-experimento. Si además se verifican las salidas de cada

paso del sistema, se puede dar otra equivalencia aún mas fuerte llamada congruencia completa (o strong congruence), denotada por $p \equiv q$. Esto implica que necesariamente $p \overset{S}{\Rightarrow} q$, $p \sim q$ y finalmente que las salidas de ambos programas en cada acción de S sean idénticas.

En los enfoques mencionados, la forma de verificar un sistema es a través de medios algebraicos, enriquecidos con elementos lógicos que permiten garantizar que el comportamiento del sistema es el deseado.

Mas aún, Milner define una relación que permite saber cuando dos soluciones concurrentes son equivalentes, es decir, existe una relación de igualdad entre soluciones, lo cual dará lugar al concepto de axiomatización que será utilizado como una de las herramientas principales para la definición del álgebra de procesos utilizada en el presente trabajo.

2.3. Conceptualización del álgebra de procesos

A través de los conceptos anteriormente presentados es posible definir una representación de procesos concurrentes que permita analizar la forma en que los sistemas actúan cuando son ejecutados y obtener información que permita entenderlos. Es necesario que dicha representación pueda hacer visible todas las interacciones entre las partes y la forma en que los elementos ejercen su influencia, es decir, un modelo que pueda reflejar el comportamiento de un sistema.

Sin embargo, para que dicho modelo sea reproducible en un ambiente determinado es necesario despreciar ciertos aspectos y solo tomar en cuenta aquellos que aporten información vital para el entendimiento del sistema modelado.

Desde un enfoque matemático la forma mas sencilla de ver un proceso es como una función de entrada-salida donde el procesamiento se realiza internamente, arrojando un resultado al final de su ejecución. Sin embargo dicha representación no permite ver ni modelar posibles interacciones entre procesos, las cuales son necesarias para describir sistemas paralelos o distribuidos.

Por otro lado, los modelos de tipo operacional son ampliamente utilizados por los diferentes enfoques que buscan analizar y modelar el comportamiento concurrente, ya que permiten representar el comportamiento de un sistema como un conjunto de estados y transiciones motivadas por accio-

nes definidas, las cuales forman una traza que permite visualizar la forma en que se realizaron las operaciones internas. Debido a sus ventajas, el álgebra de procesos utilizada en el presente trabajo tendrá este tipo representación matemática.

La palabra álgebra denota la adopción de un enfoque algebraico / axiomático para hablar de comportamiento, es decir, se utilizan técnicas del álgebra universal, por tanto, es una estructura matemática que satisface los axiomas dados por operadores básicos.

Los elementos principales del álgebra utilizada son los procesos y su manipulación se realiza gracias al establecimiento de una axiomatización. En [Bae05] se define al álgebra de procesos como *"El estudio del comportamiento de sistemas paralelos o distribuidos por medios algebraicos"*.

Sin embargo, para lograr una representación sencilla pero expresiva de los sistemas concurrentes, en el álgebra de procesos que se especificará a continuación se han omitido detalles como valores de variables auxiliares, tipos de datos o tiempos de ejecución de las acciones definidas pero se ha logrado definir un modelo de comunicación entre procesos concurrentes.

En el apéndice A se detalla la definición formal del álgebra de procesos utilizada en el presente trabajo, tomando en cuenta las diferentes extensiones realizadas. La axiomatización generada por las diferentes extensiones se encuentra en el apéndice B. Este conjunto de axiomas será aplicado en el capítulo 5 en la solución de un problema específico.

2.4. Resumen

En el presente capítulo se conceptualizó al álgebra de procesos como un modelo operacional capaz de representar el comportamiento de un sistema concurrente. También se ha realizado un recuento de algunas de las ideas mas importantes que han dado origen al álgebra de procesos y la visión de sus autores, quienes han sido un referente para casi toda la investigación concerniente al análisis y verificación de procesos concurrentes.

Aunque las ideas Hoare y Milner fueron motivadas por distintas razones y modeladas de forma muy diferente es posible ver en la tabla 2.1 la semejanza de sus conceptos, lo cual resulta lógico debido a que ambos modelaron sistemas, vistos desde el punto de vista mas general del concep-

to, donde las partes podían realizar actividades simultáneas interactuando con su entorno y se comunicaban entre ellas para lograr realizar una tarea especifica.

Elemento	**CSP**	**CSS**
Acción y transición	$a \rightarrow B$	$a \xrightarrow{\lambda} b$
Procesos concurrentes	$A \| \| B$	$A \| \| B$
Fin de ejecución	$\surd$	
Deadlock	Identificado con $STOP$	Evitado con $S \backslash \lambda$
Comportamiento del sistema	Trazas: $< \cdots >$	Ordenamiento: Λ
Procesos infinitos	$P = x \rightarrow P$	$s = \lambda s$
Procesamiento secuencial y alternativo	ab y $a \| b$	$\alpha\beta$ y $\alpha + \beta$
Representación	Figura 2.1	Figura 2.2
Comunicación	$c!v$ y $c?v$	$c.v$ y $\overline{c}.v$
Acciones ocultas	Cambios de símbolos	Operación τ
Verificación	Leyes, especificaciones y demostraciones	Semántica y derivaciones
Equivalencia		*Strong Congruence*

Tabla 2.1.: Resumen de los enfoques de Hoare y Milner

Es interesante notar que los conceptos de CSP y CSS son aplicables en arquitecturas multicore aunque su génesis no fuera en este tipo de entornos, lo cual denota cierta universalidad que permite pensar que al resolver un problema de la vida real muchos de los fenómenos obedecen a este tipo de comportamiento, por lo que su representación algebraica efectivamente es un buen modelo de representación, aunque estos enfoques no contemplan la generación de un algoritmo, sino el modelado de sistemas y su paso inmediato a una implementación.

Utilizando como base los conceptos anteriores, Wan Fokkink [Fok99] planteó el álgebra de procesos que será empleada a lo largo del presente trabajo, la cual proporciona, por un lado, una representación gráfica de procesos concurrentes y por otro, una representación algebraica que incluye al menos operadores básicos para construir procesos finitos, operadores de comunicación para expresar concurrencia, posibilidad de capturar comportamiento infinito, una representación para deadlock, un Silent step para extraer información de cálculos internos, una relación de los términos

procesos con sus gráficos de procesos así como elementos lógicos para establecer una relación de igualdad entre términos.

La definición de las características anteriores se logró a partir de la definición sintáctica y semántica de un conjunto mínimo de elementos algebraicos y posteriormente se realizaron extensiones del álgebra resultante para agregar otros operadores con el propósito de incrementar su expresividad hasta lograr modelar sistemas concurrentes.

Cada vez que se define un nuevo elemento es importante verificar que la semántica resultante sea completa y correcta, utilizando la lógica como herramienta básica para este fin, así como especificar los axiomas necesarios para su manipulación a través de relaciones de igualdad para evitar el uso de secuentes lógicos en el proceso de derivación de términos.

3. Desarrollo de la programación de procesos concurrentes

El propósito de esta sección es brindar un panorama actual de los esfuerzos que se han realizado para la generación y verificación de soluciones concurrentes.

Aunque existen esfuerzos individuales que abordan problemas de concurrencia o paralelismo particulares, de forma general se han desarrollado tres vertientes principales en el terreno de la generación de sistemas concurrentes y su verificación.

La primera tiene como base las precondiciones débiles (weakest preconditions) de Dijkstra [Dij97], así como el enfoque de los *Sistemas de acción (Action Systems)* [BKS89] el cual está basado en CSP [Hoa78] pero con un elemento adicional llamado *Refinamiento* [DS08] el cual ayuda a llegar a una solución a través de pasos sucesivos que convergen hacia una solución y la conservan correcta.

La segunda está basada en lógica temporal de proyección (PTL por sus siglas en inglés) que dio origen al Lenguaje de modelado, verificación y simulación (MSVL)[DYK05].

Por último, el *Álgebra de procesos en Tiempo Real (RTPA)* se desarrolla como un conjunto de notaciones matemáticas para describir la arquitectura y el comportamiento de sistemas, este último tomado en cuenta desde una perspectiva tridimensional [Wan02].

A continuación se realiza una descripción general, sin pretender ser exhaustiva, de las vertientes mencionadas.

3.1. Enfoques basados en Sistemas de Acción

Refinement Calculus

Fredrik Degerlund y Kaisa Sere describen un enfoque llamado *Refinement Calculus* [DS08] para diseñar sistemas concurrentes a través de un método de derivación paso a paso, empezando desde la descripción de alto nivel de una solución concurrente hasta su implementación, asegurando que el sistema obtenido es correcto. Su enfoque está soportado por la técnica de precondiciones débiles de Dijkstra[Dij97] y por los *sistemas de acción*[But96], cuyo fundamento es visualizar un sistema como un conjunto de estados que modelan computación paralela, pero su interpretación es secuencial.

A continuación se definirá el concepto de *refinamiento* y se mencionará brevemente un método para llevarlo a cabo, así como su sintaxis, la cual está basada en los comandos custodiados de Dijkstra con algunas adiciones. En este enfoque existen dos categorías sintácticas: sentencias y acciones: una sentencia tiene como uno de sus componentes un conjunto de acciones, es decir, las sentencias S están definidas como:

$$S ::= x := e | Q | S_1; \cdots; S_n | if A_1 [] \cdots A_m fi | do A_1 [] \cdots A_m od | | [var\ x; S] | \tag{3.1}$$

Donde las A_i son acciones, las x son variables, e denota expresiones y Q es un predicado.

Precondiciones débiles (wp)

Una *precondición débil* es un predicado que describe todos los estados en los cuales comienza la ejecución de una sentencia S y se garantiza que su ejecución terminará en un estado que satisface a un predicado R, denotado como,

$$wp(S, R)$$

Por ejemplo, para la sentencia $x := e$ la precondición débil sería $wp(x := e, R)$, la cual se puede representar como $R[e/x]$, que se puede interpretar como la ejecución de las expresiones e que afectan a las variables x **mientras** se cumplan las condiciones establecidas en R.

Otro nombre para las pre-condiciones débiles (y el mas utilizado) es el de **custodias**, ya que siempre se están validando para permitir la ejecución de una acción cuando su valor es verdadero.

Refinamiento

Se dice que una sentencia S está *(correctamente) definida* por la sentencia S', denotado como $S \sqsubseteq S'$, si para cada precondición Q $wp(S, Q) \Rightarrow wp(S', Q)$. Esto significa que S' satisface al menos las mismas condiciones que S y por ello se dice que una sentencia S está refinada por la sentencia S'. El refinamiento de las relaciones entre las acciones y las sentencias es definido con respecto a la semántica de las precondiciones débiles.

El refinamiento está orientado a una manipulación paso por paso, debido a que formalmente es una relación reflexiva y transitiva, de tal forma que cada paso de refinamiento se acerca mas a la solución ya que su propósito es extender el dominio de terminación de una sentencia, decrecer su determinismo o ambos. Un refinamiento inicia estableciendo condiciones iniciales sobre las funcionalidades mas básicas, en los pasos posteriores se van agregando funcionalidades al sistema, verificando que cumpla los requerimientos establecidos y termina cuando se logran satisfacer.

Sistemas de acción (Action systems)

Un sistema de acción[But96] es una sentencia de la forma

$$A = |[var\, x; S_0; doA_1[\,]\ldots[\,]A_m od]| : z$$

Donde $y = x \cup z$ es el conjunto de las variables de estado, la cuales están asociadas a un dominio y z es de tipo global. El conjunto de posibles asignaciones a variables se le llama el espacio de estados y la ejecución sucede en paralelo debido a la especificación de la operación $do \ldots od$ [Dij97].

El comportamiento del sistema de acciones se da como sigue: se ejecuta primero la sentencia S_0 que básicamente asigna valores iniciales a las variables de estados; posteriormente, conforme se van habilitando acciones, se elige una de forma no determinista y se ejecuta en el tiempo t. Las acciones son atómicas y pueden ejecutarse en paralelo si no tienen variables en común, por lo que puede considerarse como un programa en paralelo cuando está asociado a un conjunto de procesos, ya sea a través de la asignación de acciones a procesos o de la asignación de variables a procesos, aunque solo se tomará en cuenta el primer mapeo (acciones a procesos) para las soluciones a los problemas

planteados.

La asignación mencionada da lugar a particiones con respecto al conjunto de acciones del sistema, las cuales no necesariamente son disjuntas pero su unión siempre da como resultado el conjunto total de acciones, lo cual quiere decir que puede haber una acción asignada a mas de un proceso y que puede ser ejecutada en cualquiera de ellos, pero la decisión de cual proceso la ejecutará se deja como un problema de implementación.

La detección de los conflictos entre variables, por ejemplo cuando una variable se menciona en dos o mas acciones que se pretenden ejecutar simultáneamente, también está clasificado como un problema de implementación.

Un sistema de acción puede ser desarrollado pensando en una plataforma de ejecución o en un conjunto de procesos, aunque se ha puesto principal énfasis a plataformas tipo retícula, donde en principio cada acción puede ser ejecutada en una unidad de proceso, sin embargo las reglas consideradas en este enfoque pueden aplicar a una clase mas amplia de plataformas basadas en procesos concurrentes.

Debido a que los sistemas de acción son un tipo de sentencia secuencial, se puede utilizar Refinement Calculus para realizar refinamientos paso a paso. Para transformar un algoritmo secuencial en un sistema de acciones que pueda ser ejecutado de forma paralela se definieron reglas de transformación para introducir tres elementos indispensables: convertir sentencias arbitrarias en construcciones **do - od**; unir sistemas de acción secuenciales y cambiar variables en las sentencias [BS91].

Event-B

La evolución del Refinement Calculus fue llamada **Event-B** [BDSW14] y su propósito es ser un marco de trabajo formal para desarrollar programas concurrentes y distribuidos utilizando dos conceptos nuevos: la descomposición de programas en sub-modelos y la creación de planificadores de eventos.

Event-B es un lenguaje de modelado basado en estados, los cuales están compuestos por los valores del conjunto de variables definidas para un programa en particular. Consta de dos partes

principales: *La máquina*, que es la parte dinámica del modelo, donde se definen los eventos que cambian el estado del sistema y el *contexto*, donde se define la parte estática del sistema, como constantes o enumeraciones previamente definidas.

Los eventos pueden escribirse de la siguiente forma,

$$e_k \hat{=} \textbf{when}\, G_k(v, c)\, \textbf{then}\, v : |A_k(v, v', c)\, \textbf{end}.$$

Donde v es el conjunto de variables, c el conjunto de constantes, A_k representa una acción de asignación y G_k representa el conjunto de custodias que habilitan las acciones de asignación.

Al dispararse la ejecución una acción, G_k la evalúa y si las custodias lo permiten se realiza una asignación del conjunto A_k, elegida de forma no determinista, que cambia el estado del sistema de v a v'.

Específicamente, el contexto está compuesto por las contantes c, los conjuntos enumerados (que son otras constantes) y una serie de axiomas[1] que establecen propiedades de las constantes y los conjuntos enumerados.

La máquina se compone de las variables, los eventos, la definición de invariantes - cuya función principal es delimitar los valores de las variables - y la definición de variantes, que son expresiones que hacen referencia a conjuntos bien formados, las cuales sirven para saber si un nuevo paso de refinamiento converge hacia la solución. También se debe definir una acción especial llamada inicialización, la cual no tiene custodias y es la primera en ser ejecutada.

Como se ya ha mencionado, el refinamiento tiene dos funciones básicas, la primera es extender la funcionalidad del sistema y la segunda disminuir el no determinismo en la elección de los eventos $v : |A_k(v, v', c)$ a ejecutar. Por lo tanto, cuando se piensa en atacar un problema con una solución concurrente construida con Event-B:

1. Se establece la estructura contexto-máquina, antes mencionada
2. Se define la funcionalidad mas sencilla del sistema en un entorno abstracto
3. Se realizan los pasos de refinamiento necesarios,los cuales incrementan la complejidad del

[1]El concepto de axioma es el mismo utilizado en la lógica, es decir, un secuente sin premisas

sistema desde el punto de vista funcional pero lo hacen cada vez menos abstracto ya sea aumentando el número de variables, introduciendo nuevos eventos o realizando cambios en otros elementos, verificando siempre las invariantes y variantes establecidas para asegurar que en cada paso se obtendrá una representación correcta.

Descomposición

Una vez terminado el refinamiento del sistema se utiliza el enfoque divide y vencerás para descomponer la especificación del sistema en partes pequeñas que puedan ser desarrolladas de forma independiente, aunque posteriormente se debe realizar un proceso de composición y una verificación por refinamiento de todo el sistema.

Uno de los puntos mas importantes en esta técnica es el manejo de las variables, ya que cada vez que se divide el modelo en sub-modelos, cada uno puede ver solo sus propias variables, es decir, son internas para cada uno, sin embargo, esto no es siempre lo que se necesita ya que al dividir el sistema puede suceder que una variable sea necesaria en mas de una de las partes, por lo que también es posible definir variables externas, las cuales son visibles para todos los sub-modelos y sirven como su medio de comunicación. También existen acciones llamadas externas que solo modifican variables externas.

Una consecuencia directa de esta organización al realizar divisiones en el modelo es que los conjuntos de variables internas de cada sub-modelo sean disjuntos, por lo que los eventos asociados pueden ejecutarse de forma concurrente debido a que no comparten variables, sin embargo, el orden en que los eventos se ejecutan y su interacción no ha sido definida aún, por lo que ha sido un objeto de estudio el logro del paralelismo con la técnica de descomposición da sistemas definida en Event-B.

Planificadores

Una vez realizado el refinamiento para definir la representación de la solución y haber repartido el dominio de datos entre los sub-modelos, es importante cuidar la forma en que se ejecutan, ya que por definición lo hacen de forma no determinista. Una forma de cambiar esto es asociarlos con un planificador que los ejecutará en un cierto orden establecido.

Si se define un planificador dentro de un sub-modelo, cualquier evento que se encuentre dentro del planificador se ejecutará y cualquiera que no se encuentre será como si no se hubiera definido. Un sub-modelo cuyos eventos han sido planificados se le llama tarea y debido a que existen un número finito de sub-modelos, también el número de tareas es finito. Visto desde una arquitectura multicore, los planificadores pueden ser vistos como hilos de ejecución con la diferencia de que estos se definen desde el diseño y no hay ninguna forma de crear alguno durante la ejecución del programa.

El resultado final en este modelo es un conjunto de tareas finito con una ejecución concurrente, donde los elementos de dichas tareas son eventos que tienen ciertas restricciones para evitar caer en deadlock. Una forma sencilla de demostrar esto es tomando al conjunto de custodias definidas en el modelo y operarlas como si fueran una disyunción (con el operador $\vee$) y para cualquier evento, el resultado siempre es verdadero.

Una idea fundamental en los enfoques basados en Sistemas de Acción es el uso de las precondiciones débiles para la construcción de sus sentencias, ya que su uso permite saber que tarea se ejecutará al conocer la primera acción, pero mas importante, permite establecer restricciones sobre la ejecución de las tareas.

El refinamiento también es muy importante en este enfoque pero la idea general detrás de él - la técnica divide y vencerás - lo hace una herramienta muy poderosa cuando se requiere resolver problemas complicados o cuando se pretende dividir la solución para ser ejecutada por varios procesos. Se puede utilizar un refinamiento cuando se desea agregar una funcionalidad a un sistema que ayudará a cumplir con algún nuevo requerimiento pero deseamos que el sistema siga cubriendo los requerimientos previamente satisfechos.

En este enfoque las comunicaciones no se encuentran consideradas dentro de su alcance ya que se asume que en la implementación de los sistemas se tendrán los mecanismos necesarios para la interacción de los procesos, es decir, no existe una definición concreta ni restricciones sobre su uso que puedan servir como base para su implementación.

3.2. Enfoques basados en Lógica Temporal

Usualmente los enfoques basados en lógica temporal realizan extensiones para aumentar su expresividad pero siempre cuidando que sean completas y correctas, tal como sucede con BPA. Por ejemplo, la lógica temporal se extendió hacia la *Lógica Temporal por Intervalos (ITL)*, que formaliza el concepto de intervalo como una secuencia de estados definida y las reglas semánticas se aplican sobre un modelo y un intervalo, es decir, la verificación de que una fórmula es verdadera ya no depende solo del modelo definido sino también en que estados dentro del intervalo se está verificando.

En este enfoque, un *estado* s se define como un par (l_v, l_p) donde v es un conjunto de variables, es decir, $v \in V$ y p es un conjunto de proposiciones bien formadas de lógica temporal a las cuales se les asigna el valor de **verdadero** en el estado s.

Un intervalo estaba definido como una secuencia finita de estados etiquetados con números enteros, por ejemplo $\sigma =< s_0, s_1, \cdots, s_{|\sigma|} >$, donde $|\sigma| \in \mathbb{N} \cup \{0\}$ (enteros no negativos). Los intervalos son por definición ordenados debido a que reflejan secuencias de tiempo y su longitud se define como $|\sigma|$ = # de estados -1.

Una de las restricciones teóricas mas importantes de este tipo de lógica es el hecho de que todo estado s tiene un sucesor s'. Si se toma en cuenta que cada estado está asociado con un conjunto de variables y una serie de proposiciones que son verdaderas en él, se puede pensar que s es una "instantánea" de un proceso ejecutándose en la computadora y $s_0, s_1, \ldots, s_n$ es un intervalo de estados que suceden ante la ejecución de un programa.

Con esta analogía en mente, un estado que no lleva a otro quiere decir que se mantendrá en él, es decir, el sistema caerá en deadlock. Para modelar y controlar este comportamiento no deseado, se coloca un estado extra s_d con un lazo y cualquier otro estado que no tenga sucesor se conecta a s_d.

En [ZDT13] se realiza una extensión de ITL llamada *Lógica temporal de proyecciones (PTL)*, que además agrega un operador llamado proyección (*prj*) que servirá para modelar concurrencia y comportamientos infinitos. Para lograrlo, se define el elemento infinito ω y se extienden las

definiciones asociadas a los intervalos para incluirlo y lograr una extensión correcta y completa de ITL:

- Los intervalos tendrán como dominio al conjunto $\mathbb{N} \cup \{0, \omega\}$
- Es posible definir intervalos cuyo último elemento es σ_ω y para mantener su característica de ser ordenados se define que $\omega > x$, donde $x \in \mathbb{N} \cup \{0\}$.
- Los intervalos que contienen a ω tienen longitud infinita y la operación $s_{|\omega|}$ es indefinida.

Otro concepto importante es la *x-equivalencia* entre dos intervalos, denotada por $\sigma \stackrel{x}{=} \sigma'$. Dicha relación está definida para dos intervalos que tienen la misma longitud ($|\sigma| = |\sigma'|$), los valores de las variables son iguales en cada estado y los valores de las proposiciones también lo son.

Para evaluar los términos o fórmulas sobre un subintervalo en algún estado k se utilizan elementos lógicos llamados interpretaciones, los cuales establecen una relación de satisfacción con respecto a los términos que se desea evaluar.

La representación de la concurrencia se realiza a través de dos conceptos importantes, la definición del operador $||$ y el concepto de proyección que básicamente toma diferentes secuencias, cada una asociada con fórmulas de PTL (conjunto *prop*) denotadas con ϕ_i, donde existen estados especificados que realizan comunicaciones con otros en tiempos bien especificados para su ejecución concurrente, como se puede ver en el ejemplo proporcionado por [ZDT13] donde es posible ver los conceptos mencionados. Sean los procesos

$$\phi_1 \text{ ov } (2,3,3,4)||\phi_2 \text{ ov } (3,5,3,6)||\phi_3 \text{ ov } (2,1,2,3,3,1,5) \tag{3.2}$$

Donde $\phi_1, \phi_2, \phi_3 \in prop$, cada uno es una fórmula en PTL y juntos forman un sistema donde se planea que su ejecución sea en paralelo, como se puede ver en la figura 3.1, donde el proceso ϕ_1 tiene 4 estados: el primero tiene una duración de 2 intervalos de tiempo, los dos siguientes de 3 y el último de 4, tal como está definido en la ecuación 3.2 y lo mismo sucede con los procesos ϕ_2 y ϕ_3.

Este marco de trabajo es llamado *Cylinder computation Model (CCM)* y su sintaxis se define a partir de los conceptos anteriores, es decir, en función de las proyecciones y las fórmulas de PTL, sin embargo su semántica se vuelve de tipo operacional debido a que las reducciones afectan

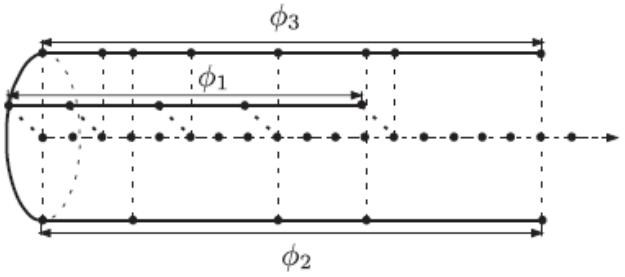

Figura 3.1.: Representación de una ejecución concurrente en PTL

directamente a los estados, cambiando su definición. Una derivación de tipo $\sigma_i \rightarrow \sigma_{i+1}$ cambia las definiciones de los estados involucrados y su representación formal es la siguiente,

$$(\bigcirc\phi, \sigma_{i-1}, s_i, i) \rightarrow (\phi, \sigma_i, s_{i+1}, i+1)$$

Esta representación se puede interpretar como una transición que sucede a partir del estado actual s_i, el cual se agrega al modelo σ_{i-1}. Cuando ϕ solicita ejecutarse en el estado s_i provoca que la definición de s_i cambie a s_{i+1}, incrementando el número de estados $(i+1)$ y por lo tanto σ_{i-1} cambia a otro modelo σ_i que incluye a s_{i+1}.

Aunque los sistemas realizados con este enfoque se definen a partir de axiomas y secuentes lógicos, en [ZDT14] se define una axiomatización para CCM que pretende explorar una metodología de verificación que combine las técnicas de *model checking* y *theorem proving* como en [Bon10].

Hasta este momento este enfoque solo contempla la representación de sistemas concurrentes por medio de una semántica definida, de la misma forma que en álgebra de procesos. También establece una relación que permite verificar términos y otra para saber cuando dos intervalos son son similares, sin embargo, la representación de un sistema es demasiado abstracta para ser implementada en un lenguaje de programación.

Para lograr una representación que pueda se mapeada a un lenguaje de programación se realiza una relación entre los elementos sintácticos de PTL y un lenguaje llamado Tempura, generando una extensión de este, cuyo nombre es Lenguaje de Modelado, Simulación y Verificación (MSVL), el cual se especifica también en [ZDT13] y contempla como parte de su definición su ejecución en arquitecturas multicore . Uno de los conceptos mas importantes para lograr esto es el de *framing*

que básicamente permite llevar el valor de una variable de un estado a otro por medio de predicados que permiten detectar las ocurrencias de las asignaciones a variables.

Gracias al concepto de framing es posible establecer comunicaciones entre intervalos ya que se definen variables globales que pueden ser detectadas y controladas por los predicados antes mencionados, es decir, existen espacios de memoria compartida por los que se sincronizan los elementos. Cualquier programa elaborado por CCM se implementa exclusivamente en MSVL debido a la relación PLT-Tempura, es decir, no hay forma de realizar un mapeo hacia algún lenguaje de programación diferente.

3.3. Álgebra de procesos en tiempo real

Este enfoque es una aplicación del álgebra de procesos y una definición operacional de sus elementos estructurales. Un aspecto interesante es la concepción de elementos arquitectónicos y funcionales integrados por medio de un proceso de refinamiento, de tal forma que la representación de una solución va cambiando hasta convertirse en un tipo de pseudocódigo. En [Wan02] se describe como un conjunto de notaciones matemáticas para describir sistemas con comportamiento estático y dinámico, donde un problema es concebido bajo tres tipos de acciones: operaciones matemáticas, manipulaciones de memoria y la sincronización de procesos.

De acuerdo el autor, los métodos formales actuales solo describen comportamientos estáticos en solo 2 de estas dimensiones (operaciones y memoria), sin embargo, esta notación no es suficiente para los sistemas en tiempo real , por lo que se ha creado una nueva capaz de describir y especificar comportamientos en las tres dimensiones llamada *Álgebra de procesos en tiempo real (RTPA)*.

Formalmente, el comportamiento de un sistema es resultado de las operaciones realizadas que afectan o cambian el estado del sistema, vistas como eventos de entrada - salida que afectan a variables o locaciones de memoria.

El *comportamiento estático* de un sistema es aquel que se puede determinar en tiempo de diseño y compilación, es decir, tiene que ver con las operaciones matemáticas y la memoria del sistema, mientras que el comportamiento dinámico solo se puede determinar durante la ejecución y tiene

que ver con los tiempos de sincronización de procesos.

Los componentes de software que serán descritos se clasifican en dos categorías, *componentes arquitectónicos* y *componentes operacionales*, mientras que la especificación de los sistema se puede divirir en tres subsistemas: *sistema de arquitectura*, *sistema de comportamientos estáticos* y *sistema de comportamientos dinámicos*.

Los procesos son las unidades básicas de comportamiento de un sistema de software y representan las transiciones de un sistema de un estado a otro. Un *proceso* en RTPA es una operación computacional que evoluciona a un sistema de un estado a otro cambiando sus entradas, salidas o variables internas.

En este enfoque se definen 16 meta-procesos[NW05] con su respectiva sintaxis y una semántica operacional, que corresponden a las operaciones mas comunes de los lenguajes de programación. Algunos de estos procesos reciben o arrojan información, por lo que se definieron también tipos de datos básicos y abstractos.

Un proceso puede ser un simple meta-proceso, que sirve como base para la construcción de un sistema, o un proceso complejo construido sobre reglas combinacionales con una dependencia hacia un meta-proceso. Para construir un sistema es necesario combinar los meta-procesos a través de la definición de relaciones estipuladas por reglas de semántica operacional para reflejar los comportamientos secuenciales y concurrentes, así como sentencias de control y operaciones dependientes del tiempo.

La especificación del sistema empieza con la definición del subsistema arquitectónico, ya que a través de este se elige la estructura de los procesos, por ejemplo, si se ejecutan en paralelo de forma secuencial.

Posteriormente se definen los subsistemas estático y dinámico, los cuales tienen diferentes niveles de detalle. Se componen de elementos de comportamiento estático y estos a su vez de elementos de comportamiento dinámico, que son los mas elementales, conformando así los diferentes niveles de refinamiento.

Uno de los aspectos mas importantes en este enfoque es la definición de los procesos, tipos y relaciones, los cuales son conjuntos finitos y por lo tanto es posible establecer una semántica

para cada componente, lo cual garantiza que un sistema definido con estos elementos funcione correctamente.

3.4. Resumen

Durante los últimos años el desarrollo de enfoques para el análisis de procesos concurrentes ha logrado realizar descripciones, verificación y han llegado a su implementación. Aunque todos los enfoques mencionados en el presente capítulo son de tipo operacional abordan de forma distinta el problema de paralelizar una solución, lo cual aporta distintas ideas que enriquecen la propuesta desarrollada en el presente trabajo.

Los trabajos actuales aportan ideas muy específicas y detalladas para lograr una representación confiable de sistemas concurrentes. Por ejemplo, los que están basado en Action System tienen como base los comandos custodiados propuestos por Dijkstra, los cuales obedecen a una sintaxis bien definida y su implementación es útil en los procesos de verificación.

Los enfoques basados en lógica temporal logran caracterizar el comportamiento de un sistema concurrente a través del concepto de estado sucesor; al igual que en CSS, definen una equivalencia entre el comportamiento de dos procesos por medio de la equivalencia entre intervalos y permiten representar al deadlock para su detección.

Otro aspecto interesante es la idea de realizar una axiomatización que ayude a facilitar el proceso de verificación.

A nivel conceptual RTPA establece elementos arquitectónicos y operacionales cuyo uso puede llevar a definir sistemas concurrentes en distintos niveles de detalle, sin embargo, el modelo empleado es sencillo pero intuitivo, por lo que puede resultar difícil definir los elementos arquitectónicos y si este paso falla, el error se propagará por los demás niveles de refinamiento.

Otra posible desventaja del proceso de refinamiento se puede manifestar cuando la definición lograda no cumple con las especificaciones requeridas, ya que puede ser complicado dar marcha atrás para encontrar el error o puede ser difícil detectarlo en pasos intermedios del proceso.

La técnica utilizada en este documento para abordar problemas con concurrencia consiste en

utilizar diferentes representaciones para una solución concurrente, las cuales se relacionan a través de un mapeo bien definido que hace posible pasar de una a otra, como se explicará en el siguiente capítulo.

4. El proceso de programación con procesos concurrentes

Actualmente la programación cuenta con múltiples herramientas, lenguajes y mecanismos que le permiten transformar una actividad artesanal en un proceso profesional que puede modelarse, medirse y mejorarse bajo esquemas de programación pensados para máquinas de Von Neumman.

Por otro lado la programación concurrente ha tenido un gran auge desde que los grandes fabricantes de microprocesadores introdujeron las arquitecturas multicore en el mercado, ya que se ha visto el potencial de este modelo de programación para explotar el poder de procesamiento de este tipo de equipos, sin embargo para lograrlo es necesario cambiar la forma en que se concibe y resuelve un problema, siendo al día de hoy una labor complicada.

Por lo tanto, es importante comenzar por plantear que la programación de procesos concurrentes es un problema por resolver ya que no existe la infraestructura metodológica con que cuenta la programación secuencial, sobre todo si se toma en cuenta que al realizar un programa que resuelve un problema dado, el hardware a utilizar se convierte en una restricción mas que debe ser contemplada, así como la selección de un lenguaje de programación, ya que debe contar con las instrucciones necesarias para llevar a cabo las operaciones especificadas en la descripción de la solución del problema.

Usualmente el planteamiento de una solución concurrente a un problema dado se realiza a través una representación donde se pueden hacer ciertas especificaciones generales o específicas, ya sean en forma gráfica, simbólica (matemática) o solo con texto, pero en cualquier caso dicha descripción se encuentra en un nivel de abstracción distinto al de un programa de computadora.

Intentar realizar un programa de computadora partiendo de una descripción de alto nivel no es un proceso sencillo ya que por un lado pueden existir diferentes criterios para codificar un mismo elemento. Por otro lado, no hay una garantía de que la programación refleje la solución modelada, a menos que exista una relación entre los elementos estructurales de ambas representaciones.

4.1. El proceso

Una de las ideas centrales en las que se basa el enfoque de este trabajo es el análisis que realiza Yang y otros en [YCW+14] acerca de los retos asociados al no determinismo en el desarrollo de programas paralelos (en particular multihilos) correctos. Cuando se ejecuta un programa paralelo puede haber varias formas en las que las tareas se ejecutan, es decir, una ejecución con las mismas entradas puede dejar mas de una una traza de las acciones que lleva a cabo debido al no determinismo inherente a la concurrencia, sin embargo, existen ejecuciones que llevarán al sistema a deadlock y otras a una ejecución correcta. En el trabajo mencionado, el problema de buscar solo ejecuciones correctas de programas concurrentes no deterministas es abordado en tiempo de ejecución, cuando ya se ha elaborado un programa y solo se limitan sus ejecuciones, por lo que no es posible corregir el código para mejorar algún aspecto del sistema.

En el presente trabajo, el análisis de la ejecución de programas concurrentes será realizado desde un tipo de representación donde sea posible eliminar las ejecuciones que lleven a deadlock antes de programar el sistema, garantizando la concurrencia de la solución y para lograr dicho análisis es necesario contar con un método que pueda servir como base para plantear soluciones concurrentes que puedan ser verificadas y posteriormente convertidas en código de algún lenguaje de programación.

Pancake y Bergmark [PB90] plantean un proceso para resolver problemas por medio del desarrollo de programación paralela (Figura 4.1), el cual empiezan con el planteamiento de una estrategia de alto nivel para atacar el problema y termina con la ejecución del programa. Este enfoque se centra principalmente en el tipo de problemas que resuelven los científicos y su necesidad manifiesta de procesamiento, la cual se pretende satisfacer utilizando paralelismo.

La elaboración de un modelo abstracto comienza con la delimitación del dominio del problema

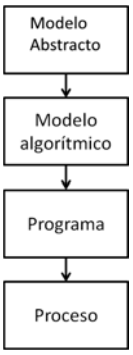

Figura 4.1.: Proceso para resolver problemas con programación concurrente

y seleccionando una estrategia para atacarlo. El resultado final se plantea en forma de diagramas o expresiones matemáticas que modelen la solución, para luego analizar que secciones de código pueden paralelizarse.

El siguiente paso es brindar una solución algorítmica de la solución, es decir, una representación que brinde una idea general acerca de la forma en que será dividido el sistema y como se plantea la solución del problema a través de algoritmos expresados en un lenguaje que pueda ser mapeado fácilmente a código en algún lenguaje de programación.

Una vez generada la representación algorítmica, se realiza la implementación, es decir, el programa que resuelve el problema y a través de las herramientas del lenguaje de programación elegido se realiza el último paso que es convertir el código en un ejecutable.

Un aspecto a tomar en cuenta durante el proceso es el entorno donde se ejecutará el programa ya que es una de las principales limitantes cuando se pretende tomar decisiones de diseño o rendimiento, en este documento el entorno será delimitado en el siguiente capítulo dentro del marco de trabajo.

Con el propósito de realizar un análisis mas detallado del proceso, este será representado como una función que recibe como entradas un problema y la definición de la arquitectura donde se ejecutará la solución final; la salida es dicha solución en forma de un ejecutable que resuelve el problema y cada etapa será una subrutina que "procesará" una representación de la solución y elaborará otra distinta que tendrá un propósito específico en el proceso, hasta llegar a una representación ejecutable en una arquitectura determinada (Figura 4.2).

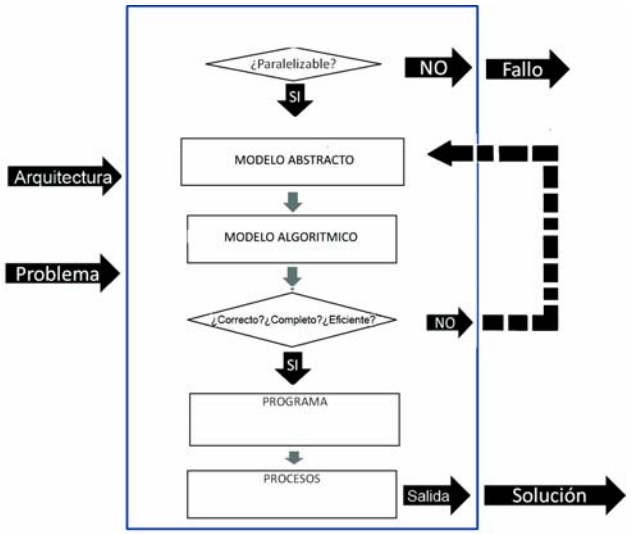

Figura 4.2.: Análisis del proceso

Una vez que se tienen las entradas (el problema y la arquitectura), el primer aspecto a considerar es si el problema puede ser paralelizado, o dicho de otra forma, ¿Es posible resolver el problema con procesos concurrentes?. Una vez contestada la pregunta afirmativamente se puede empezar a elaborar el modelo abstracto, en caso contrario, el proceso termina.

Realizar un modelo abstracto tiene un cierto nivel de complejidad pero conforme se avanza en el análisis de un problema, es mas sencillo desglosarlo y proponer una solución de alto nivel utilizando herramientas de Ingeniería de Software.

El siguiente reto que se presenta está asociado a la elaboración de algoritmos, ya que usualmente en este tipo de representación solo se proporciona un algoritmo secuencial con especificaciones generales de las secciones que pueden estar sujetas a paralelización pero no se incluyen detalles acerca de la interacción entre procesos o como será su carga de trabajo. El propósito del presente trabajo es brindar una descripción algorítmica para soluciones concurrentes.

Otro aspecto a considerar es la naturaleza de la solución propuesta, en particular existen dos aspectos fundamentales que cumplen los algoritmos secuenciales y que también deben tener los algoritmos concurrentes:

- La solución es completa. Se debe garantizar que un sistema compuesto de procesos concu-

rrentes siempre puede evolucionar hacia estados válidos.

- La solución es correcta. El sistema cumple con las especificaciones dadas por el problema.

Además de las características anteriores, una solución concurrente debe ser capaz de resolver el problema de forma mas eficiente que otra secuencial que resuelva el mismo problema, es decir, si ambas soluciones son ejecutadas en un entorno con mas de un procesador, la concurrente debe terminar en un menor tiempo.

La eficiencia es una de las principales razones para usar soluciones concurrentes y una vez que se pueda garantizar, el siguiente paso será convertir la solución en un programa y posteriormente ejecutarlo, sin embargo, cuando una solución no tiene las características mencionadas será necesario regresar a las etapas anteriores para ajustar la solución.

Cabe señalar que el proceso puede terminar en cualquier momento si no se encuentra una solución con las características descritas y quizá dependa mas del tiempo empleado en realizar o modificar la solución que en el número de veces que se modifique, ya que la intención es que desde la primera vez se obtenga el resultado deseado.

Para elaborar una representación en cada etapa del proceso es importante utilizar enfoques que puedan servir de guía para llegar a un sistema concurrente, como se puede ver en la figura 4.3.

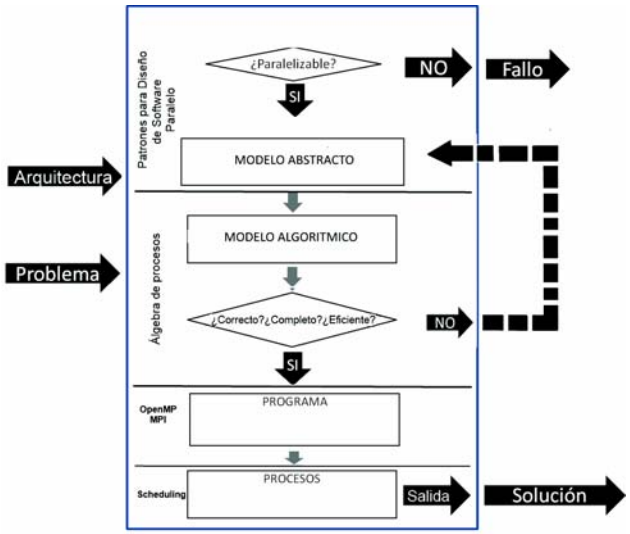

Figura 4.3.: Enfoques para seguir el proceso

En la parte de definición de modelos abstractos se eligió el trabajo de *Patrones para diseño de Software Paralelo* [OA10] debido a este enfoque no solo proporciona una representación gráfica de la solución, también ayuda a tomar decisiones acerca de la forma en que se dividen las funciones del sistema y el dominio de datos, como se analizará en el capítulo siguiente.

El enfoque presentado en este documento es utilizado para la definición del modelo algorítmico, fundamentado en el álgebra de procesos, el cual está completamente alineado con el enfoque de la etapa anterior, ya que utiliza como insumos sus salidas. Es importante mencionar que el análisis de eficiencia de las soluciones queda fuera del alcance de este trabajo aunque se tiene la hipótesis que se puede lograr a través del uso de las bases aquí mencionadas.

Actualmente se conocen dos enfoques ampliamente utilizados en la generación de programas concurrentes, lo cuales transforman soluciones secuenciales en paralellas: OpenMP [DM98] Y MPI [SOHL$^+$98], sin embargo, no se puede decir que estos formen parte del proceso ya que no existe una conexión con las etapas anteriores, solo permiten paralelizar una solución aunque no sea posible su verificación.

Esfuerzo similares se han realizado en la última etapa del proceso, como los trabajos de [RS94] y [KA99] que están dedicados al análisis de la planificación de procesos concurrentes en tiempo de ejecución, partiendo de una solución no verificada y llegando a una ejecución completa pero no es posible mejorar la eficiencia en esta etapa del proceso debido a que la solución dada puede ser ineficiente en todas sus ejecuciones.

El propósito de seguir el modelo para resolver problemas a través de la programación concurrente es que la solución a un problema dado vaya acercándose cada vez mas a un programa concurrente que pueda ser ejecutado en una computadora, cuidando que sea correcta, completa y eficiente.

4.2. El objetivo

El presente trabajo tiene como propósito la definición de una representación algorítmica, como parte de la solución al problema de la programación concurrente, que refleje la concurrencia de la solución (interacción entre procesos) y pueda ser verificado para garantizar, entre otras propiedades,

que las soluciones planteadas con este enfoque no caigan en deadlock o existan procesos en estado de inanición.

4.3. El problema

En nuestros días la programación concurrente sigue siendo un proceso incierto que en muchas ocasiones no recompensa el esfuerzo invertido en ella cuando se llega a una solución menos eficiente que su contraparte secuencial o la diferencia es mínima. Por esa razón es importante tratar de generar un método que pueda servir de guía para seguir el proceso de resolución de problemas en cada una de sus etapas y que las abstracciones generadas:

- Siempre sean apegadas al problema que se pretende resolver.
- Se encuentren relacionadas a través de un mapeo.
- Mantengan sus características.

El proceso para resolver problemas por medio del desarrollo de programación paralela de Pancake y Bergmark antes mencionado sirve como base para generar dicho método. El primer paso del proceso ya está dado, debido a que cada vez son mas utilizados los patrones de software como elementos que definen el comportamiento que puede tener un sistema de forma abstracta, por ejemplo, con notación de ingeniería de software.

Generar las descripciones algorítmicas de los problemas a partir de la representación anterior, buscando un mapeo para ello, permite dar certidumbre a su definición y garantizar la concurrencia al momento de tomar cada uno de los procesos secuenciales que generó el algoritmo y colocarlos como piezas de código independientes que establecerán comunicaciones durante su ejecución.

También es de gran relevancia que los algoritmos generados no tengan la estructura de algún lenguaje de programación, ya que en este nivel de abstracción se pueden tomar decisiones acerca del entorno donde se ejecutará el programa y tomando los algoritmos solamente, será posible migrar a uno u otro lenguaje de forma sencilla y directa.

Por lo tanto, el problema que se pretende abordar es la falta de una metodología que permita

resolver problemas con soluciones concurrentes partiendo solamente de su definición y cuyo resultado una representación de la solución en forma de código en algún lenguaje de programación que sea correcta, completa y eficiente.

4.4. Relevancia

Abordar problemas y resolverlos con concurrencia a través de un proceso establecido es útil para homologar el proceso de análisis, verificación y construcción de sistemas de este tipo. Uno de los pasos claves es la generación de representaciones que ayuden en el análisis de la solución y permitan una transición sencilla hacia su implementación.

El enfoque de este trabajo es diferente a los mencionados en los capítulos 2 y 3 debido a que el uso del álgebra de procesos permite realizar un análisis de la concurrencia de forma secuencial con el propósito recuperar la traza de cada proceso involucrado aunque las acciones que los conforman sucedan de forma simultánea.

Con los modelos basados en Sistemas de acción mencionados en el capítulo 3 no es posible recuperar la traza del proceso, debido a que su enfoque está orientado a modelar las acciones que percibe el usuario desde fuera del sistema, planteando restricciones sobre el comportamiento de los procesos pero no existe alguna forma de reflejar su ejecución por separado, por lo tanto no es posible definir definir algoritmos concurrentes.

Por otro lado, los modelos basados en lógica temporal si pueden analizar este comportamiento pero sería necesario establecer una serie de condiciones y axiomas que permitan evaluar este comportamiento desde su semántica.

Finalmente, desde el punto de vista de RTPA es posible crear comportamientos paralelos muy complejos pero la semántica está definida por tipos de elemento y no existe un concepto de traza que permita visualizar la ejecución de cada tarea por separado.

Por otro lado, todos los enfoques mencionados tienen mecanismos para verificación de sistemas concurrentes pero no para determinar algoritmos concurrentes, es decir, realizan un mapeo directo de la definición del problema a la implementación y no se contempla el análisis de eficiencia de la

solución.

En el enfoque utilizado en este documento la solución de un problema empieza definiendo un modelo de alto nivel a través de ecuaciones de álgebra de procesos y otros elementos básicos que permiten desglosar el problema en partes; posteriormente se realizan operaciones algebraicas para asegurar que la solución cumple (o no cumple) con determinadas características y como producto de esa manipulación matemática surge la nueva abstracción del modelo, que es la definición algorítmica de la solución.

4.5. Resumen

El propósito del presente capítulo es realizar un análisis del proceso propuesto por Pancake y Bergmark para resolver problemas por medio de paralelización, el cual comienza con el planteamiento de una solución para un problema dado por medio de una representación de alto nivel y termina con un ejecutable en un entorno determinado. Cada etapa del proceso implica realizar una representación de la solución a niveles de abstracción distintos y para lograrlo es necesario incluir algunos pasos intermedios para asegurar que la solución sea correcta, completa y eficiente al pasar de una representación a otra y que cada etapa se conecta con la anterior por medio de un mapeo entre una representación y otra.

Aunque es cierto que actualmente existen enfoques que permiten general programas concurrentes, estos no están apegados a este proceso, lo cual puede traer como consecuencia que se construyan implementaciones con comportamientos no deseados y que solo podrán ser visibles después de realizar todo el esfuerzo para construir el programa y ejecutarlo.

También existen enfoques que puede ayudar a realizar algún paso del proceso en específico pero es necesario que se encuentren relacionados para que la salida de una etapa pueda servir como entrada en la siguiente.

Este trabajo pretende abordar los dos primeros pasos del proceso, el primero a través de un enfoque existente (los patrones para el diseño de software paralelo) y el segundo utilizando como herramienta el álgebra de procesos. Su propósito es realizar una descripción algorítmica de los

procesos que deben ser programados en forma secuencial, así como la definición de las funcionalidades del sistema y el modelo de comunicaciones de manera explícita ya que es el elemento mas importante en la interacción de los procesos concurrentes que forma un sistema.

Actualmente no existen metodologías para transformar una solución concurrente representada de forma abstracta en un programa de computadora, que contemplen la verificación del sistema y tomen en cuenta aspectos de eficiencia. En el enfoque de este trabajo se realiza la verificación pero no pretende brindar un análisis de eficiencia, solo brindar bases teóricas que permitan realizarlo por medios algebraicos.

Debido a lo anterior, el problema de programación con procesos concurrentes no termina con la generación de algoritmos, ya que además del análisis de eficiencia existen otros retos en la etapa de generación del programa como la adaptación del programador a los lenguajes de programación existentes o la forma en que cada lenguaje implementa las operaciones relacionadas con concurrencia como la paralelización de procesos o las comunicaciones, así como en la etapa de ejecución de procesos, como el balance de cargas para lograr ejecuciones mas eficientes.

5. Generando algoritmos concurrentes

Para lograr el análisis de un sistema concurrente es necesario realizar un modelo de representación que permita conocer su comportamiento de un sistema, a través de aspectos como:

- La instrucción que se está ejecutando.
- Los valores de las variables.
- Las instrucciones que le restan por ejecutar.
- Las interacciones que se establecen entre procesos.

Estos elementos definen el **estado** del sistema, el cuál está compuesto por los valores estáticos que pueden tomar las variables asociadas a uno mas procesos que se relacionan entre sí. En el presente trabajo se propone una representación simplificada del entorno de procesamiento de sistemas concurrentes, de tal forma que sea posible identificar sus estados, transiciones y los procesos que ejecutan las acciones necesarias para que el sistema cambie de estado.

La unidad básica de representación algorítmica para un problema resuelto por medio de programación concurrente es el proceso secuencial, donde cada instrucción debe terminar satisfactoriamente para ejecutarse la siguiente, por lo que solo puede existir una ejecutándose en un momento en el tiempo por el procesador y aunque dicho proceso incluya muchos subprogramas o rutinas, siempre estará relacionado con un solo flujo de ejecución. La unión de de dos o mas procesos secuenciales que tienen una interacción entre ellos y se planea que su ejecución sea simultánea es un sistema concurrente y si los procesos que compone al sistema pueden tomar procesadores distintos, entonces el sistema se vuelve paralelo.

5.1. Modelo de representación

El cambio de arquitectura de las computadoras de uso comercial ha tenido una gran influencia en el campo de la programación debido a que las arquitecturas multicore demandan un nuevo paradigma que explote al máximo sus capacidades, como la programación concurrente, la cual existe desde hace varias décadas, aunque al día de hoy no existe un consenso en la forma de realizarla tal como sucede con otros tipos de codificación, por lo que llevar un problema a una representación con código en algún lenguaje de programación es una tarea complicada.

Con la finalidad de realizar un análisis que permita resaltar los elementos principales de los sistemas concurrentes y proponer una forma estructurada de realizar un programa concurrente se definirá un marco de trabajo que tiene varios elementos de una arquitectura multicore, aunque con algunas condiciones ideales para simplificar su definición y manejo.

También será necesario definir la representación de los problemas planteados, dependiendo la etapa en que se encuentren, de acuerdo al modelo mencionado en el capítulo anterior y al propio alcance del presente trabajo.

5.1.1. Marco de trabajo

El marco de trabajo que será utilizado para delimitar el modelo con el que se representarán los sistemas concurrentes será un entorno con mas de una unidad de procesamiento, que puede ejecutar mas de un flujo de instrucciones a la vez y cada unidad tiene su propio almacenamiento de datos, es decir, obedece a la arquitectura MIMD [FR96]. Se puede ver como un arreglo de unidades de procesamiento $\{C_1 \cdots C_n\}$ donde cada uno puede ejecutar un determinado conjunto de instrucciones distinto con su propio espacio de memoria $\{M_1 \cdots M_n\}$ simultáneamente.

Los procesos ejecutados en esta arquitectura son controlados por el mismo reloj, por lo tanto siempre es posible saber el orden de ejecución de las instrucciones y cada procesador de instrucciones está sincronizado con el reloj global. Este modelo garantiza que un evento de comunicación γ no tendrá retardos una vez que sea ejecutado. La figura 5.1 esquematiza el modelo completo.

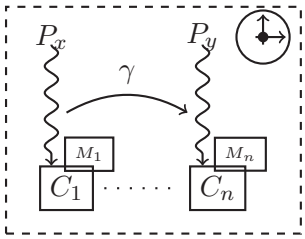

Figura 5.1.: Marco de trabajo

5.2. Proceso de solución

El proceso que se siguió en el presente trabajo para la obtención de una representación algorítmica para un problema que se resuelve con procesos concurrentes está compuesto de tres grandes fases: Análisis del problema, planteamiento algebraico y verificación algebraica, cada una de las cuales arroja un producto que es una representación distinta del problema a resolver, siempre tendiente a una representación en código de computadora.

Cabe mencionar que cada fase también cuenta con pasos intermedios para lograr obtener la representación deseada. En la figura 5.2 se esquematiza el proceso completo y en las siguientes secciones se detallará.

5.2.1. Análisis del problema

Como ya se mencionó en la sección 4.1, el procedimiento para atacar un problema que se pretende paralelizar comienza con el entendimiento y análisis del mismo para delimitar su dominio y elegir una estrategia para resolverlo. Aunque no hay un método establecido para realizar esta labor, esta parte resulta la menos abstracta y es posible utilizar diagramas, fórmulas, esquemas o cualquier forma de representación que ayude a reflejar en un lenguaje conocido la solución que se pretende adoptar.

En el caso particular del presente trabajo, se ha elegido utilizar en esta etapa el método basado en *Patrones para diseño de software en paralelo*[OA10] como la estrategia para realizar el modelo abstracto de la solución a problemas por medio de procesos concurrentes, debido a que contiene

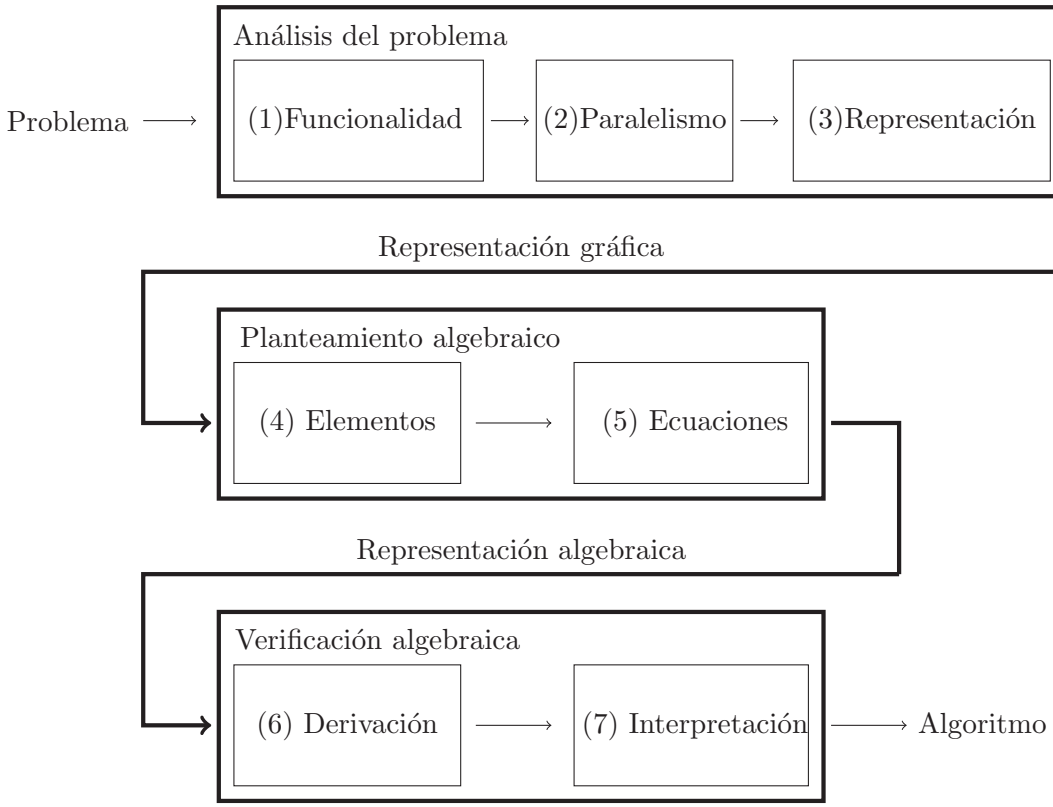

Figura 5.2.: Esquema del proceso de solución

especificaciones para realizar el análisis de mas alto nivel de un problema logrando una representación gráfica de acuerdo al contexto del problema. A continuación se enuncian algunas de sus características:

- La documentación asociada con un patrón paralelo tiene una estructura definida y está basada en UML para facilitar su comprensión.
- Se tienen varios niveles de abstracción que hacen posible modelar diferentes componentes de un sistema. Esta característica es útil porque permite combinar los elementos estructurales propios de los patrones con términos de la Signatura para obtener una representación algebraica de una forma sencilla pero apegada a la solución gráfica.
- Los patrones de mas alto nivel - los arquitectónicos - se encuentran organizados de tal forma que, al identificar el tipo de paralelismo y el tipo de procesamiento adecuados, es posible elegir

el patrón adecuado para la solución al problema. Determinar el tipo de paralelismo es útil para mejorar el entendimiento del problema, realizar una representación útil y organizarlo de la forma mas conveniente, identificando una organización funcional que permita dividir al sistema en tareas bien definidas sobre un dominio de datos establecido.

- Finalmente, un patrón paralelo trae con su definición un cúmulo de experiencia que garantiza su funcionamiento por lo que constituye una base sólida sobre la que se puede iniciar el proceso de resolución de problemas con programación concurrente.

Identificar funcionalidades y datos

Cuando se tiene necesita abordar un problema el primer paso es identificar las distintas funcionalidades que serán necesarias y el conjunto de datos que entran y salen del sistema. La forma mas simple de ver un proceso, tal como lo enuncia [Bae05], es como una función de entrada-salida(Figura 5.3), vista como una "caja negra" que tiene algunos espacios para recibir entradas y del otro lado arroja las salidas de su ejecución.

Figura 5.3.: Modelo de caja negra

Esta representación es útil para esquematizar el comportamiento del sistema y establecer relaciones entre bloques disjuntos que pueden depender unos de otros e identificar si existe mas de un bloque que realiza la misma funcionalidad.

Otro aspecto a tomar en cuenta es la forma que recibe las entradas el sistema. Puede suceder que los datos pueden dividirse para ser procesados en paralelo o la entrada de datos solo la recibe un proceso.

Tomando en cuenta ambos aspectos se puede identificar el tipo de sistema que se requiere, de acuerdo con la clasificación de [OA10], la cual realiza una división de los tipos de sistemas concurrentes tomando en cuenta si todos los procesos que los forman son idénticos y ejecutan la misma funcionalidad (**Sistemas Homogéneos**) o si son piezas de código con funcionalidades

diferentes (**Sistemas Heterogéneos**).

Identificar el tipo de paralelismo

La definición del tipo de paralelismo es uno de los pasos mas importantes y solo puede llevarse a cabo una vez que se ha realizado el análisis del problema anteriormente mencionado.

Una de las motivaciones principales para identificar el tipo de paralelismo (y en general para hacer un sistema concurrente) es la búsqueda de un desempeño mayor al que se podría lograr si se atacara el problema con programación secuencial, inclusive resulta muy atractiva la idea de explotar la capacidad de un equipo multicore, e idealmente lograr ambas es una buena razón para realizar este proceso. Sin embargo, también existen otras razones para ser minuciosos en el análisis de la mejor organización para un sistema elaborado con procesos concurrentes, por ejemplo, cuando se tienen limitaciones en hardware, como lo comenta [SSOG93].

Determinar el tipo de paralelismo también puede determinar la plataforma de desarrollo, el lenguaje de programación([OA10]) e inclusive la técnica a utilizar para construir el sistema.

Para generar los procesos necesarios y así definir el sistema concurrente es necesario analizar los dos aspectos fundamentales antes mencionados,la organización de las distintas funcionalidades y la forma en que se pueden dividir y procesar los datos. Tomando estos factores en cuenta, Ortega-Arjona y Roberts en [CD99] especifican tres tipos de organización de tareas o tipos de paralelismo:

paralelismo funcional. Se puede ver como una colección de procesos secuenciales con comunicaciones explícitas entre ellos.

paralelismo de dominio. Se comporta como un hilo de control que opera en conjunto con los datos distribuidos en los demás nodos.

paralelismo de actividad. Pretende explotar ambos estilos de paralelismo en el mismo sistema.

Construir la representación gráfica

Una vez identificadas las funcionalidades, el domino de datos y el tipo de paralelismo se puede utilizar la tabla 5.1 ([OA10]) que clasifica los patrones de tipo arquitectónico de acuerdo al tipo

de paralelismo identificado para el problema a resolver y el tipo de procesamiento del sistema, lo cual constituye una buena guía para determinar el patrón que se debe utilizar en un problema determinado.

Patrón	Tipo de paralelismo			Tipo de procesamiento	
Arquitectónico	Funcional	Dominio	Actividad	Homogéneo	Heterogéneo
Parallel Pipes and Filters	x				x
Parallel Layers	x			x	
Communicating Sequential Elements		x		x	
Manager - Workers			x	x	
Shared Resources			x		x

Tabla 5.1.: Clasificación de patrones arquitectónicos

La elección del patrón permite definir la organización de procesos que formarán el sistema concurrente y ayuda a identificar las comunicaciones que se establecerán entre ellos, para definir posteriormente el conjunto de datos que se pueden transmitir, así como los canales de comunicación.

5.2.2. Planteamiento algebraico

El propósito principal de utilizar los patrones de software es realizar un mapeo entre sus elementos con los términos algebraicos y así representar la solución planteada de forma diferente sin alterar la estructura previamente determinada.

Al tener una organización de procesos definida, es posible obtener una representación algebraica del sistema concurrente estableciendo relaciones entre la representación gráfica y los elementos algebraicos necesarios para realizar la definición:

- Los elementos funcionales del diagrama (los procesos) se representan como términos recursivos custodiados, con base en el álgebra de procesos definida a continuación.
- Los componentes de los procesos son las acciones que realiza cada proceso, las cuales están perfectamente ubicadas en el álgebra de procesos dentro del conjunto A.
- Los enlaces entre proceso son los canales de comunicación, los cuales también se encuentran bien especificados.

Todos los elementos anteriores son necesarios para lograr una definición algebraica que sea equi-

valente al esquema obtenido en el paso anterior.

Notación algebráica

Para realizar la representación algebraica de un problema, se debe comenzar por definir un conjunto llamado A compuesto por las acciones atómicas, las cuales se representan con letras minúsculas y son términos cerrados desde el punto de vista sintáctico, es decir, no aceptan parámetros o modificadores.

Algunas acciones atómicas que comúnmente se definen son:

- Acciones secuenciales
- Acciones de comunicación

Para facilitar la representación de sistemas concurrentes se considerará como una acción atómica a un conjunto de acciones secuenciales que suceden en el dominio un proceso y no tienen ninguna comunicación con algún otro, por ejemplo, una suma o alguna mas compleja como un ordenamiento. En general, su representación es de la forma $x =$'Función o Acción', donde $x \in A$ y el enunciado después del signo igual puede ser tan explícito como se desee.

Las acciones de comunicación se definen a partir de tres elementos, un evento de envío, uno de recepción, y el conjunto de datos a enviar. Por ejemplo, para definir el envío del dato '1' por el canal A se debe especificar:

- $a \triangleq$ Enviar el dato '1' por el canal A.
- $b \triangleq$ Recibir el dato '1' por el canal A.
- $\Delta \triangleq \{0, 1\}$

A partir de los elementos anteriores se puede definir la comunicación entre las acciones a y b de la siguiente forma: $c \triangleq \gamma(a, b)$.

Esta definición de comunicación implica que la acción a se encuentra en un proceso y b en otro, que el canal A une solo a esos procesos, por lo que está disponible siempre que se quiera utilizar por ellos y que este evento es también una acción atómica garantizada, de acuerdo con el marco de

trabajo antes definido.

La Signatura del álgebra utilizada está compuesta del conjunto A de acciones atómicas definido anteriormente y de un conjunto de operadores unarios (∂_H,τ) o binarios ($+$,$\cdot$,$||$) sobre los elementos de A, con los que se forman los términos algebraicos utilizados en la definición de un problema, donde:

- Sean $a, b \in A$
- $a + b$ Significa que se puede ejecutar a o bien b.
- $a \cdot b$ Significa que b se ejecuta después de a.
- $a||b$ Significa que a y b se ejecutarán concurrentemente.
- $\tau(a) \cdot b$ Significa que a se ejecutará como una acción interna del sistema, por lo que el sistema solo arrojará en la traza que b se ejecutó.
- $\partial_H(a + b)$ con $a \in H$, significa que si se ejecuta a el sistema caerá en deadlock, por lo que $\partial_H(a + b) = b$, ya que $x + \delta = x$ (Axioma 16, apéndice B).

Un término puede formarse de otros términos utilizando los símbolos mencionados, paréntesis como símbolos de agrupación y se representan con letras mayúsculas.

Ejemplos de términos:

- $X = a + b$
- $Y = a \cdot (b||c)$
- $W = \tau(X)$
- $Z = \partial_H(Y + W)$

También es posible definir ecuaciones recursivas para representar comportamientos infinitos, por ejemplo las ecuaciones

- $X = a \cdot Y$
- $Y = b \cdot X$

Representan la sucesión de acciones atómicas $a \cdot b \cdot a \cdot b$...

En el apéndice A se detallan los elementos del álgebra de procesos antes mencionados, incluyendo la notación utilizada para definir los elementos de un sistema y establecer las ecuaciones que definan su comportamiento.

Mapeo de la solución

La definición gráfica obtenida durante el análisis del problema con el método antes expuesto contiene ciertos elementos bien definidos:

1. Dos o mas términos que representan los procesos involucrados en la solución.
2. Estructuras que permiten la comunicación entre dichos procesos.

   También existen de forma implícita otros elementos:

3. Operaciones internas que procesan los datos entrada.
4. El conjunto de datos de entrada.

Para el caso del conjunto de los datos de entrada (4), su definición se realizará en notación de conjuntos, ya sea explícita o implicita, siempre utilizando como indentificador la letra griega Δ.

Las operaciones internas de los procesos (3) son acciones atómicas y se representarán con letras minúsculas, las cuales pueden estar acompañadas de subíndices cuya utilidad será aclarada en la misma definición. Por ejemplo x, y, z_1 son acciones definidas para un proceso y $x, y, z_1 \in A$.

Las estructuras de comunicación (2) son elementos bien definidos en el álgebra de procesos utilizada en el presente trabajo, se representan con la letra *gamma* e involucran a dos elementos de procesos distintos, donde uno de ellos se define explícitamente para transmitir un dato del conjunto Δ y el otro para recibirlo.

Ejemplo. Sea $\Delta = \{0, 1\}$, A el conjunto de acciones atómicas y $a, b \in A$ definidas como sigue, $a_3(1) \triangleq$ Envía el dato 1 por el canal 3. $b_3(1) \triangleq$ Recibe el dato 1 por el canal 3. La estructura de comunicación corespondiente se define como otra acción atómica $c \triangleq \gamma(a, b)$ o también $c \triangleq \gamma(b, a)$, en realidad el orden no importa debido a la naturaleza misma de la definición de las acciones.

Es importante hacer notar que las definiciones de los elementos se denotarán con el símbolo $\triangleq$

ya que el signo "=" denota la relación de igualad entre dos elementos.

Las definiciones anteriores se puede generalizar para incluir mas de un elemento del conjunto Δ si se utiliza una variable (generalmente la letra d) para representar a cualquier elemento del conjunto. Retomando el ejemplo antes mencionado se puede cambiar su definición de la siguiente forma: Sea $d \in \Delta$, $a, b \in A$ de definen como sigue,

$a_3(d) \triangleq$ Envía el dato d por el canal 3.

$a_3(d) \triangleq$ Recibe el dato d por el canal 3.

$c \triangleq \gamma(a, b)$

Es importante resaltar que incluir a la variable d en la definición de a_3 y b_3 aporta generalidad pero no cambia su esencia debido a que se conserva la propiedad de que el dato que envía el proceso a_3 es el mismo que recibe el proceso b_3.

Estrictamente, la notación $a_3(d)$ con $d \in Delta = \{0, 1\}$, por ejemplo, corresponde a $a_3(0) + a_3(1) = \Sigma_{d \in \Delta} a_3(d)$, lo cual denota que se utilizará el término atómico $a_3(0)$ en el caso de que el dato a enviar sea cero o $a_3(1)$ si el dato es 1. Sin embargo, para simplificar las derivaciones no se incluirá el símbolo de sumatoria (Σ) y se asume que el término $a_3(d)$ se interpreta de la misma forma que $\Sigma_{d \in \Delta} a_3(d)$.

Es posible realizar una generalización mas si como parte de la solución se definiera un conjunto de canales de comunicación $C = \{c_1, c_2...\}$ y se utilizara $x \in C$ para definir a y b de la siguiente forma:

$a_x(d) \triangleq$ Envía el dato d por el canal x.

$b_x(d) \triangleq$ Recibe el dato d por el canal x.

Finalmente, los procesos involucrados en la solución (1) serán representados en forma de una igualdad donde el término de la izquierda será una letra mayúscula y el de la derecha un término válido en el álgebra de procesos descrita anteriormente, siempre y cuando los elementos que lo componen también se encuentren definidos.

Un proceso se define estableciendo las acciones que ejecutará ya que cada elemento solo puede ver sus propias acciones y bajo esa filosofía se construyen los términos algebraicos que los representarán.

Por ejemplo, es posible definir un proceso que contenga un elemento de una comunicación y que no exista ningún otro que contenga el elemento que hace falta para dicha comunicación.

Para evitar ese tipo de errores la definición de los elementos algebraicos comienza con el conjunto Δ, posteriormente con los elementos del conjunto A, a continuación se establecen los eventos de comunicación y finalmente con los elementos anteriores se forman los procesos.

Continuando con el ejemplo se sumarán las siguientes definiciones a las anteriores,

$j \triangleq$ Sumar dos números x, y tal que $x, y \in \Delta$

$k \triangleq$ Recibe un número del usuario

$l \triangleq$ Escribe el resultado

$R \triangleq k \cdot a \cdot k \cdot a \cdot b \cdot l$

$S \triangleq b \cdot b \cdot j \cdot a$

$T \triangleq R||S$

Donde la definición de T es la representación de una solución por verificar en notación de álgebra de procesos, la cuales finita ya que concluye cuando se ejecuta las últimas acciones de R y de T. Como ya se mencionó, es posible representar procesos infinitos agregando letras mayúsculas del lado derecho de las ecuaciones siempre y cuando sea un elemento definido (se encuentra del lado izquierdo de algún término), por ejemplo,

$R \triangleq k \cdot a \cdot k \cdot a \cdot b \cdot l \cdot R$

$S \triangleq b \cdot b \cdot j \cdot a \cdot S$

$T \triangleq R||S$

Otro aspecto que se debe reflejar en la definición de un término es la forma en que pueden suceder las acciones atómicas,ya que su orden de ejecución puede no ser el mismo en todas las ejecuciones pero comportarse de la misma forma.

Esto se puede reflejar en el término R anteriormente definido de la siguiente forma:

$R \triangleq ((k \cdot a \cdot k \cdot a) + (k \cdot k \cdot a \cdot a)) \cdot b \cdot l \cdot R$

Por lo tanto, ahora el término R puede recibir un número del usuario, enviarlo y luego repetir el proceso o bien puede recibir primero dos números y posteriormente enviarlos en las siguientes dos acciones. Los demás elementos de R no pueden suceder en otro orden, por lo tanto permanecen en el mismo lugar.

La idea general es realizar una definición exhaustiva para contemplar un número mayor de posibles ejecuciones del proceso.

Mapeo de la propiedad a verificar

El enunciado del problema puede plantear de forma explícita las propiedades que debe cumplir la solución o solo encontrarse implícitas en él; puede plantear ciertas condiciones que son deseables para el sistema o que no.

Es muy importante lograr identificar cada una de esas características o condiciones y expresarlas en términos del álgebra anteriormente expresada.

Una de las propiedades que se desea evitar en las soluciones concurrentes es que en algún momento de la ejecución un sistema R caiga en deadlock, lo cual se puede representar como $R' = \delta$, donde R es la definición de una solución en términos algebráicos y δ es el estado deadlock.

Por otro lado, si un sistema es finito, es posible representar su terminación por medio de la igualdad $R = \surd$.

Otras propiedades que se pueden representar son:

- Ningún estado del sistema es igual a deadlock. Sea el conjunto de estados del sistema S y $R \in S$, entonces $R \neq \delta$.
- Se ejecuta alguna acción en el sistema. Sea $ainA$, entonces existe un estado en el cual se tiene $R = a \cdot Q$ donde Q es un término que se ejecutará después de a o $Q = \surd$.
- Se ejecutan dos acciones concurrentemente. Sean $a, b \in A$, $R = a \cdot Q || b \cdot Q'$, donde Q tiene el mismo significado y Q' también pero no son iguales necesariamente.
- Existe una comunicación entre dos términos. Para $a, b \in A$, existe la definición $\gamma(a, b)$ y en algún estado del sistema $R = a \cdot Q | b \cdot Q'$.

5.2.3. Verificación algebraica

El propósito de la derivación algebraica es verificar que la solución satisfaga las especificaciones del problema o que no realice un comportamiento indeseable.

Una vez definido el sistema algebraico que se utilizará para representar una solución y la propiedad a demostrar en términos algebraicos, es posible comenzar con la derivación algebraica para llegar a la conclusión deseada a partir de pasos sucesivos.

El proceso de verificación inicia con la representación algebraica de la solución y termina de forma satisfactoria cuando es posible llegar a la representación de la característica deseada o cuando se demuestra que no es posible realizar derivaciones que lleven a la ecuación de estado que representa una característica no deseada.

Como se ya se ha mencionado, el uso del álgebra de procesos para modelar sistemas concurrentes responde a la posibilidad de verificar propiedades deseables (o no deseables) de los mismos. La razón es que su sintaxis y su semántica tienen un sustento lógico, lo cual permite asegurar que la solución es correcta y completa..

Las reglas que permiten la derivación algebraica se encuentran definidas en términos de una relación de igualdad que permite realizar sustituciones de términos gracias a la axiomatización establecida para esta álgebra, la cual se encuentra enunciada en el apéndice B, por lo tanto la forma de manipular términos es a través de derivaciones válidas de acuerdo con la axiomatización para encontrar otros bisimilares.

Derivación algebraica

Una vez definidas las ecuaciones es posible comenzar el proceso de derivación, el cual es equivalente a realizar la simulación de una ejecución paso por paso y su propósito es definir cual será la siguiente acción atómica a ejecutarse, por ejemplo, el término $X = a \cdot b$ ejecutará la acción a y se transformará en $X' = b$, denotado como $X \xrightarrow{a} X'$ y a formará parte de la traza del sistema.

Cuando se trabajan con términos compuestos de acciones atómicas, se deben sustituir las letras mayúsculas por su composición. por ejemplo, para las ecuaciones:

- $X = a + b$
- $Y = c \cdot d$
- $Z = X \cdot Y$

Se necesita sustituir X ya que la siguiente acción a ejecutar es la primera que se encuentre dentro de este término, por lo tanto, $Z = (a + b) \cdot Y = a \cdot Y + b \cdot Y$.

Cuando se trabaja con mas de dos procesos concurrentes se debe tomar en cuenta que en cada paso de derivación pueden participar a lo mas dos, mientras que los otros no cambian su estatus. Por ejemplo, si se tiene el sistema $S = P||Q||R$, P podría comunicarse con Q o con R pero no con ambos al mismo tiempo, si se supone el primer caso, entonces la ecuación se desarrollaría de la siguiente forma $S = (P||Q)||R$, realizando la derivación $P||Q$ y conservando a R sin cambios. Aplicando el axioma $p||q = p\lfloor\lfloor q + q\lfloor\lfloor p + p|q$ (A06 del apéndice B) se puede esperar uno de los cuatro 4 resultados siguientes:

1. $S = a \cdot (P'||Q||R)$ donde a es una acción atómica local de P que lo hizo evolucionar a P'
2. $S = b \cdot (P||Q'||R)$ donde b es una acción atómica local de Q que lo hizo evolucionar a Q'
3. $S = a \cdot (P'||Q||R) + b \cdot (P||Q'||R)$ donde a y b están definidos como en las opciones anteriores pero en este caso ambas acciones atómicas podrían ejecutarse.
4. $S = \gamma(a, b) \cdot (P'||Q'||R)$ donde a y b están definidos como en las opciones anteriores pero son los componentes de una acción de comunicación$\gamma(a, b)$, por lo que su ejecución hace evolucionar tanto a P como a Q.

En consecuencia, el proceso completo de derivación de una ecuación con procesos concurrentes debe contemplar en cada paso todas las posibles interacciones entre pares de procesos, es decir, un paso de derivación para n procesos en ejecución tiene $\binom{n}{2}$ posibilidades, formando un árbol finito (ver sección 6.3) donde cada nueva ramificación es un camino de la ejecución del sistema, los cuales serán vistos como una ejecución exitosa si en ningún paso de derivación caen en deadlock, lo cual quiere decir que cada vez que en el proceso de derivación existe un estado con deadlock inmediatamente se analiza otra ramificación de la derivación.

Volviendo al ejemplo $S = P||Q||R$, la figura 5.4 muestra las opciones para realizar el primer paso

de derivación:

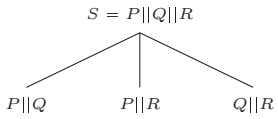

Figura 5.4.: Opciones de un primer paso de verificación para la ecuación $S = P||Q||R$

Tomando las opciones antes mencionadas, se desarrolla el primer paso para la opción $P||Q$ en la figura 5.5 :

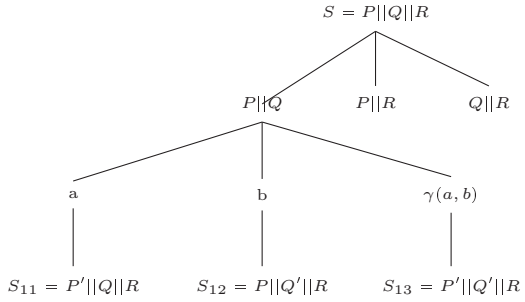

Figura 5.5.: Resultados del primer paso de verificación para la ecuación $S = (P||Q)||R$

En el primer paso del ejemplo alguna de las ramas resultantes caerá en deadlock, ya que si alguna de las acciones a o b no lleva al sistema a deadlock, lo hará la acción $gamma(a, b)$, aunque puede suceder que todos los caminos caigan en deadlock y entonces se tratará con la siguiente rama, es decir $P||R$ y en automático se realiza una "poda" de toda la rama donde se detectó el deadlock, es decir, ya no se toma en cuenta, resultado un proceso de backtrack que busca las secuencias de estados donde el sistema se ejecuta correctamente (no cae en deadlock).

Una forma inmediata de identificar ramas que no valen la pena explorar es observando la próxima acción a ejecutar de cada proceso: si tienen acciones internas por ejecutar se pueden tomar en cuenta inmediatamente y el árbol de ejecución tendrá un nivel mas, cambiando el estado de un elemento a la vez; si la siguiente acción es un evento de comunicación es posible notar mediante las definiciones de las comunicaciones que caerá en deadlock, que son aquellas que no se complementan con su contraparte. Por otro lado, las acciones atómicas definidas dentro de los procesos que no tienen

que ver con comunicación se ejecutarán siempre, por lo que solo se abrirá una rama posible de ejecución.

Interpretación de la traza

A partir del proceso de derivación algebraica es posible obtener una representación algorítmica de los procesos que lo componen a través de la traza de ejecución que se genera. A continuación se describirá la forma de obtener dicha representación a partir de este proceso y en el capítulo 6 se explicará porque es posible hacerlo.

Un detalle mencionado por Pancake y Bergmark en [PB90] con respecto a la representación algorítmica usual en sistemas concurrentes es que normalmente se tiene un algoritmo secuencial y algunas anotaciones acerca de como paralelizar alguna de las partes, sin embargo en el presente trabajo se pretende, como se ha mencionado anteriormente, brindar una representación algorítmica concurrente que sea un mapeo de otra descripción de mas alto nivel, pero asegurando que sea correcto, es decir, que siga teniendo las características originales del sistema y que sea capaz de reflejar:

- La existencia de dos o mas procesos ejecutando tareas concurrentemente.
- La interacción de dichos procesos a través de mecanismos de comunicación
- La definición de tareas secuenciales que serán ejecutadas por cada proceso.

Por lo tanto, la caracterización de dicho algoritmo será un poco diferente a los modelos que integren las acciones de los distintos procesos en un solo código, por ejemplo aquellos que se definen con estructuras de tipo *do* . . . *od* [Dij75]. El concepto de algoritmo concurrente que se propone para el presente trabajo es el siguiente:

> *Conjunto* $\mathbb{A}$ *conformado por dos o mas algoritmos que contienen entre sus instrucciones algún método de interacción con al menos algún otro algoritmo de* $\mathbb{A}$*. Cada elemento se construye secuencialmente pero siempre tomando en cuenta que su ejecución será concurrente.*

Además de estructuras de control, variables y otros elementos comunes en algoritmos secuenciales,

los elementos que integrarán los algoritmos concurrentes son el conjunto de acciones atómicas A que se definen cuando se construye la Signatura de la solución a un problema dado. Esto es conveniente por dos razones, la primera es que el proceso de derivación trabaja con estos elementos y no se necesita realizar ninguna conversión para definir el algoritmo y segundo, porque la definición de estos elementos se hace pensando en que serán implementados, salvo la comunicación, la cual debe estar definida en el lenguaje de programación que se elija.

Con el propósito de aumentar la expresividad del algoritmo se pueden cambiar los términos algebraicos por su definición en lenguaje natural sin que esto afecte en alguna forma la validez de la solución.

A partir del análisis del comportamiento del sistema también es posible determinar las acciones que realizó cada proceso por separado, es decir, obtener el algoritmo para cada proceso, ya que el método de derivación establece que se debe trabajar una acción a la vez, excepto en las comunicaciones y aún en ellas cada proceso debe tener una acción de envío o recepción definida, por lo que siempre es posible recuperar cada acción ejecutada en cada paso de derivación.

Una de las propiedades mas importantes para todo sistema es que no caiga en deadlock, ya sea verificando que el sistema siempre termina o para el caso de sistemas con procesos infinitos, encontrando los diferentes estados por los que el sistema puede pasar. En cualquiera de los dos casos, la derivación algebraica realizada permite obtener las acciones atómicas que ejecutó cada proceso y el estado al que evolucionó el sistema, por lo que resulta posible recopilar las acciones por proceso y determinar su secuencia.

En el caso de sistemas con procesos infinitos es necesario analizar muy bien sus estados debido a que resulta necesario colocar sentencias de control iterativas cuando el sistema cae en un estado en el que había caído previamente.

A través de la verificación para comprobar la ausencia de deadlock se pueden obtener todos los posibles algoritmos que se pueden generar, no solo uno, debido a que explora todos los posibles estados del sistema y gracias a la axiomatización del álgebra de procesos se pueden determinar varias soluciones algorítmicas correctas, aunque una explicación mas amplia de este fenómeno se puede ver en el siguiente capítulo en el apartado 6.3.2.

En el caso de sistemas finitos también es posible obtener un algoritmo concurrente sin explorar todos los estados del sistema, el cual estará libre de deadlock si las derivaciones pasan siempre por estados válidos y se puede llegar a $\surd$.

5.3. Ejemplos de aplicación

5.3.1. Problema 1: Buffer de dos pasos

Con el propósito de ejemplificar cada una de las etapas del proceso anteriormente detalladas se propone un problema muy sencillo cuya solución es concurrente y que a su vez permite utilizar todos los conceptos asociados con la búsqueda de una representación algorítmica. El enunciado del problema es el siguiente:

> **Problema 1:** `Buffer de dos pasos`
> *Se requiere un programa que reciba datos binarios por el canal R; se invierta el dato (cambiar cero a uno y viceversa); se coloque el valor a la izquierda del número N (cuyo valor inicial es 11), se transmita por el canal E el último dígito de la derecha de N y se elimine. Se debe transmitir aún cuando el sistema se encuentre recibiendo.*

Análisis del problema

En este caso se pueden identificar dos funcionalidades (Paso (1) Funcionalidad) que son: 1) la que recibe datos e invierte el número y 2) aquella que concatena el dato y realiza la transmisión. Desde el punto de vista del modelo de caja negra, la entrada de la segunda es la salida de la primera, como se esquematiza en la figura 5.6. Nótese que al separar las funcionalidades es es posible realizar la recepción de datos al mismo tiempo que se transmite, además el comportamiento del sistema es similar a un buffer que guarda un dato un tiempo antes de transmitirlo, lo cual permite realizar tareas mientras se recibe otro dato, es decir, existe la concurrencia.

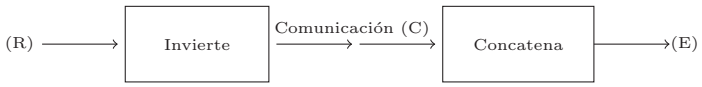

Figura 5.6.: Modelo de caja negra para el problema 1

En este problema se tiene un paralelismo funcional (Paso (2) Paralelismo) debido a que los datos se reciben de forma secuencial pero su procesamiento se realiza de forma paralela, es decir, se tienen procesos secuenciales comunicándose, por lo que se trata de un sistema heterogéneo y cualquier dato de entrada será procesado a través de las funcionalidades identificadas, por lo tanto ambos procesos tiene como dominio de datos a todas las entradas posibles, que en este caso solo son el conjunto $\Delta = \{0, 1\}$. Con esta información es posible determinar que el patrón "Parallel Pipes and Filters" es la opción adecuada de acuerdo con la tabla 5.1.

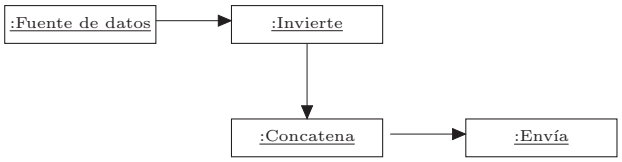

Figura 5.7.: Patrón parallel pipes and filters para el problema 1

Panteamiento algebraico

La figura 5.7 representa la solución al problema planteado en forma gráfica (Paso (3) Representación). Para transformarla a una de tipo algebraico se definirá:

- El dominio de datos (conjunto $Delta$).
- Las acciones que caerán en deadlock al ejecutarse (Conjunto H).
- Las acciones secuenciales (f, g).
- El conjunto de acciones internas que no es necesario evidenciar al usuario (Conjunto I).
- Las acciones de comunicación (b_1, b_2, a_2, a_3) y los eventos de comunicación que los relacionan (c_2) de acuerdo al patrón obtenido.
- Los procesos concurrentes (P, Q) de acuerdo al patrón obtenido (Paso (4) Elementos).
- La ecuación que representa al sistema completo, en este caso R.

La definición 5.1 contiene todos los elementos antes mencionados.

$$
\begin{aligned}
\Delta &\triangleq \{0,1\} \\
H &\triangleq \{a_2(d), b_2(d) | \forall d \in \Delta\} \\
I &\triangleq \{c_2(d) | d \in \Delta\} \\
f &\triangleq \text{Invierte un número binario} \\
g &\triangleq \text{Concatena a la izquierda de N un número binario y separa el de la derecha} \\
b_1(d) &\triangleq \text{Recibe el dato d por el canal 1} (d \in \Delta) \\
b_2(d) &\triangleq \text{Recibe el dato d por el canal 2} (d \in \Delta) \\
a_2(d) &\triangleq \text{Envía el dato d por el canal 2} (d \in \Delta) \\
a_3(d) &\triangleq \text{Envía el dato d por el canal 3} (d \in \Delta) \\
c_2(d) &\triangleq \gamma(a_2(d), b_2(d)) \\
P &\triangleq b_1(d) \cdot f \cdot a_2(d) \cdot P \\
Q &\triangleq b_2(d) \cdot g \cdot a_3(d) \cdot Q \\
R &\triangleq \tau_I(\partial_H(P||Q))
\end{aligned}
\tag{5.1}
$$

Con la representación algebraica obtenida (Paso (5) Ecuaciones) ya es posible realizar la verificación de alguna característica del sistema y una de las mas importantes es la ausencia de deadlock. Como ya se planteó previamente, su representación algebraica es

$$R \neq \delta$$

.

Verificación algebraica

A continuación se enunciarán los estados del sistema en el proceso de derivación (Paso 6) para el problema 1. El desarrollo completo puede encontrarse en el apéndice C.2.

$$\begin{aligned} R =& \tau_I(\partial_H(P||Q)) \\ R =& b_1(d) \cdot \tau_I(\partial_H(f \cdot a_2(d) \cdot P||Q)) \end{aligned} \tag{5.2}$$

Ejecutando $b_1(d)$.

$$\begin{aligned} R_1 =& \tau_I(\partial_H(f \cdot a_2(d) \cdot P||Q)) \\ R_1 =& f \cdot \tau_I(\partial_H(a_2(d) \cdot P||Q)) \end{aligned} \tag{5.3}$$

Ejecutando f.

$$\begin{aligned} R_2 =& \tau_I(\partial_H(a_2(d) \cdot P||Q)) \\ R_2 =& \tau_I(c_2(d)) \cdot \tau_I(\partial_H(P||g \cdot a_3(d) \cdot Q)) \\ R_2 =& g \cdot \tau_I(\partial_H(P||a_3(d) \cdot Q)) \end{aligned} \tag{5.4}$$

Ejecutando g.

$$\begin{aligned} R_3 =& \tau_I(\partial_H(P||a_3(d) \cdot Q)) \\ R_3 =& b_1(d) \cdot \tau_I(\partial_H(f \cdot a_2(d) \cdot P||a_3(d) \cdot Q)) + a_3(d) \cdot \tau_I(\partial_H(P||Q)) \end{aligned} \tag{5.5}$$

Aquí existen dos posibles caminos de ejecución, el primero es cuando se ejecuta $b_1(d)$.

$$\begin{aligned} R_4 =& \tau_I(\partial_H(f \cdot a_2(d) \cdot P||a_3(d) \cdot Q)) \\ R_4 =& f \cdot (\tau_I(\partial_H(a_2(d) \cdot P||a_3(d) \cdot Q))) \end{aligned} \tag{5.6}$$

Luego f.

$$\begin{aligned} R_5 =& \tau_I(\partial_H(a_2(d) \cdot P||a_3(d) \cdot Q)) \\ R_5 =& a_3(d) \cdot (\tau_I(\partial_H(a_2(d) \cdot P||Q))) \end{aligned} \tag{5.7}$$

Al ejecutar $a_3(d)$ el sistema pasa a la ecuación R_1.

Continuando con la otra alternativa de R_3, al ejecutar $a_3(d)$ el sistema pasa al estado R.

Debido a que las dos últimas ejecuciones de acciones atómicas que se analizaron llevan al sistema a un estado existente, significa que el sistema siempre evolucionará hacia algún estado válido (a uno de los que se representan en las ecuaciones), es decir, nunca caerá en deadlock, por lo tanto,

$$R \neq \delta$$

que es lo que se quería demostrar.■

Para obtener el algoritmo se realiza un análisis de los estados por los que pasa la solución propuesta, es decir, la traza resultante del proceso de derivación(Paso (7) Interpretación). La figura 5.8 muestra un esquema donde se pueden ver dichas ecuaciones.

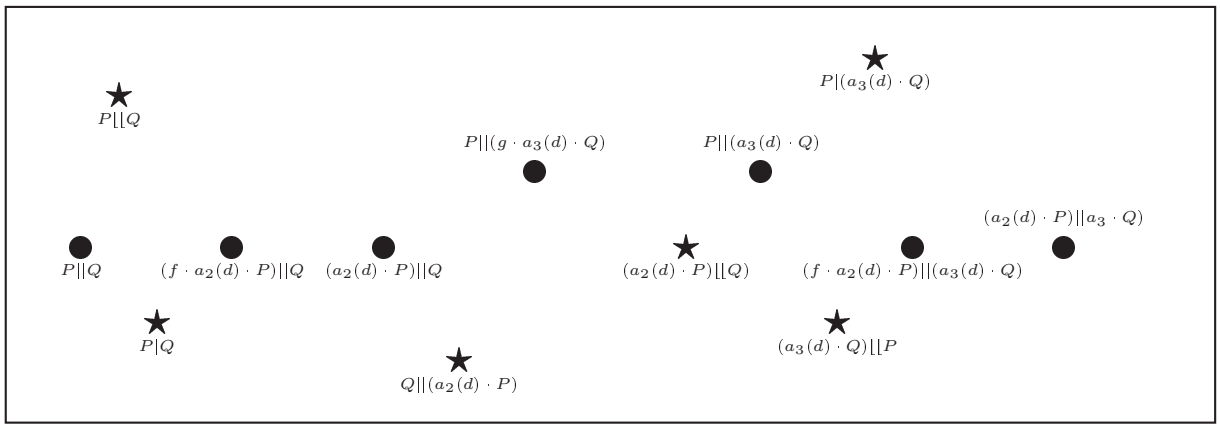

Figura 5.8.: Posibles estados para la definición 5.1

En la figura 5.8, los estados por los que pasa el sistema en una ejecución exitosa están representados por el símbolo • y los estados marcados con ⋆ son aquellos que llevan al sistema a deadlock. Es posible enriquecer el esquema colocando una flecha dirigida que indique el estado inicial y el estado al que evolucionó etiquetando cada flecha para indicar cual fue la acción que se ejecutó, dando como resultado la figura 5.9, la cual representa la evolución de la solución planteada a través de los estados válidos, eliminando los inválidos y etiquetándolos con la ecuación a la que corresponde en la derivación, iniciando con $R = P||Q$ y evolucionando a estados válidos siempre.

En la figura 5.9 se ha etiquetado un estado como $R_2\epsilon$ debido a que el sistema cambia de estado al ejecutar $c_2(d)$, sin embargo de acuerdo a la definición 5.1 $c_s(d) \in I$ por lo tanto, esta ejecución sería

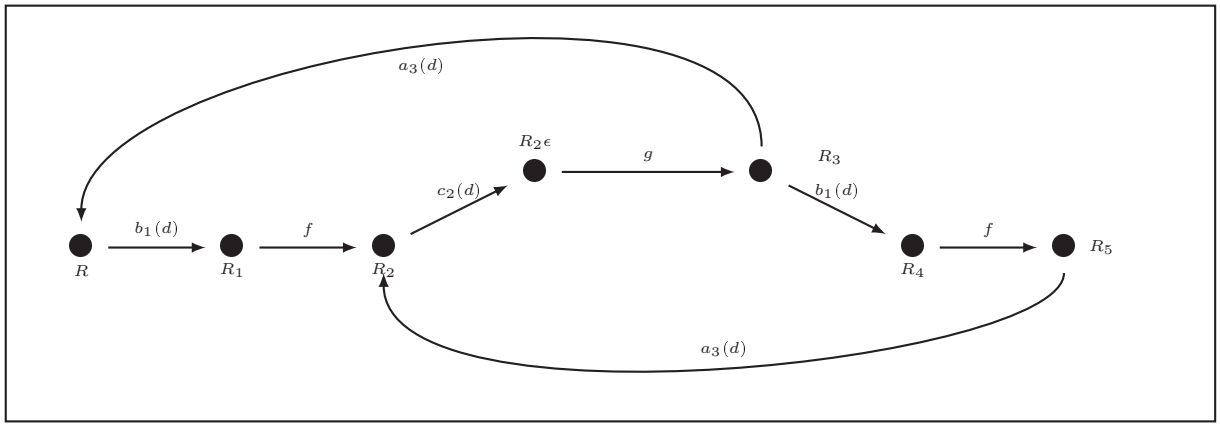

Figura 5.9.: Derivación para la definición 5.1

como una transición ϵ en un autómata, ya que así está definido el comportamiento de τ_I, aunque en la derivación se maneja como un paso intermedio para pasar de R_2 a R_3.

Tomando como base la figura 5.9 se puede obtener las especificaciones de algoritmos secuenciales para los procesos P y Q de la siguiente forma:

- Se identifican los estados R_i obtenidos del proceso de derivación.
- Se asocia cada estado con la acción $a \in R_i$ que al ejecutarse hizo evolucionar al sistema a dicho estado, indicando a que proceso pertenece dicha acción, en este caso P o Q. El resultado es una relación con el esquema $X \xrightarrow{a \in R_i} X'$ de los estados con las acciones ejecutadas.

La tabla 5.2 muestra el resultado de este pequeño proceso para el Problema 1, donde cada renglón es una relación con el esquema mencionado y la tabla completa representa el comportamiento que observará el sistema al ser ejecutado.

A partir del análisis del comportamiento del sistema también es posible determinar las acciones que realizó cada proceso por separado, es decir, obtener el algoritmo para cada proceso, ya que el método de derivación establece que se debe trabajar una acción a la vez, excepto en las comunicaciones y aún en ellas cada proceso debe tener una acción de envío o recepción definida, como se muestra en la tabla 5.3, donde la columna $No.$ corresponde a la fila donde se encontró la acción en

No.	Estado anterior	Acción	Proceso	Estado Actual
1	$P\|\|Q$	$r_R(d)$	P	$f \cdot a_2(d) \cdot P)\|\|Q$
2	$(f \cdot a_2(d) \cdot P)\|\|Q$	f	P	$(a_2(d) \cdot P)\|\|Q$
3	$(a_2(d) \cdot P)\|\|Q$	$a_2(d)*, b_2(d)*$	$P\|Q$	$P\|\|(g \cdot a_3(d) \cdot Q)$
4	$P\|\|(g \cdot a_3(d) \cdot Q)$	g	Q	$P\|\|(a_3(d) \cdot Q)$
5	$P\|\|(a_3(d) \cdot Q)$	$a_3(d)$	Q	$P\|\|Q$
6	$P\|\|(a_3(d) \cdot Q)$	$b_1(d)$	P	$(f \cdot a_2(d) \cdot P)\|\|(a_3(d) \cdot Q)$
7	$(f \cdot a_2(d) \cdot P)\|\|(a_3(d) \cdot Q)$	f	P	$(a_2(d) \cdot P)\|\|(a_3(d) \cdot Q)$
8	$(a_2(d) \cdot P)\|\|(a_3(d) \cdot Q)$	$a_3(d)$	Q	$(a_2(d) \cdot P)\|\|Q$

*Equivalen a $c_2(d)$

Tabla 5.2.: Relación entre estados y acciones para el problema 1

la tabla 5.2 y las acciones ejecutadas fueron colocadas debajo del proceso que las ejecutó.

La tabla 5.3 es la primera aproximación del algoritmo secuencial, pero si se analiza la derivación con cuidado se puede notar que los primeros pasos los realiza solo el primer proceso, reflejando un comportamiento secuencial, hasta que se da el evento de comunicación donde se involucran los dos procesos y a partir de ahí se pueden dar varios estados, pero las acciones de cada proceso suceden siempre secuencialmente mientras la derivación lleve a estados distintos, sin embargo, después de ejecutarse la acción g y también después de la acción f (por segunda vez) se repiten los estados, lo

No.	P
1	$b_1(d)$
2	f
3	$a_2(d)^*$
6	$b_1(d)$
7	f

No.	Q
3	$b_2(d)^*$
4	g
5	$a_3(d)$
8	$a_3(d)$

*Acciones simultáneas que equivalen a $c_2(d)$

Tabla 5.3.: Procesos P y Q para el problema 1

cual implica que el sistema debe entrar en un bucle.

Algoritmo P	Algoritmo Q
1: $b_1(d)$	1: **while true do**
2: f	2: $b_2(d)$
3: **while true do**	3: g
4: $a_2(d)$	4: $a_3(d)$
5: $b_1(d)$	5: **end while**
6: f	
7: **end while**	

Figura 5.10.: Algoritmo concurrente para el problema 1

Para lograr repetir las acciones que suceden después de la acción g del proceso Q se puede colocar un bucle que vaya al inicio de las acciones del proceso Q. En el caso del segundo ciclo se da por la misma acción pero al ejecutarse la acción f del proceso P, en cuyo caso basta colocar un ciclo hacia la acción de comunicación para representar el comportamiento. La especificación simbólica del algoritmo (con los elementos de A) se puede ver en la figura 5.10 y su versión mas expresiva en la figura 5.11.

Proceso P	proceso Q
1: Recibe el dato d binario por el canal 1	1: **while true do**
2: Invierte el dato d	2: Recibe el dato d binario por el canal 2
3: **while true do**	3: Concatena izquierda de N y separa derecha N
4: Envía el dato d por el canal 2	4: Envía el dato d por canal 3
5: Recibe el dato d binario por el canal 1	5: **end while**
6: Invierte el dato d	
7: **end while**	

Figura 5.11.: Algoritmo concurrente (expresivo) para el problema 1

5.3.2. Problema 2: Autómata celular unidimensional

A continuación se definirá otro problema para ser tratado con el método mostrado en el presente capítulo.

> **Problema 2:** `Autómata celular unidimensional`
> *Se requiere un sistema que pueda modelar un autómata celular unidimensional con condiciones de frontera periódicas y una función de transición local f para cuatro células.*

Análisis del problema

Los Autómatas celulares (AC) [Kar05] son sistemas discretos dinámicos y también modelos de computación masiva con componentes en paralelo que interactúan solo localmente con reglas que tienen una invariante con respecto a su espacio y tiempo, Su comportamiento se establece por medio de reglas de transición que realizan operaciones locales y cambian su estado dado en un tiempo t a otro distinto en el tiempo t + 1.

Un AC unidimensional se encuentra formado por células y se organizan en una retícula donde cada célula puede tomar valores de estado específicos, tiene una regla de transición local f que actualiza su estado con base en sus valores y los de sus sus vecinos.

El autómata cuenta con una regla de transición global que está definida por la aplicación de la

regla de transición local a todas las celdas en el mismo paso de tiempo.

El presente ejemplo es un autómata unidimensional de cuatro células, organizadas como una retícula donde los vecinos se encuentran conectados por medio de un canal de comunicación como se muestra en la figura 5.12.

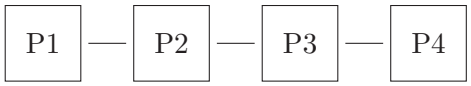

Figura 5.12.: Retícula problema 2

Para definir el comportamiento de un autómata celular es necesario contemplar lo que sucede en las fronteras de la retícula, ya que como se puede observar en la figura 5.12 las células A y D carecen de un vecino. En este caso, se definirá que el autómata tendrá una frontera periódica, es decir, que los elementos de los extremos estarán también conectados, como se muestra en la figura 5.13.

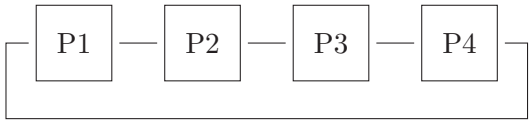

Figura 5.13.: Autómata problema 2

Por lo tanto, este sistema se puede ver como cuatro procesos que recibirán valores de sus vecinos a través de un canal, los cuales se etiquetarán con números como se muestra en la figura 5.14. Dichos canales son bidireccionales y todos los procesos ejecutarán la misma función de transición f, pero cumpliendo la función global de transición, es decir, cada proceso transmitirá y recibirá la información del momento actual t una vez y cuando todos calculen su función local el nuevo estado del sistema será el estado $t + 1$.

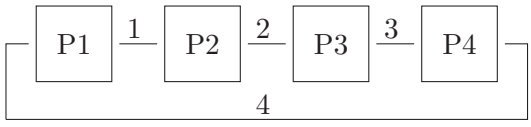

Figura 5.14.: Autómata problema 2 con canales etiquetados

En este problema solo se identifica como funcionalidad(Paso (1) Funcionalidad) la función f de transición local y los datos son dados por cada proceso, por lo tanto, el sistema es homogéneo.

También se puede observar que la funcionalidad no se puede dividir pero los datos se encuentran distribuidos entre los procesos, por lo que se puede clasificar al tipo de paralelismo (Paso (2) Paralelismo) como de dominio.

De acuerdo con la tabla 5.1 y la información obtenida, el patrón a utilizar es Comunicating Sequential Elements, cuya estructura aplicada al presente problema (Paso (3) Representación) se muestra en la figura 5.15.

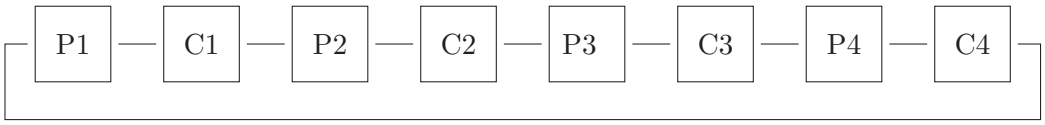

Figura 5.15.: Ejemplo 2 con Comunicating Sequential Elements

Planteamiento algebraico

Para el presente problema se identifican las siguientes acciones atómicas (Paso (4) Elementos):

- Función de transición local f. Esta se verá como una acción atómica y por lo tanto su duración es despreciable.
- Cada linea entre dos elementos de la figura 5.15 se modelará como un evento de comunicación. En total se definirán 8, como se puede ver en la figura 5.16.

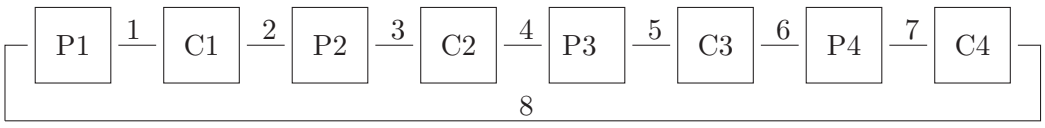

Figura 5.16.: Ejemplo 2 con comunicaciones etiquetadas

- Se definirá el dominio de datos del problema Δ, así como un conjunto H de acciones que al ejecutarse caerán en deadlock(Paso (4) Elementos).
- Finalmente se definirán los procesos P1, P2, P3 y P4, así como los canales de comunicación C1,

C2,C3 y C4. La representación algebraica se detalla en la definición 5.8(Paso (5) Ecuaciones).

Como en el ejemplo anterior, los términos de la forma $\Sigma_{d\in\Delta}a_i(d)$ o $\Sigma_{d\in\Delta}b_i(d)$ se simplificarán por $a_i(d)$ y $b_i(d)$ respectivamente (Paso 5).

$$
\begin{aligned}
f &\triangleq \text{Función de transición local}\\
\Delta &\triangleq \{d \mid d \quad \text{Puede ser transmitido por los procesos P1,P2,P3 y P4}\}\\
r_i(d) &\triangleq \text{Recibe el dato d por el canal i} \quad (d \in \Delta,\ i \in \{1,2,3,4,5,6,7,8\})\\
e_i(d) &\triangleq \text{Envía el dato d por el canal i} \quad (d \in \Delta,\ i \in \{1,2,3,4,5,6,7,8\})\\
c_i(d) &\triangleq \gamma(e_i(d), r_i(d)) \quad (d \in \Delta,\ i \in \{1,2,3,4,5,6,7,8\})\\
H &\triangleq \{r_i(d), e_i(d)\}\\
P1 &\triangleq (e_1(d)\cdot r_8(d)\cdot e_8(d)\cdot r_1(d) + e_8(d)\cdot r_1(d)\cdot e_1(d)\cdot r_8(d))\cdot f\cdot P1\\
C1 &\triangleq (r_1(d)\cdot e_2(d) + r_2(d)\cdot e_1(d))\cdot C1\\
P2 &\triangleq (e_3(d)\cdot r_2(d)\cdot e_2(d)\cdot r_3(d) + e_2(d)\cdot r_3(d)\cdot e_3(d)\cdot r_2(d))\cdot f\cdot P2\\
C2 &\triangleq (r_3(d)\cdot e_4(d) + r_4(d)\cdot e_3(d))\cdot C2\\
P3 &\triangleq (e_5(d)\cdot r_4(d)\cdot e_4(d)\cdot r_5(d) + e_4(d)\cdot r_5(d)\cdot e_5(d)\cdot r_4(d))\cdot f\cdot P3\\
C3 &\triangleq (r_5(d)\cdot e_6(d) + r_6(d)\cdot e_5(d))\cdot C3\\
P4 &\triangleq (e_7(d)\cdot r_6(d)\cdot e_6(d)\cdot r_7(d) + e_6(d)\cdot r_7(d)\cdot e_7(d)\cdot r_6(d))\cdot f\cdot P4\\
C4 &\triangleq (r_7(d)\cdot e_8(d) + r_8(d)\cdot e_7(d))\cdot C4\\
R &\triangleq \partial_H(P1||C1||P2||C2||P3||C3||P4||C4)
\end{aligned}
\tag{5.8}
$$

Una vez mas se pretende probar que $R \neq \delta$.

Derivación algebraica

Para la derivación de este problema (Paso 6) se agruparán los términos por pares para operarlos y para realizar esto se tomará en cuenta si los procesos comparten un canal, debido a que cada término comienza con un elemento de una acción de comunicación. La ecuación R se puede reescribir como

$$R = \partial_H((P1||C1)||(P2||C2)||(P3||C3)||(P4||C4)) \tag{5.9}$$

Una vez mas solo se colocarán los resultados del proceso de derivación y el detalle se puede ver en el apéndice C.3.

Al realizar la derivación, en los primeros pasos cada par arrojó una acción atómica de comunicación y fue transformando la ecuación inicial en un mas compleja.

Cuando se derivó $(P1||C1)$ se ejecutó $c_1(d)$ y el nuevo estado de sistema es 5.10.

$$\begin{aligned} R_1 =& \partial_H(b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_2(d) \cdot C1||(P2||C2) \\ & ||(P3||C3)||(P4||C4) \end{aligned} \tag{5.10}$$

Cuando se derivó $(P2||C2)$ se ejecutó $c_3(d)$ y el nuevo estado de sistema es 5.11.

$$\begin{aligned} R_2 =& \partial_H(b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_2(d) \cdot C1 \\ & ||b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_4(d) \cdot C2 \\ & ||(P3||C3)||(P4||C4)) \end{aligned} \tag{5.11}$$

Cuando se derivó $(P3||C3)$ se ejecutó $c_5(d)$ y el nuevo estado de sistema es 5.12.

$$\begin{aligned} R_3 =& \partial_H((b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_2(d) \cdot C1) \\ & ||b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_4(d) \cdot C2 \\ & ||b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_6(d) \cdot C3 \\ & ||(P4||C4)) \end{aligned} \tag{5.12}$$

Cuando se derivó $(P4||C4)$ se ejecutó $c_7(d)$ y el nuevo estado de sistema es 5.13.

$$\begin{aligned} R_4 =& \partial_H(b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_2(d) \cdot C1||b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_4(d) \cdot C2 \\ & ||b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_6(d) \cdot C3||b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_8(d) \cdot C4) \\ R_4 =& \partial_H((b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_8(d) \cdot C4)||(b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_2(d) \cdot C1) \\ & ||(b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_4(d) \cdot C2)||(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3)) \end{aligned} \tag{5.13}$$

Es importante notar que los términos fueron reagruparon por pares con el mismo criterio que la primera vez y se repitió el proceso. Las acciones ejecutadas después de cada derivación fueron $c_8(d) \cdot c_2(d) \cdot c_4(d) \cdot c_6(d)$ y el estado del sistema es reflejado en la ecuación 5.14.

$$\begin{aligned} R_8 = \partial_H(a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4||a_2(d) \cdot b_3(d) \cdot f \cdot P2||C1 \\ ||a_4(d) \cdot b_5(d) \cdot f \cdot P3||C2||a_6(d) \cdot b_7(d) \cdot f \cdot P4||C3) \end{aligned} \tag{5.14}$$

Repitiendo el mismo proceso, la derivación arroja las acciones $c_8(d) \cdot c_2(d) \cdot c_4(d) \cdot c_6(d)$ nuevamente y el nuevo estado del sistema es la ecuación 5.15.

$$\begin{aligned} R_{12} = \partial_H(b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4 \\ ||b_3(d) \cdot f \cdot P2||a_1(d) \cdot C1 \\ ||b_5(d) \cdot f \cdot P3||a_3(d) \cdot C2 \\ ||b_7(d) \cdot f \cdot P4||a_5(d) \cdot C3) \end{aligned} \tag{5.15}$$

Una vez mas se realiza el proceso, la derivación arroja las acciones $c_1(d) \cdot c_3(d) \cdot c_5(d) \cdot c_7(d)$ y el nuevo estado del sistema es 5.16.

$$R_{16} = \partial_H((f \cdot P1||C1)||(f \cdot P2||C2)||(f \cdot P3||C3)||(f \cdot P4||C4)) \tag{5.16}$$

Cuando se realiza nuevamente el proceso partiendo de 5.16 se ejecuta la acción f para cada proceso $(f \cdot f \cdot f \cdot f)$, y el nuevo estado del sistema es

$$R_{20} = \partial_H(P1||C1||P2||C2||P3||C3||P4||C4) = R \tag{5.17}$$

Que es la ecuación inicial, por lo que si se continuara el proceso de derivación se obtienen nuevamente las ecuaciones anteriores, por lo tanto, el sistema compuesto no cae en deadlock $(R \neq \delta)$ que es lo que se quería demostrar.■

En la figura 5.17 se muestra el proceso de derivación de forma gráfica como en el ejemplo anterior, donde los puntos son los estados del sistema en ejecución y las transiciones están etiquetadas con la acción atómica que se ejecutó para lograr cada transición de estado (Paso 7).

Tal como se problema anterior, se identifican los estados R_i por lo que pasó el sistema y se identifican la acción atómica que provocó la transición de un estado a otro. La tabla 5.4 muestra el

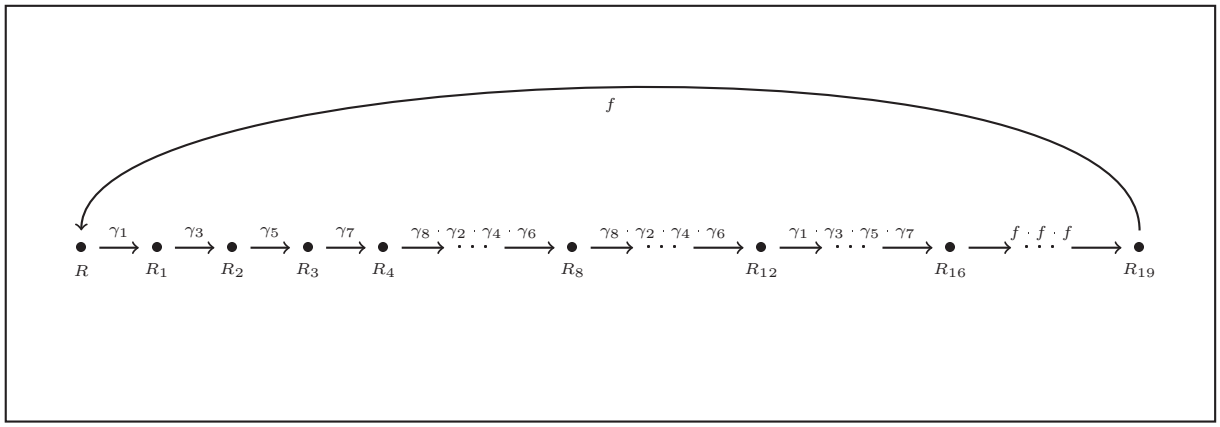

Figura 5.17.: Derivación para el problema 2

resultado de este proceso para el problema 2. Debido a que la mayoría de las acciones ejecutadas son de comunicación, es muy importante entender como se construyó la tabla.

Identificar la acción ejecutada es sencillo ya que cada paso de derivación me arroja esta información como traza, sin embargo, cuando se trata de una acción de comunicación se debe identificar que proceso ejecutó $a_i(d)$ y cual $b_i(d)$. Esto se puede saber consultando la definición de cada elemento, aunque en este caso es facil hacerlo debido a que el término siempre termina con su identificador, por ejemplo, el término $C2$, que se puede ver en la ecuación 5.18, termina con $C2$.

$$C2 \triangleq (r_3(d) \cdot e_4(d) + r_4(d) \cdot e_3(d)) \cdot C2 \tag{5.18}$$

Aplicando esta observación, en la derivación del apéndice 5.8 se puede ver el paso de la ecuación 5.19 que corresponde a la ejecución de la acción $c_6(d)$. Aquí se puede observar que el término cuyo primer elemento es $a_6(d)$ termina con $P4$, por lo tanto, es el proceso $P4$ el que lo ejecuta.

$$\begin{aligned}(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3) = \\ c_6(d) \cdot (b_7(d) \cdot f \cdot P4||a_5(d) \cdot C3)\end{aligned} \tag{5.19}$$

Leyendo la tabla 5.4 se pueden identificar las acciones que realiza cada proceso, como se observa en la tabla 5.5

No.	Estado anterior	Acción	Proceso	Estado Actual
1	R	$a_1(d), b_1(d)*$	$P1\|C1$	R_1
2	R_1	$a_3(d), b_3(d)*$	$P2\|C2$	R_2
3	R_2	$a_5(d), b_5(d)*$	$P3\|C3$	R_3
4	R_3	$a_7(d), b_7(d)*$	$P4\|C4$	R_4
5	R_4	$b_8(d), a_8(d)*$	$P1\|C4$	R_5
6	R_5	$b_2(d), a_2(d)*$	$P2\|C1$	R_6
7	R_6	$b_4(d), a_4(d)*$	$P3\|C2$	R_7
8	R_7	$b_6(d), a_6(d)*$	$P4\|C3$	R_8
9	R_8	$a_8(d), b_8(d)*$	$P1\|C4$	R_9
10	R_9	$a_2(d), b_2(d)*$	$P2\|C1$	R_{10}
11	R_{10}	$a_4(d), b_4(d)*$	$P3\|C2$	R_{11}
12	R_{11}	$a_6(d), b_6(d)*$	$P4\|C3$	R_{12}
13	R_{12}	$b_1(d), a_1(d)*$	$P1\|C1$	R_{13}
14	R_{13}	$b_3(d), a_3(d)*$	$P2\|C2$	R_{14}
15	R_{14}	$b_5(d), a_5(d)*$	$P3\|C3$	R_{15}
16	R_{15}	$b_7(d), a_7(d)*$	$P4\|C4$	R_{16}
17	R_{16}	f	$C1$	R_{17}
18	R_{17}	f	$C2$	R_{18}
19	R_{18}	f	$C3$	R_{19}
20	R_{19}	f	$C4$	R

*Equivalen a $c_i(d)$

Tabla 5.4.: Relación entre estados y acciones para el problema 2

No.	**P1**
1	$a_1(d)$
2	$b_8(d)$
3	$a_8(d)$
4	$b_1(d)$
5	f

No.	**P2**
1	$a_3(d)$
2	$b_2(d)$
3	$a_2(d)$
4	$b_3(d)$
5	f

No.	**P3**
1	$a_5(d)$
2	$b_4(d)$
3	$a_4(d)$
4	$b_5(d)$
5	f

No.	**P4**
1	$a_7(d)$
2	$b_6(d)$
3	$a_6(d)$
4	$b_7(d)$
5	f

No.	**C1**
1	$b_1(d)$
2	$a_2(d)$
3	$b_2(d)$
4	$a_1(d)$

No.	**C2**
1	$b_3(d)$
2	$a_4(d)$
3	$b_4(d)$
4	$a_3(d)$

No.	**C3**
1	$b_5(d)$
2	$a_6(d)$
3	$b_6(d)$
4	$a_5(d)$

No.	**C4**
1	$b_7(d)$
2	$a_8(d)$
3	$b_8(d)$
4	$a_7(d)$

Tabla 5.5.: Procesos P1,P2,P3, P4, C1, C2, C3 y C4 para el problema 2

En este problema la ejecución es lineal, es decir, sucede una acción tras otra y una vez terminadas todas el sistema vuelve al estado inicial. Para hacer esto se colocará un ciclo en cada proceso y el resultado final se muestra en la figura 5.18.

Algoritmo P1	Algoritmo P2	Algoritmo P3	Algoritmo P4
1: **while true do**	1: **while true do**	1: **while true do**	1: **while true do**
2: $a_1(d)$	2: $a_3(d)$	2: $a_5(d)$	2: $a_7(d)$
3: $b_8(d)$	3: $b_2(d)$	3: $b_4(d)$	3: $b_6(d)$
4: $a_8(d)$	4: $a_2(d)$	4: $a_4(d)$	4: $a_6(d)$
5: $b_1(d)$	5: $b_3(d)$	5: $b_5(d)$	5: $b_7(d)$
6: f	6: f	6: f	6: f
7: **end while**	7: **end while**	7: **end while**	7: **end while**

Algoritmo C1	Algoritmo C2	Algoritmo C3	Algoritmo C4
1: **while true do**	1: **while true do**	1: **while true do**	1: **while true do**
2: $b_1(d)$	2: $b_3(d)$	2: $b_5(d)$	2: $b_7(d)$
3: $a_2(d)$	3: $a_4(d)$	3: $a_6(d)$	3: $a_8(d)$
4: $b_2(d)$	4: $b_4(d)$	4: $b_6(d)$	4: $b_8(d)$
5: $a_1(d)$	5: $a_3(d)$	5: $a_5(d)$	5: $a_7(d)$
6: **end while**	6: **end while**	6: **end while**	6: **end while**

Figura 5.18.: Algoritmo concurrente para el problema 2

5.4. Resumen

El enfoque presentado en este capítulo para atacar problemas con concurrencia comienza con la representación de la solución a nivel conceptual y gráfico. Para lograrlo, se ha recurrido a los *Patrones para diseño de software en paralelo*, los cuales permiten un análisis minucioso de los sistemas y facilitan la tarea de elegir la organización de procesos concurrentes adecuada al problema.

Para lograr una representación algebraica de la solución conceptual generada se utilizan los elementos sintácticos del álgebra de procesos, entre ellos la definición del conjunto de acciones

atómicas A, la caracterización de las comunicaciones y la construcción de los procesos.

Una vez definida la solución algebraica se procede a verificar la validez de la solución por medio de derivaciones algebraicas, las cuales simulan la ejecución del sistema de forma secuencial pero reflejando la concurrencia en las ecuaciones. Este proceso termina cuando se llega al predicado $a \xrightarrow{a} \surd$ con $a \in A$, denotando que el sistema termina o cuando después de cierto número de derivaciones se repite algún estado, lo cual refleja el comportamiento de un sistema que se ejecutará indefinidamente.

La derivación algebraica permite saber el estado en el que se encuentra el sistema y es la basepara obtener un algoritmo concurrente, ya que el producto de este proceso es un conjunto de ecuaciones que representan al sistema en diferentes estados y una traza o ruta que indica el camino que sigue la ejecución, así como las acciones que fueron haciendo evolucionar al sistema de un estado al otro. Si se sigue dicha traza y se separan las acciones colocándoles en el orden que fueron ejecutadas se obtendrá una descripción algorítmica del sistema, cuya notación es la que se establece cuando se define el conjunto A.

En el presente capítulo se realizó una definición de algoritmo concurrente, la cual describe brevemente la naturaleza de la representación esperada y las características que debe tener, como se ve en las soluciones de los problemas, representadas respectivamente por las figuras 5.10 y 5.18.

En el siguiente capítulo se explicará porque está garantizada la ejecución exitosa de una solución representada con ACP y porque es posible obtener el algoritmo a partir del proceso de verificación.

6. Análisis del proceso de obtención del algoritmo concurrente

En el presente capítulo se mostrará la forma en que los conceptos mas importantes del método mostrado en el capítulo anterior sustentan el supuesto de que es posible obtener un algoritmo concurrente a partir del proceso de verificación algebraica.

Como se muestra en la figura 5.2 exienten 3 grandes etapas que componen el método, lo cuál es lógico ya que cada una corresponde a una representación distinta del problema de acuerdo con el planteamiento de Pancake y Bergmark: se comienza con un esquema que aporta la estructura de los procesos; a continuación se realiza un mapeo para convertir dicha representación en definiciones algebraicas y por último se operan las expresiones y mediante un proceso de análisis es posible obtener una representación algorítmica,

6.1. Análisis del problema

Cuando se tiene un problema por resolver normalmente se plantea en forma de enunciados o algún tipo de narrativa en lenguaje cotidiano. El primer paso para su solución es realizar una representación mas abstracta pero que muestre que se tiene un mayor entendimiento del mismo.

Los Patrones para diseño de software en paralelo permiten esquematizar un problema concurrente utilizando criterios bien definidos, aportando elementos que sirven de guía para construir una solución al problema.

Como en todo proceso de construcción de software, el primer paso es detectar las funciones que

debe realizar un sistema y en este tipo de sistemas, determinar el dominio de datos y la forma en que serán divididos. Con esta información ya es posible generar un esquema de alto nivel que modele la solución del problema eligiendo uno de los patrones arquitectónicos utilizando la tabla 5.1, evitando un proceso de ensayo y error esquematizar la solución del problema planteado.

Sin embargo, una de las mayores aportaciones de los patrones de software al método es que el tipo de esquema tiene una estructura definida, por lo que es posible realizar un mapeo hacia su representación algebraica.

Otra característica importante de la solución planteada con un patrón de software es que facilita la derivación algebraica debido a que están pensados en facilitar el proceso de comunicación y evitar los deadlocks.

Por ejemplo, en el análisis del problema 2 se llegó a la conclusión de que el patrón a utilizar fue *Comunicating Sequential Elements*, cuya estructura incluye no solo los procesos con su función de transición, sino también la adición de otros elementos llamados canales de comunicación que son buffers con la función de guardar un valor, pero también son un tipo de estructura "espejo' que provocan que en cada proceso de comunicación siempre exista un número par de ellos y así lograr que todos puedan comunicarse simultáneamente, evitando que alguno de los procesos caiga en inanición o que el sistema caiga en deadlock.

6.2. Planteamiento algebraico

La representación algebraica se obtiene a partir del análisis realizado para obtener el esquema de representación de la solución y de los elementos del esquema. El dominio de datos es el primer elemento a determinar y normalmente ya se ha identificado(conjunto Δ), así como las acciones que debe realizar cada proceso.

La cantidad de procesos, así como sus comunicaciones están determinadas por el esquema generado previamente.

Cada proceso se transforma en un término de álgebra de procesos y se define de manera recursiva, tomando en cuenta que también deben ser custodiadas, es decir, siempre existe al inicio una acción

atómica y los términos incluidos del lado derecho de una definición deben encontrarse también del lado izquierdo aunque se encuentren en otra ecuación. Esto permite definiciones de términos basados en acciones atómicas que tendrán el atributo de contar con un número de estados finitos, por la naturaleza del álgebra de procesos y del propio concepto de custodia.

Una vez definidas las acciones y los procesos que las ejecutarán se define el término principal que será el estado inicial del sistema, el cual tiene un número finito de estados debido a que lo hereda de los términos que lo componen por lo tanto el proceso de derivación siempre será finito.

Uno de los primero retos en la definición de un término es tomar en cuenta los posibles escenarios bajo los cuales el término debe funcionar. Por ejemplo, es posible definir el proceso $R \triangleq f \cdot g \cdot h$, sin embargo puede suceder que el término puede también definirse como $R \triangleq h \cdot f \cdot g$, en cuyo caso se utiliza la definición $R \triangleq f \cdot g \cdot h + h \cdot f \cdot g$.

6.3. Verificación algebraica

Como se ha dicho a lo largo del presente trabajo, el propósito primario de realizar representaciones algebraicas es verificarlas, ya que esta representación permite modelar sistemas en una forma sencilla y analizar su comportamiento.

Una parte muy importante del enfoque es la forma en que se define la axiomatización del apéndice B, ya que cada extensión del álgebra detallada en el apéndice A obedece a funcionalidades específicas que deben reflejarse en la definición y verificación de las soluciones representadas.

La primera extensión de BPA se llama PAP y contiene la suficiente expresividad para detallar el comportamiento concurrente paso a paso por acción, así como el mecanismo de comunicación entre procesos, es decir, gracias a esta extensión es posible expresar la concurrencia y modelar su comportamiento como si fuera una prueba de escritorio secuencial pero sin perder los detalles de las acciones ejecutadas.

La segunda extensión permite garantizar la concurrencia exitosa, es decir, evitar que los sistemas caigan en deadlock. Por ejemplo, la regla $x + \delta = x$ indica que el sistema ha llegado a una ejecución no determinista de dos acciones pero una de ellas dejará al sistema "congelado", por lo tanto,

siempre se elige ir por la acción válida, lo cual significa que de dos estados posibles uno se evitará y solo se seguirá el flujo del estado x.

Otro caso muy común es que se quiera realizar una acción de transmisión o recepción de datos sin su complemento, lo cual también es llevado a deadlock por parte del operador ∂ en el axioma A19 y un tercer caso recurrente es cuando se pretende realizar una comunicación $a|b$ pero $c = \gamma(a, b)$ no forma parte de la definición del conjunto A.

La tercera extensión es la que permite modelar comportamiento infinito, aportando una restricción muy importante al modelo completo que consiste en expresar las representaciones de los sistemas como ecuaciones custodiadas.

Finalmente la implementación del silent step y el operador τ_I le dan nuevamente una mayor expresividad al modelo ya que permiten el uso de acciones intermedias que quizá no cambien el comportamiento del sistema completo pero su especificación puede ser muy útil cuando se implemente algún sistema definido bajo este enfoque.

La derivación algebraica es un proceso unidireccional e incremental ya que siempre se toman en cuenta los términos de izquierda a derecha. Al realizar un paso de derivación dicho elemento desaparece, como si se hubiera ejecutado.

Cuando se aplica un axioma en un término siempre afecta a los elementos que se encuentran a la izquierda, siempre tomando en cuenta la jerarquía de las operaciones. Por ejemplo, si se tiene el término $z + z + y + y$ y se pretende aplicar el axioma A03 (apéndice B), existen dos posibilidades: que se pueda aplicar sobre $z + z$ o sobre $y + y$, sin embargo el proceso indica que se debe trabajar con $z + z$ y entonces el resultado sería $z + z + y + y = z + y + y$. En el caso de las operaciones relacionadas con comunicaciones sucede lo mismo, salvo que, debido su la naturaleza ,se debe tomar el elemento de la izquierda de cada proceso simultáneamente, por ejemplo, el término $a \cdot b|c \cdot d$ está compuesto a su vez por los términos $a \cdot b$ y $c \cdot d$, por lo que al aplicar el axioma A08 se tiene que $a \cdot b|c \cdot d = \gamma(a, c) \cdot (b||d)$, si $\gamma(a, c)$ está definida.

Cada paso de derivación refleja un cambio en el sistema y cada ecuación resultante refleja un nuevo estado en el mismo, estableciendo una relación entre estos elementos.

Un paso de derivación implica asumir la ejecución de una acción perteneciente a un término y la

forma en que se representa es eliminando la acción de la izquierda en el siguiente paso, por ejemplo, para el término $x \cdot y$ el siguiente paso de derivación sería y. Esta dinámica garantiza que en cada paso la ecuación se vuelve mas simple o por lo menos que se ha encontrado un nuevo estado del sistema y por lo tanto, que está mas cerca la solución.

En el caso de los procesos finitos el proceso comienza con una representación del sistema que consta de términos procesos o simples y termina cuando la última acción se ejecuta correctamente ($a \xrightarrow{a} \surd$).

Los procesos que no son finitos tienen una representación similar a los finitos pero contienen también variables recursivas, por lo que no se espera que el sistema llegue a un estado $a \xrightarrow{a} \surd$. En este caso, el proceso de verificación está enfocado en encontrar un número finito de ecuaciones que representen los estados del sistema y que en algún punto los estados se repitan, debido a que el sistema parte de ecuaciones custodiadas como se menciona en el apéndice A, garantizando que cada proceso definido en conjunto de ecuaciones recursivas es regular [BK89].

6.3.1. Comportamiento del sistema

Cuando se realiza la definición del problema en términos algebraicos, existen dos tipos de acciones atómicas: aquellas que realizan cálculos para cumplir las reglas de negocio del sistema y las acciones para comunicar y sincronizar los procesos involucrados. Es claro que las primeras suceden bajo el dominio exclusivo de alguno de los procesos, es decir, son locales, mientras las segundas si tienen la dependencia de dos procesos que requieren establecer una comunicación y justamente estas son las que pueden llevar al sistema a comportamientos no deseados, por lo tanto es vital su correcta manipulación cuando se derivan términos algebraicos.

Cuando se tienen acciones de comunicación el proceso de derivación se vuelve complejo en cuanto a seguir un flujo de ejecución, debido a que en realidad siempre se está trabajando con un solo término y una ejecución alternativa indica que existe mas de una forma de cambiar de estado. Por ejemplo en la ecuación $a_i \cdot (p' || q) + b_i \cdot (q' || p)$ existen 3 estados posibles:

1. Solo se ejecuta la acción a_i. Puede suceder que la acción b_i se encuentre en espera de algún dato o simplemente solo a_i logró tener recursos de procesamiento. Este caso el sistema pasa

al estado $(p'||q) + b_i \cdot (q'||p)$.

2. Solo se ejecuta la acción b_i. Como en el caso anterior, puede ser que solo se ejecute en un momento en el tiempo la acción b_i, en cuyo caso el sistema pasa al estado $a_i \cdot (p'||q) + (q'||p)$.

3. Ambas acciones atómicas se ejecutan. Aunque no se dice explícitamente en la ecuación, cuando se tiene un caso de ejecuciones alternativas (+) significa que cada término sigue un camino de derivación distinto, por ejemplo puede suceder que después de ejecutarse la acción a_i se ejecuten dos o mas acciones antes de b_i, dando pie a otros estados del sistema. Sin embargo, puede existir un estado del sistema donde ambas acciones se ejecutan al mismo tiempo, dicho estado sería $(p'||q) + (q'||p)$.

Este análisis es interesante porque hace notar que los estados de cada paso de derivación no siguen un flujo de ejecución (el cual no se puede saber en sistemas concurrentes debido al no determinismo), sino una búsqueda de estados posibles y su relación con las acciones ejecutadas.

El proceso de derivación para el problema 1 arrojó varias ecuaciones que representan estados posibles del sistema encontrar las ecuaciones de la figura 5.9, las cuales representan el conjunto (finito) de los estados que conforman el comportamiento del sistema, donde cada estado siempre evoluciona a otro válido, debido a que el sistema es regular por definición.

6.3.2. Obteniendo el algoritmo

Al observar el proceso completo de derivación y realizar el esquema de la figura 5.9 fue posible determinar la forma en que puede obtener la representación algorítmica a partir de la algebraica.

Sin embargo, no es fácil entender como sucede la concurrencia en un esquema estático sin detenerse a ver mas a detalle el proceso de verificación y particularmente el significado de cada paso de verificación con respecto al proceso completo.

Por otro lado, una derivación puede ser un proceso muy largo ante determinadas verificaciones, sin embargo, la información que arroja sirve para determinar la representación algorítmica que en la mayoría de los casos no es única, como se explicará un poco mas adelante.

Manejo de la concurrencia en ACP

El proceso de derivación algebraica permite visualizar lo que pasa cuando se ejecuta cada acción atómica definida y a que estado evoluciona el sistema, arrojando como resultado principal todos los estados posibles en los que el sistema puede caer.

En ACP el proceso de derivación permite mostrar la concurrencia de forma secuencial, es decir, aunque se trabaje con una sola acción a la vez, cuando se deriva es posible determinar que pasa cuando se ejecutan acciones simultáneas debido a que se contempla la evolución del sistema cuando sucede cada acción, siempre y cuando las ejecuciones no lleven al sistema a deadlock. Por ejemplo, el término $a \cdot X||b \cdot Y$ puede seguir un camino de derivación distinto cuando se ejecuta a que cuando se ejecuta b, inclusive se contempla la opción de que ambas acciones sean parte de un proceso de comunicación, por lo que el sistema puede llegar a un estado $X||Y$ donde a y b han sido ejecutadas, aunque en la derivación se pudieron haber realizado varios pasos. Por ejemplo, al observar nuevamente la tabla 5.2 del ejemplo resuelto en el capítulo anterior, cuando el sistema se encuentra en el estado 7 $((a \cdot e_C(d) \cdot P)||(e_C(d) \cdot Q))$ y al mismo tiempo se ejecutan las acciones a y $e_C(d)$, el sistema evoluciona directamente al estado 3 $((e_C(d) \cdot P)||Q)$.

Cuando se realiza una ejecución en paralelo, parece que las combinaciones de acciones en el tiempo dan origen a estados no contemplados pero esto no es así, lo que sucede es que el sistema pasa de un estado a otro pero no en el orden en que se realizó la derivación, debido a que se ejecuta mas de una acción simultáneamente.

El comportamiento de la comunicación

Existen dos detalles importantes a contemplar en el ejemplo planteado en el capítulo anterior, el primero tiene que ver con el momento en el que Q debe ejecutarse por primera vez para lograr la sincronización con P y el segundo con la forma de implementar el mecanismo de comunicación, es decir, el mapeo de $s_C(d) \triangleq \gamma(e_C(d), r_C(d))$ a código, aunque el alcance del presente trabajo no contepla la implementación del algoritmo en algún lenguaje de programación, es importante definir su funcionamiento ya que de eso depende que la solución planteada se comporte como se espera.

El primer detalle mencionado puede verse como un problema de implementación, sin embargo, debido a la forma en que se definen los algoritmos concurrentes en este trabajo, se debe establecer que se deben tener dos programas independientes listos para arrancar su ejecución aunque el orden en que lo hagan no debería importar mientras se garantice que ambos coexistirán.

La comunicación ente procesos se puede definir como síncrona y cada vez que se encuentre una tarea de recepción se debe esperar al otro proceso para que transmita. Aunque ya sea ha dicho que la comunicación está garantizada, esto solo tiene que ver con que el canal de comunicación siempre estará disponible si las operaciones de comunicación suceden de la forma mencionada.

Esta definición de comunicación se puede ver en la solución del problema 2 cuando se establece el ciclo para el algoritmo del proceso Q, ya que dicho bucle debe cubrir dos casos, el primero es cuando se ejecuta la acción b y luego la acción $e_E(d)$ porque debe satisfacer que el sistema se encuentre como en el estado inicial, es decir, no debe encontrarse ningún dato por enviar por parte del proceso Q (lo cual se cumple) y que el sistema debe encontrarse listo para recibir un dato por medio del proceso P. El otro caso es cuando se ejecuta la acción a y luego $e_E(d)$, en cuyo caso se necesita que se suceda el proceso de comunicación entre los procesos. Ambos casos se pueden cubrir estableciendo la comunicación síncrona y la dinámica de espera antes mencionada, ya que para el primer caso la acción $r_C(d)$ tendrá que esperar que se ejecuten las acciones $r_C(d)$ y a antes de sincronizar la comunicación. Para el segundo caso el sistema pasa inmediatamente del estado 8 de la tabla 5.2 al estado 3.

Soluciones equivalentes

Una vez realizadas las derivaciones correspondientes para un problema representado algebráicamente, lo que se obtiene son los estados por los que el sistema debe pasar para tener una ejecución exitosa. La figura 6.1 pretende representar este comportamiento tomando como base a todos los puntos del esquema como los posibles estados por los que puede pasar un sistema representado por términos algebraicos, los estados marcados con $\star$ representan los estados en los que el sistema no debe caer y los estados marcados con $\bullet$ dentro de la línea punteada representan los estados válidos que fueron el resultado de las derivaciones de la representación inicial de su solución, la cual se

compone de los estados:

$$A \rightarrow F \rightarrow I \rightarrow L \rightarrow P \rightarrow U \tag{6.1}$$

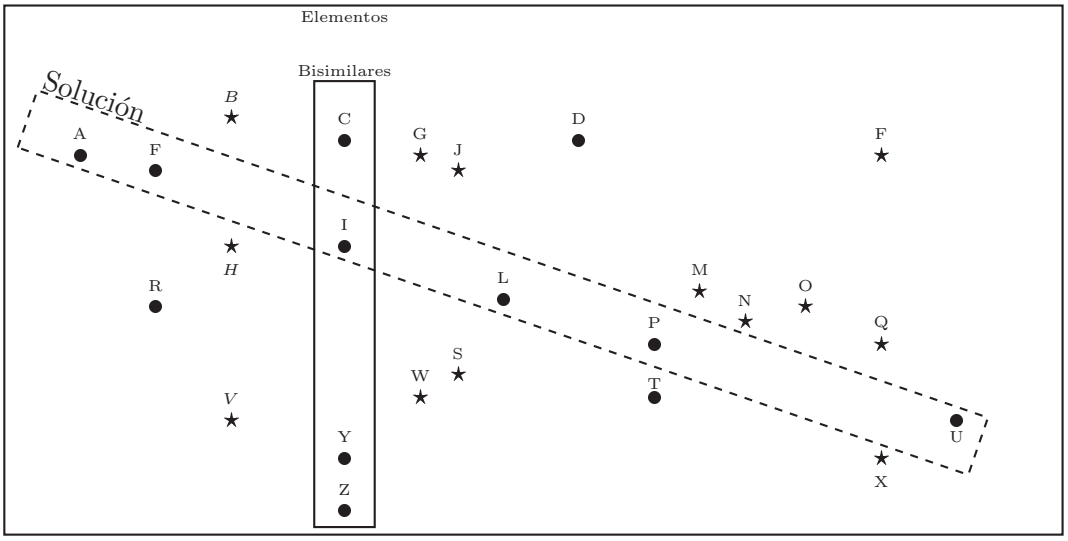

Figura 6.1.: Posibles estados de un sistema

Bajo el enfoque de ACP es posible contar con dos soluciones para un problema dado y que sus términos sean distintos, pero ambas soluciones deberán equivalentes no solo desde el tipo de elementos procesados, sino de la forma en que lo hacen. La figura 6.1 también representa este comportamiento colocando a los elementos Bisimilares alineados verticalmente, como se muestra en el recuadro de color negro que encierra a los estados C,I,Y y Z. Cada estado del sistema es una ecuación o una derivación de ella, por lo tanto , también tiene sentido hablar de estados Bisimilares $C \underline{\leftrightarrow} Y$.

La secuencia 6.1 enuncia los estados por los cuales pasará el sistema que resuelva el problema con la representación algebraica R, y puede ser que se tenga otra representación S que también sea solución, sin embargo, esta última puede no pasar por los mismos estados, pero si por estados equivalentes, como sucede con la secuencia

$$A \rightarrow R \rightarrow Z \rightarrow L \rightarrow T \rightarrow U \tag{6.2}$$

Donde $A \underline{\leftrightarrow} A$, $R \underline{\leftrightarrow} F$, $Z \underline{\leftrightarrow} I$, $\underline{\leftrightarrow} L$, $T \underline{\leftrightarrow} P$ y $U \underline{\leftrightarrow} U$, por lo tanto, se puede decir que $R \underline{\leftrightarrow} S$.

6.4. Resumen

El procedimiento para obtener un algoritmo concurrente a partir de una representación gráfica de una solución ha sido definido y ejecutado, sin embargo es importante preguntarse como los pasos realizados ayudan a obtener el resultado esperado.

La representación por medio de patrones de software aporta certeza en cuanto a la relación entre procesos y sus comunicaciones, así como la posibilidad de definir un mapeo del esquema a los elementos que conforman la definición de un sistema en el álgebra de procesos.

el sistema modelado tendrá un número finito de estados, es decir, la derivación siempre terminará sin importar si los procesos son finitos o no.

El uso de ACP para la derivación de una solución garantiza que las ejecuciones no caigan en deadlock y que sea posible especificar comportamientos internos sin complicar demasiado su representación.

La naturaleza operacional del álgebra de procesos hace posible el proceso de verificación debido a que cada paso de derivación en una representación algebraica significa que se ejecuta una acción provocando cambios en el estado del sistema, es decir, se genera una nueva ecuación.

Además de las restricciones algebraicas analizadas, el proceso de verificación ayuda en la determinación del algoritmo, ya que cuando se realizan las derivaciones necesarias para verificar una propiedad, como la ausencia de deadlock, el resultado es una representación secuencial de un sistema concurrente, lo cual facilita la obtención de las acciones que cada proceso realiza analizando su traza.

La definición recursiva custodiada de los términos y la forma en que se realiza la derivación permiten garantizar que el proceso de verificación sea finito, ya sea porque al ejecutarse cada paso se elimina al menos una acción del sistema, evolucionando hasta que termina su ejecución ($R = \surd$)

o porque se repiten estados anteriores, debido a la naturaleza regular de las soluciones. Gracias a esta característica, siempre es posible obtener una representación algorítmica de la solución.

Finalmente, un proceso de derivación puede llevar a la obtención de varias soluciones debido a la naturaleza de ACP, en particular gracias al concepto de bisimulación que permite no solo definir la axiomatización del álgebra de procesos, sino establecer relaciones de igualdad entre términos, lo cual permite construir mas de una representación algorítmica de una solución.

7. Conclusiones y trabajo futuro

A continuación se presentarán los resultados obtenidos del proceso de elaboración del presente documento, la aplicación del enfoque presentado y la investigación realizada para lograr una representación algorítmica de una solución concurrente.

7.1. Sumario

Siempre se ha hablado del rápido avance de las Tecnologías de la Información y la Comunicación, sin embargo, es importante mencionar que esto es cierto en el terreno del hardware pero en el mundo del software el cambio no ha sido tan acelerado, como se puede ver en el terreno de la programación concurrente ya que desde hace tiempo existen equipos que permiten la ejecución de mas de un proceso a la vez pero aún no se ha podido explotar esta capacidad de procesamiento.

Uno de los retos mas importantes para la programación concurrente es cambiar la forma en que se codifica un programa de computadora, debido a que cambiar de paradigma no solo implica agregar funcionalidad a un lenguaje o al mismo sistema operativo, también requiere un cambio en la forma de ver los sistemas y como atacar los problemas que se pretenden paralelizar.

Aún antes del llamado "Free lunch is over" ya existían esfuerzos por parte de Hoare, Milner, Dijkstra y otros encaminados a representar sistemas sistemas físicos con un comportamiento concurrente, ya sea porque trabajaban en conjunto con otros sistemas o porque eran capaces de interactuar con seres humanos, asumiendo en este último caso que las personas eran otro elemento del sistema independiente pero complementario, dando origen a conceptos fundamentales que hoy son utilizados para el análisis de software concurrente.

Otro factor que ha influido en el desarrollo del área de sistemas concurrentes es la aplicación de la lógica al campo del software, como elemento principal en los enfoques actuales de verificación de sistemas tanto secuenciales como concurrentes, sin embargo al día de hoy todavía no es posible hablar de una metodología para el desarrollo de software concurrente, debido a que aún no se ha logrado la madurez en el análisis de soluciones que se tiene en el enfoque secuencial.

El álgebra utilizada en el desarrollo del presente trabajo se basa en aspectos lógicos y matemáticos que hacen que su definición sea completa y correcta, tomando en cuenta aspectos como la equivalencia entre dos estados del sistema o entre dos soluciones dadas. También cuenta con una axiomatización que permite aplicar reglas de igualdad directamente en la derivación de ecuaciones sin definir secuentes lógicos que permite la verificación de sistemas concurrentes.

Los enfoques actuales en el campo de la verificación de sistemas compuestos por procesos concurrentes han logrado generar soluciones verificables a problemas dados, buscando llevar la solución planteada hacia una implementación. Al plantear la representación de un sistema por medio un modelo de alto nivel y transformarlo directamente a un lenguaje de programación es necesario conservar a la solución correcta y completa y esto solo se puede lograr estableciendo restricciones que permitan mapear estructuras de alto nivel a un código que se comporte de una forma determinada y por lo tanto el lenguaje de programación debe ser elaborado pensando en ese modelo en particular.

El presente trabajo propone un método de análisis basado en la idea de que es útil la representación algorítmica de un sistema concurrente, ya que permite plantear soluciones que son independientes de un lenguaje de programación y facilita el paso de una descripción de alto nivel hacia la codificación de la solución debido a que realiza un mapeo hacia otra representación de la misma solución en términos mas abstractos pero una estructura similar a un lenguaje de programación y permite dividir la solución piezas de código secuencial.

Una representación algorítmica de un sistema también permite realizar análisis que pueden ser muy parecidos a la ejecución de un programa, inclusive es posible detectar posibles problemas de implementación que son visibles en alguna otra representación, concretamente en el caso de las comunicaciones.

Aunque aquí los términos "paralelo" y "concurrente" han sido utilizados de forma indistinta, se

tiene siempre en mente el hecho de que las soluciones planteadas son concurrentes pero pueden volverse paralelas si se tiene el hardware adecuado, que en este enfoque son equipos multicore con memorias independientes y un reloj global, y aunque las especificaciones mencionadas son muy específicas, han servido para generar un modelo que permite realizar un análisis de procesos sin tomar en cuenta aspectos que puedan aumentar la complejidad del problema.

7.2. Solución al problema planteado

En el capítulo 4 fue planteado el problema a resolver, que básicamente es la generación de una descripción algorítmica de un sistema compuesto por procesos concurrentes que resuelve un problema paralelizable, atacando el problema general del proceso de programación concurrente.

Después del análisis realizado al proceso de construcción de software concurrente planteado en el mismo capítulo es posible ver que, una vez planteada la solución a un problema por medio de procesos concurrentes, es necesario llevarla a distintos niveles de abstracción ya que de entrada dicha solución se encuentra bosquejada o inclusive bien definidia en forma gráfica o matemática.

El siguiente nivel de abstracción justamente es el algorítmico, aunque en este punto hay algunas prácticas comunes que dificultan el proceso, ya que como menciona Pancake y Bergmark en [PB90], al realizar la representación algorítmica de la solución normalmente se recurre a los algoritmos secuenciales y solo se estipulan algunas secciones que se pueden paralelizar, es decir, no hay una definición algorítmica a nivel de procesos secuenciales comunicándose e interactuando y por lo tanto, no es posible un análisis de eficiencia debido a que no existe un mapeo formal entre la representación algorítmica y la implementación en software, lo cual provoca que los desarrollos concurrentes logren su eficiencia por medio de un proceso de prueba y error.

La solución propuesta comienza con una definición de alto nivel que puede ser llevada fácilmente a una representación algebráica debido a que existe una relación directa entre los componentes de ambas abstracciones.

El modelo algebráico permite verificar propiedades de la solución, evita ejecuciones no deseadas y finalmente proporciona una descripción algorítmica de los procesos que forman un sistema con-

currente, tomando en cuenta su interacción e incluyendo en su código los eventos de comunicación, es decir, se tiene un algoritmo concurrente que puede ser sujeto a análisis posteriores y cuya implementación implicará definir los mecanismos de comunicación a nivel del lenguaje de programación, pero los procesos serán mapeados como programas secuenciales de forma directa.

Otra característica importante de la representación algorítmica obtenida es que está construida a partir de un proceso que siempre busca evitar ejecuciones que caigan en deadlock, sin embargo, pueden existir varias ejecuciones exitosas del programa que no pasan por los mismos estados y ser válidas, es decir, son equivalentes.

7.3. Avances y retos

Gracias al desarrollo de los diferentes enfoques para atacar problemas por medio de procesos concurrentes ha sido posible definir el enfoque del presente trabajo, ya sea porque se utilizan algunos de los conceptos como base o porque aportan ideas que fueron utilizadas para mejorar el método.

Uno de los factores que pueden determinar la eficiencia de un programa compuesto de procesos concurrentes es el número de procesadores disponibles en la arquitectura donde se ejecutará la solución creada, por lo tanto, las soluciones que sean implementadas en un lenguaje de programación deberán contemplar la forma en que el programa sea dividido en función de este factor y también la asignación del trabajo a los procesadores, lo cual implica también retos importantes.

Aunque el propósito de este trabajo es plantear una solución algorítmica ante un problema dado, dicha solución no es única pero cualquier otra será equivalente (o congruente módulo Bisimulación equivalente) a la primera, inclusive puede que una sea mas eficiente que otra, sin embargo, ese análisis escapa a los propósitos de este trabajo, pero el enfoque desarrollado en este documento puede servir como base para dar certeza al proceso de programación concurrentes, es decir, que las soluciones concurrentes generadas sean correctas y su ejecución sea mas rápida que su equivalente secuencial.

Un reto inherente al proceso utilizado para obtener un algoritmo concurrente a partir de una

solución dada es la simplificación del proceso de derivación, ya que la axiomatización desglosa todos los posibles comportamientos de un término aunque es posible ver el resultado de ciertos pasos antes de ejecutar la derivación, sin embargo, para validar estas observaciones es necesario realizar las demostraciones correspondientes para obtener de manera formal un nuevo axioma que simplifique la manipulación de una solución algebraica. Por ejemplo, si se tiene la definición $c \triangleq \gamma(a, b)$ y el sistema $R = a \cdot P || b \cdot Q$ el resultado será $R_1 = c \cdot (P' || Q')$ porque esto sucede siempre que se da un evento de comunicación, sin embargo, no existe un axioma en ACP que pueda ser utilizado para establecer esta igualdad.

7.4. Trabajo futuro

Como se menciona la final del capítulo 4 el enfoque aquí planteado es apenas el inicio del camino hacia la solución de problemas con sistemas concurrentes. De acuerdo con modelo propuesto por Pancake y Bergman presentado en el mismo capítulo, apenas se han atacado dos pasos en la representación de soluciones concurrentes, por lo que todavía hay mucho camino por recorrer.

El siguiente paso es buscar una representación de la solución en algún lenguaje de programación, sin embargo, antes de llegar ella se deben resolver algunos retos importantes:

- Analizar la complejidad de la solución. Este tema ya ha sido tratado en textos como [Tay83] y mas recientemente en [LDM$^+$12], aunque el enfoque utilizado es diferente al propuesto. La importancia de abordar este reto radica en el hecho de poder acotar las soluciones propuestas en términos de complejidad computacional y saber si su implementación es factible con respecto al dominio de datos que pueda recibir.
- Mejorar la eficiencia de los sistemas concurrentes. La idea utilizar el álgebra como herramienta para la representación de procesos es que también pueda ser de utilidad para realizar análisis de las soluciones generadas y mejorarlas en términos de tiempo de ejecución empleado, ya sea aplicando las fórmulas definidas o generando fórmulas nuevas, extendiendo el álgebra de procesos. Un reto con respecto a este tópico es aumentar de la expresividad del álgebra de procesos para representar *timing*. Todo sistema en ejecución depende del entorno en el cual se

ejecute, en particular de la disponibilidad de los recursos necesarios para su funcionamiento y también de la correcta sincronización entre procesos, lo cual afectará su desempeño, por lo tanto, representar retardos de tiempo, intervalos de espera y otros fenómenos relacionados con el timing es un paso obligado en el para el análisis de la eficiencia de un sistema con procesos concurrentes.

- Balancear las cargas de trabajo. En los entornos multicore el número de recursos de procesamiento está bien definido, por lo tanto es posible pensar en soluciones concurrentes que contemplen de antemano este dato para determinar cual es la cantidad de procesos concurrentes necesarios para un funcionamiento que aproveche mejor los recursos y eficiente la ejecución.
- Búsqueda de algoritmos que contemplen como un parámetro fundamental la cantidad de procesadores de la plataforma sobre la cual se ejecutará el sistema. Una mejora al punto anterior sería que el propio sistema concurrente tuviera como primer procesamiento la detección del número de procesadores y su correspondiente re-organización para lograr escalar sistemas a entornos multicore cada vez mayores.

Un trabajo alternativo que puede facilitar aún mas la definición de representaciones algorítmicas puede ser la elaboración de estructuras algebraicas correctas para cada uno de los patrones para diseño de software paralelo y utilizarlas en las soluciones a problemas dados una vez que identifique el tipo de paralelismo y de sistema adecuados.

Inclusive la idea general de separar las acciones de procesos concurrentes puede servir como base para realizar análisis de otro tipo de sistemas que requieran la especialización de tareas, división de datos y el trabajo colaborativo.

Por otro lado, una posible mejora para este trabajo es la automatización del proceso de derivación, ya que se tiene una axiomatización bien definida y la posibilidad de realizar definiciones de forma muy simple, lo cual puede facilitar la construcción de un sistema que reciba como entrada la definición de los procesos y las comunicaciones, realice las derivaciones guiado por la axiomatización del álgebra de procesos y arroje como salida las acciones que cada proceso puede ejecutar.

El presente trabajo define un modelo de comunicación por medio de eventos de envío y recepción

de información que forman parte de los procesos de un sistema concurrente, es decir, se basa en el enfoque de paso de mensajes, sin embargo, se podría pensar en una versión de este enfoque para memoria compartida o inclusive que funciona para ambos modelos de comunicación.

También es posible contemplar la idea de lleva este modelo a otros entornos menos limitados, como una red de computadoras y buscar la solución a problema de sistemas distribuidos, siempre y cuando se contemple también la representación del tiempo, ya que el modelo de comunicación es similar.

GLOSARIO

A. Definición del álgebra para procesos concurrentes (ACP)

Álgebra de Procesos Básica (BPA)

A continuación se presentan los elementos básicos que conforman el álgebra de procesos definida en [Fok99]. Por un lado, su sintaxis se puede ver como la definición de su estructura desde el punto de vista puramente formal [CK89] donde se definen aspectos como el tamaño de una sentencia, el conjunto de símbolos que la conforman o la definición de estos últimos y la herramienta formal utilizada para este propósito es la lógica de ecuaciones donde los elementos centrales son las ecuaciones. Por otro lado, como ya se había mencionado, se utiliza la semántica operacional para realizar descripciones de sistemas a partir de reglas de transición[AFV99].

El álgebra de procesos básica (BPA por sus siglas en inglés) se enfoca en la especificación y manipulación de términos por medio de una colección de operadores. Todos los elementos mencionados se definen dentro de una Signatura [1], que en caso de BPA está definida como sigue,

$$\Sigma = A \cup \{+, \cdot\}$$

Donde:

- **A** es un conjunto de acciones atómicas que representan comportamiento indivisible. Cada acción atómica a es una constante (un término proceso con aridad cero) que puede ejecutarse a sí misma y terminar exitosamente, representada de la forma $a \xrightarrow{a} \surd$.

[1] Signature en inglés

- $(+)$ es un operador binario llamado composición alternativa. Si los términos cerrados t_1 y t_2 representan los procesos p_1 y p_2 respectivamente, entonces el término cerrado $t_1 + t_2$ representa el proceso que se ejecuta, ya sea p_1 o p_2. Esta elección es no determinista y no existe modo de controlar o predecir cual será el término que se ejecutará.
- $(\cdot)$ es un operador binario llamado composición secuencial. Si los términos cerrados t_1 y t_2 representan los procesos p_1 y p_2 respectivamente, entonces el término cerrado $t_1 \cdot t_2$ representa el proceso que ejecuta primero s_1 y luego s_2, es decir, la última accción de p_1 lleva a la raíz de p_2. También es posible omitir el símbolo y solo colocar los términos si esto no causa confusión.

Otra forma de representar términos bajo este enfoque de álgebra es a través de un gráfico de proceso o *Sistemas de Transiciones Etiquetados* (LTS) donde cada nodo representa un estado diferente del sistema y cada arista expresa que el sistema puede cambiar de un estado a otro cuando sucede la acción etiquetada.

A continuación se presentan un par de ejemplos acerca de la representación de procesos y su significado. En la figura A.1 si un término se encuentra en el estado $c \cdot (a + b)$ y sucede la acción c, pasa al estado $a + b$, o de forma mas compacta, $c \cdot (a + b) \xrightarrow{c} a + b$ y este último puede ejecutar a o b y terminar $(a + b \xrightarrow{a} \surd \vee a + b \xrightarrow{b} \surd)$.

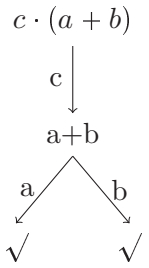

Figura A.1.: Representación del término $c \cdot (a + b)$

En la figura A.2, cuando el término es $((a + b) \cdot c) \cdot d$, se puede ejecutar a o b, dividiéndose en dos caminos idénticos pero previamente determinados por la ejecución reciente.

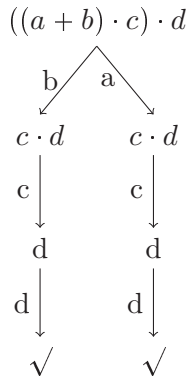

Figura A.2.: Representación del término $((a + b) \cdot c) \cdot d$

Equivalencia entre términos

En BPA existen otros conceptos cuyo propósito es principalmente establecer mecanismos para saber cuando dos términos son equivalentes, lo cual es una tarea compleja ya que ni siquiera con una representación gráfica como un LTS es posible realizar una identificación inmediata, ya que no basta que dos términos puedan realizar las mismas acciones, sino que el esquema de la ruta seguida (la ramificación) debe ser similar. A través de equivalencias semánticas entre LTS's es posible distinguir igualdades entre ellos, sin embargo, para lograr un tipo de equivalencia que permita asegurar que dos elementos tengan el mismo comportamiento es necesaria una relación muy rígida que tome en cuenta cada nivel del LTS.

La relación llamada Bisimulación equivalente permite establecer una relación entre dos términos que pueden ejecutar las mismas transiciones y contar con la misma estructura de derivación ya que diferentes derivaciones pueden implicar un comportamiento distinto, debido a que se rompería la relación para al menos un término.

Por ejemplo sea el término $r \triangleq (a \cdot b) + (c\dot{d})$, si se requiere un término $r' \equiv r$ módulo Bisimulación equivalente es necesario encontrar $a' \equiv a$, $b' \equiv b$, $c' \equiv c$, $d' \equiv d$, $ab' \equiv a \cdot b$,$cd' \equiv c \cdot d$ módulo Bisimulación equivalente, tal que a', b', c', d, ab', cd' formen parte de r' debido a que la definición de los términos es recursiva.

Esta relación es muy útil ya que gracias a ella es posible definir igualdades entre términos simples

dando origen a una Axiomatización que permite trabajar con términos en cualquier nivel de un LTS. A lo largo de las extensiones realizadas a BPA se hará referencia a los nuevos axiomas que se van agregando y la axiomatización completa se puede consultar en el Apéndice B, en particular, los axiomas definidos solo para BPA son del A01 al A05.

Álgebra de Procesos con Paralelismo (PAP)

Esta es la primera extensión de BPA la cual aporta los operadores necesarios para expresar concurrencia y capturar el comportamiento de un sistema que se compone de procesos ejecutándose concurrentemente donde un proceso puede influir en el comportamiento de otro a través de las comunicaciones definidas para cumplir con un objetivo común. Para expresar concurrencia se introduce el operador **merge** (||) de la siguiente forma:

$$s||t$$

Donde s y t son términos procesos. Este operador binario ejecuta sus argumentos en paralelo de forma no determinista. Esto significa que $s||t$ puede elegir ejecutar una transición inicial del término s o del t, cambiándo ese término a su estado siguiente (s' o t'), pero conservando al otro intacto y listo para una segunda ejecución de alguna acción de cualquiera.

En la Tabla A.1 se desarrollaron cuatro ejecuciones del término $a \cdot b||c \cdot d$, como se puede observar en cada columna y al final de la misma se enumera el orden de la ejecución. La Ejecución I primero ejecuta el término $a \cdot b$ y luego el $c \cdot d$ y la Ejecución IV hace lo contrario. Sin embargo, un mejor escenario para analizar el comportamiento de || son las ejecuciones II y III, ya que eligen uno de los términos y ejecutan la primera acción, pero el otro término lo dejan intacto como se ve en el paso 1.

Este proceso puede representarse de forma gráfica, sin embargo, debido a su complejidad, el LTS es muy grande y se vuelve difícil su construcción. Por lo tanto, utilizar gráficos para representar procesos suele ser poco práctico y aunque brinda una buena idea de su comportamiento su manipulación puede complicar el problema en lugar de resolverlo.

Por otro lado, el operador || puede elegir ejecutar una comunicación entre las transiciones iniciales

PASO	Ejecución I	Ejecución II	Ejecución III	Ejecución IV
0	$a \cdot b \| \| c \cdot d$	$a \cdot b \| \| c \cdot d$	$a \cdot b \| \| c \cdot d$	$a \cdot b \| \| c \cdot d$
1	$b \| \| c \cdot d$	$b \| \| c \cdot d$	$a \cdot b \| \| d$	$a \cdot b \| \| d$
2	$c \cdot d$	$b \| \| d$	$b \| \| d$	$a \cdot b$
3	d	d	b	b
Secuencia:	a, b, c, d	a, c, b, d	c, a, b, d	c, d, a, b

Tabla A.1.: Algunas ejecuciones posibles del término $a \cdot b||c \cdot d$

de s y t, a través de la función **comunicación**, definida como

$$\gamma : A \times A \to A$$

Donde **A** es el conjunto de acciones atómicas antes definido. Para utilizar esta relación es necesario definir explícitamente dos términos constantes, uno para transmitir un dato determinado por un canal especificado (x) y otro para recibir el mismo dato por el mismo canal (y) y la relación de comunicación entre ambos términos sería de la forma $r = \gamma(x, y)$, es decir, la interacción entre los términos x, y también es un término, pero mas importante, es una acción atómica de acuerdo con la definición de γ, por lo que $r \in A$.

La función γ es conmutativa ya que su comportamiento no depende del lugar que ocupen los término al establecer γ sino de su definición. Supóngase por ejemplo que se definen las acciones $a, c, e \in A$ como sigue:

- $a \triangleq$ {Envía dato 1 por el canal α}
- $c \triangleq$ {Recibe dato 1 por el canal α}
- $e \triangleq \gamma(a, c)$

Cambiar la definición de e por $e \triangleq \gamma(c, a)$ no hace ninguna diferencia en el comportamiento del término. Al ser e una acción atómica, el sistema debe ser capaz de ejecutar sin interrupción las acciones a, c simultáneamente. Retomando el ejemplo del término $a \cdot b||c \cdot d$ con las nuevas definiciones de a y c, las ejecuciones son diferentes, tal como se ve en el cuadro A.2.

Como ya se mencionó anteriormente, es importante que el álgebra de procesos definida y cualquier

PASO	Ejecución I	Ejecución II
0	$a \cdot b \| \| c \cdot d$	$a \cdot b \| \| c \cdot d$
1	$b \| \| d$	$b \| \| d$
2	b	d
Secuencia:	e, b, d	e, d, b

Tabla A.2.: Ejecuciones posibles del término $a \cdot b||c \cdot d$ con $e = \gamma(a, c)$

extensión conserve una semántica correcta y completa, sin embargo, tal como menciona [Mol90], el operador *merge* no tiene un comportamiento adecuado para obtener una PAP correcta y completa, es decir, no es lo suficientemente expresiva, por ello se introdujeron dos operadores nuevos:

1. **Left merge (s⌊⌊t)**. Este operador toma como primera acción (siempre) al proceso de la izquierda, en este caso s y ejecuta su primera acción atómica; el proceso t permanece igual y una vez realizado lo anterior, el sistema se comporta como si tuviera el operador *merge*.
2. **Comunication merge ($s|t$)**. Ejecuta una comunicación entre la primera acción atómica de s y la primera acción atómica de t, comportándose luego como *merge*.

Una vez definidos los operadores anteriores, la Signatura sería la siguiente:

$$\Sigma_{PAP} = A \cup \{+, \cdot, \|, \lfloor\lfloor, |\}$$

Los axiomas relacionados con PAP son del A06 al A15 del glosario B.

Deadlock y Encapsulación

Un evento de comunicación $\gamma(a, b)$ tiene sentido si las acciones atómicas a y b se ejecutan dentro de una instrucción de comunicación ($|$) y no de forma aislada, sin embargo, este último caso se puede suscitar y es necesario modelarlo, por lo tanto se introduce una constante llamada **deadlock** (δ). y se redefine la función de comunicación para incluirla: $\gamma : A \times A \to A \cup \delta$. Ahora es posible definir que dos acciones, $a, b \in A$ no se comunican, definiendo $\gamma(a, b) \triangleq \delta$, aunque en la práctica se asume que cualquier evento de comunicación no definido llevará al sistema a δ.

Una ejecución aislada de alguno de los eventos definidos como partes de una comunicación, como a y b en $\gamma(a, b)$ no puede darse en un sistema de procesos concurrentes y debe evitarse a toda costa, por lo cual se introduce un operador unario de encapsulación (∂_H) para un conjunto H de acciones atómicas, las cuales llevarán al sistema a δ. Esto significa que $\partial_H(t)$ puede ejecutar todas las acciones de t que no se encuentren en H.

Los axiomas relacionados son del A16 al A25 del glosario B.

Comportamiento infinito

Hasta ahora solo se han modelado procesos finitos, pero si se tiene algún sistema cuya ejecución no deba terminar por ser requerido de esa forma, se debe introducir una nueva forma de representación. Para modelar comportamiento infinito de procesos se utiliza un conjunto de ecuaciones llamadas recursivas, donde el lado izquierdo es una variable recursiva y del lado derecho términos procesos con posibles ocurrencias de las variables recursivas, es decir:

$$\begin{aligned} X_1 &= t_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= t_n(X_1, \ldots, X_n) \end{aligned}$$

La solución de este sistema será entonces un grupo de n procesos $p_1, \ldots, p_n$, los cuales reemplazarán a cada variable recursiva en el lado derecho de la ecuación y cada proceso p_i será equivalente a cada $t_i(p_1, \ldots, p_n)$, por ejemplo, $E = \{X = aY; Y = bX\}$ es una representación recursiva cuya solución es $p_1 = ababa\ldots$ y $p2 = babab\ldots$ si sustituimos X por p_1 y Y por p_2. Las soluciones se denotan como $< X|E >$ y $< Y|E >$ respectivamente, es decir, $< X|E >$ es el proceso que sustituye a la variable recursiva X en la representación, en este caso, el proceso $p1$ y lo mismo sucede para $p2$. Cabe mencionar que cada representación solo puede tener una solución en general y si se encuentra mas de una ambas son equivalentes, aunque existen casos donde se tiene mas de una solución pero esto se debe a que no se definieron ecuaciones custodiadas.

Con esta representación nuevamente se realiza una extensión de ACP para incluir esta notación en las reglas de derivación y los axiomas correspondientes son del A26 al A28.

Silent step

La constante τ, llamada *Silent Step*, representa una secuencia de acciones que pueden ser eliminadas de un gráfico de proceso o una acción atómica que se ejecuta a sí misma y termina satisfactoriamente ($\tau \xrightarrow{\tau} \surd$). El operador comunicación cambia su dominio y ahora puede tomar a τ como una acción atómica mas, pero si τ está involucrada en cualquier comunicación, el resultado siempre será deadlock (δ), lo cual resulta útil para verificar si existen acciones que efectivamente no sean importantes en la ejecución de un programa o si su eliminación cambia la definición de una solución, por ejemplo, la ecuación $a + \tau(a + b)$ es equivalente a $(a + b)$ ya que $a + \tau(a + b) \Leftrightarrow a + a + b \Leftrightarrow a + b$.

Cambiando un poco la notación de τ a τ_I se define un operador que recibe un argumento, donde I es un conjunto que contiene una serie de acciones que se quiere ocultar. Este operador fue diseñado para verificar que el programa sea correcto, es decir, que se comporte como lo especificó el usuario, ocultando aquellas definiciones que no cambian el comportamiento del sistema, como podrían ser cálculos internos o también permite extraer algunos comportamientos internos del programa para hacerlos evidentes a otro nivel.

Los axiomas derivados de los conceptos anteriores son del A29 al A35.

Notación

Secciones secuenciales

Debido a que la representación algebraica debe reflejar principalmente las interacciones del sistema, una forma de simplificarla es tomando las funcionalidades secuenciales propias de cada proceso como una acción atómica. Por ejemplo, si se requiere un ordenamiento solamente se define una acción, como se puede ver A.1, asumiendo que se ejecutarán siempre sin interrupción y su programación es totalmente secuencial.

$$o \triangleq \text{ Ordena el arreglo R de forma ascendente.} \tag{A.1}$$

Donde $o \in \Sigma$ y R pertenece al conjunto de variables del sistema. Para el problema 2 se pueden establecer las acciones atómicas de la definición A.2.

$$\begin{aligned} a &\triangleq \text{Invierte un número binario} \\ b &\triangleq \text{Concatena a la izquierda un número binario y separa el de la derecha} \end{aligned} \tag{A.2}$$

Comunicaciones

De acuerdo al patrón seleccionado se definen los canales y eventos de comunicación, estos últimos son parejas de acciones, ya que se debe definir una para enviar y otra para recibir datos, así como su vinculación por medio de la función δ, como se ve en la definición A.3, donde cada acción de comunicación está vinculada con un canal y un dato involucrado en el proceso. Los canales de comunicación se representan como cualquier variable del sistema pero los datos permitidos se especifican como un conjunto $\Delta = \{x|x \text{ es un dato válido para ser enviado por el canal C}\}$. En este caso los términos atómicos a y b complementan el evento de comunicación para el dato $0 \in \Delta$, mientras que c y d lo hacen para el dato $1 \in \Delta$. Debido a que se define una acción atómica, se debe definir el conjunto Δ con valores del tipo de dato mas básico posible que puedan ser enviados por un canal.

$$\begin{aligned} \Delta &\triangleq \{0, 1\} \\ a &\triangleq \text{Envía el dato 0 por el canal C} \\ b &\triangleq \text{Recibe el dato 0 por el canal C} \\ s &\triangleq \gamma(a, b) \\ c &\triangleq \text{Envía el dato 1 por el canal C} \\ d &\triangleq \text{Recibe el dato 1 por el canal C} \\ t &\triangleq \gamma(c, d) \end{aligned} \tag{A.3}$$

Con la notación antes vista no es posible representar eventos de comunicación de una forma implícita, es decir, no definir cada uno ellos explícitamente sino utilizar símbolos que representen un conjunto de ellos, por lo que se modificará un poco agregando subíndices en los términos y un argumento, convirtiéndolos en términos unarios pero cerrados. Los subíndices representan el canal

por el que transmite o recibe el dato y el argumento será el dato a enviar, como se encuentra ejemplificado en la definición A.4, donde $d \in \Delta$ se convierte en el dato a transmitir y C el canal por el que se realiza la comunicación. A través de esta definición es posible establecer los eventos de comunicación válidos para todos los datos posibles con solo dos términos. Una observación muy importante es que el argumento de e es el mismo que el de r y los subíndices también son iguales, por lo que el evento de comunicación solo puede ser completo si se transmite y recibe el mismo dato por el mismo canal, como en el caso particular de $s_C(1) = \gamma(e_C(1), r_C(1))$.

$$\begin{aligned} \Delta &\triangleq \{0, 1, 2, 3, 4\} \\ e_C(d) &\triangleq \text{Envía el dato d por el canal C} \\ r_C(d) &\triangleq \text{Recibe el dato d por el canal C} \\ s_C(d) &\triangleq \gamma(e_C(d), r_C(d)) \end{aligned} \tag{A.4}$$

Generalmente se establece la restricción de que si un evento de comunicación no se encuentra definido siempre será igual a δ, con el objetivo de evitar ambigüedades al realizar el proceso de derivación

Procesos

Los procesos son términos que se representan con letras mayúsculas y se definen a partir de al menos un término simple válido (como una acción) y se puede acompañar de otros procesos, relacionándose a través de los operadores de la Signatura definida para el álgebra de procesos utilizada.

$$\Sigma_{PAP+ACP} = A \cup \{+, \cdot, \|, \lfloor\lfloor, |, \partial, \tau\}$$

En la definición A.5 se muestra la definición de un proceso P con términos $a, b, c \in A$, donde se ejecuta a secuencialmente y posteriormente se puede ejecutar b o c, los cuales también deben ser

especificados antes de su uso.

$$\begin{aligned} a &\triangleq \text{Se ejecuta el cálculo r=16x} \\ b &\triangleq \text{Se envía r por el canal C} \\ c &\triangleq \text{Se envía x por el canal C} \\ P &\triangleq a \cdot (b + c) \end{aligned} \tag{A.5}$$

Si se toma como base la definición A.4, se pueden definir los procesos que ejecutan el envío y recepción del dato 1, como se puede apreciar en la definición A.6. También se debe define que ambos procesos serán ejecutados de forma concurrente.

$$\begin{aligned} P &\triangleq e_C(1) \\ Q &\triangleq r_C(1) \\ R &\triangleq P||Q \end{aligned} \tag{A.6}$$

Si se requiere una decisión no determinista en el sentido de que no se establece un orden para que llegue o se transmita un dato, dejando abiertas las posibilidades de que se de en cualquier orden, es posible especificarlo con el símbolo "+", redefiniendo los procesos de A.6 y considerando todas las posibilidades de Δ definida en A.4. Esto se puede apreciar en la definición A.7, la cual solo ejecutará una de las instrucciones de P y una de Q.

$$\begin{aligned} P &\triangleq e_C(0) + e_C(1) + e_C(2) + e_C(3) + e_C(4) \\ Q &\triangleq r_C(0) + r_C(1) + r_C(2) + r_C(3) + r_C(4) \\ R &\triangleq P||Q \end{aligned} \tag{A.7}$$

Aunque la notación anterior facilita la especificación de procesos con comunicaciones puede suceder que Δ sea un conjunto demasiado grande. Tomando en cuenta este caso, es posible hacer otro ajuste a la representación de los procesos introduciendo el símbolo de sumatoria (Σ) común con dos particularidades: la primera es que no se especifica la suma de elementos tomando sus límites inferior y superior, sino especificando un conjunto de elementos involucrados tomados en cualquier orden, lo que significa realmente es que existe la disponibilidad de ejecutar una instrucción que involucra a cualquiera de los elmentos del conjunto especificado; la segunda es que este símbolo

solo debe interpretarse así en el contexto de la definición de un término, por lo que no debe haber confusión símbolo que define la Signatura. La definición A.8 aplica este principio en los procesos P y Q de la definición A.7.

$$\begin{aligned} P &\triangleq \sum_{d\in\Delta} e_C(d) \\ Q &\triangleq \sum_{d\in\Delta} r_C(d) \\ R &\triangleq P||Q \end{aligned} \tag{A.8}$$

En la figura 5.6 se establecen los canales de comunicación como lo establece el enunciado del problema 2 añadiéndo un canal de comunicación interna entre procesos. La definición A.9 establece los procesos para este problema, los cuales incluyen los eventos de comunicación y las acciones atómicas definidas en A.2, por lo tanto,

$$A = \{r_R(0), r_R(1), a(0), a(1), b(0), b(1), e_C(0), e_C(1), r_C(0), r_C(0), e_E(0), e_E(1), s_C(0), s_C(1)\}.$$

$$\begin{aligned} \Delta &\triangleq \{0,1\} \\ r_R(d) &\triangleq \text{Recibe el dato d por el canal R}(d \in \Delta) \\ a &\triangleq \text{Invierte un número binario} \\ e_C(d) &\triangleq \text{Envía el dato d por el canal C}(d \in \Delta) \\ r_C(d) &\triangleq \text{Recibe el dato d por el canal C}(d \in \Delta) \\ b &\triangleq \text{Concatena a la izquierda de N un número binario y separa el de la derecha} \\ e_E(d) &\triangleq \text{Envía el dato d por el canal E}(d \in \Delta) \\ s_C(d) &\triangleq \gamma(e_C(d), r_C(d)) \\ P &\triangleq \sum_{d\in\Delta} r_R(d) \cdot a \cdot \sum_{d\in\Delta} e_C(d) \\ Q &\triangleq (r_C(1) + r_C(2)) \cdot b \cdot (e_E(0)e_E(1)) \\ R &\triangleq P||Q \end{aligned} \tag{A.9}$$

El conjunto Δ es el de los número binarios, por lo tanto se tienen acciones atómicas con parámetro constante 0 y las mismas con parámetro 1, ya que deben ser constantes los argumentos para que sea

un término cerrado. La acción $r_R(d)$ está definida para recibir un dato del exterior, aunque no se especifica si viene de otro proceso, de una captura o alguna otra forma de ingreso de información, basta con saber que la recepción se realizará y lo mismo sucede para la transmisión $e_E(d)$. El procesos P se define con sumatorias mientras que Q con el símbolo de suma normal, sin embargo solo es para propósitos de utilizar ambas notaciones, ya que este caso es posible y no complica demasiado los términos debido a que el conjunto Δ es pequeño.

El deadlock

Al definir procesos concurrentes, existe la posibilidad de que el sistema pueda caer en un estado en el cual no pueda seguir su ejecución, pero si se analiza este hecho con detenimiento, tomando en cuenta la definición de términos anteriormente descrita, solo puede suceder esto en los eventos de comunicación, ya que las demás secciones de código son consideradas como acciones que siempre van a terminar exitosamente, incluyendo las que realizan las reglas de negocio o los cálculos matemáticos.

Para evitar que un evento de comunicación pueda provocar un estado del sistema no deseable se utiliza la constante δ (*deadlock*), la cual expresa que dos acciones $f, g \in \Delta$ no pueden establecer comunicación, por lo que es necesario redefinir el rango de la función γ, resultando $\gamma : A \times A \rightarrow A \cup \delta$ y la ausencia de comunicación se especificará como $\delta \triangleq \gamma(f, g)$.

La implicación que tiene δ en la ejecución de un sistema es nula, es decir, cuando se tiene la acción atómica $\gamma(a, b)$ el sistema se queda en el estado actual, por lo que se puede decir que δ es una acción nula o simplemente una inacción.

Si además se toma en cuenta que dentro de la definición de las acciones atómicas puede haber muchas que no se pueden comunicar debido a su propia definición, realizar la definición $\delta \triangleq \gamma(a_1, a_2)$ con $a_1, a_2 \in \Delta$ puede ser muy extensa. Para evitar esto se utiliza el operador ∂_H, donde H es un conjunto de acciones que caerán en deadlock al ejecutarse, es decir cada acción es igual a δ, este operador es uniario y tiene un argumento que es el término que ejecutará una acción. Una de las ventajas mas importantes de la presente definición de álgebra de procesos es precisamente la inclusión de la contante δ y el operador ∂ debido a que de manera explícita se puede garantizar

la concurrencia evitando que sistema vaya hacia estados que no le permitan continuar, lo cual significa que siempre que se defina un sistema se tomará este aspecto en consideración. Retomando el problema 2 y la definición A.9 se pueden cambiar el proceso R como se establece en la definición A.10 donde se ejemplifica el uso de ∂_H, tomando como término principal a $P||Q$ de la definición A.8 y como H todos los eventos $e_C(d), r_C(d)$, con $d \in \Delta$, lo cual significa que si alguna de estas acciones se pretendiera ejecutar, ∂_H no la deja evitando que el sistema caiga en deadlock, lo cual significa que el proceso Q solo podrá realizar la transmisión de un dato si efectivamente recibe la comunicación exitosa de alguno de los datos.

$$R = \partial_{\{e_C(d), r_C(d) | \forall d \in \Delta\}}(P||Q) \tag{A.10}$$

Código oculto

La constante τ se utiliza para representar una o mas acciones que pueden ser omitidas en la especificación de un sistema, es decir, la definición de un sistema se basa en la funcionalidad de mas alto nivel que se desea analizar desde el punto de vista del álgebra de procesos, sin embargo ciertas acciones que ayudan a construir el sistema no se reflejan en dicha definición y esas son las que están representadas por τ.

El uso de esta constante es delicado por varias razones, las primera es que se debe tener mucho cuidado en el nivel de abstracción de las acciones representadas para no omitir comportamientos importantes; la segunda es que puede darse el caso de en realidad las acciones omitidas si sean relevantes. Inclusive el mal uso de τ puede acarrer errores tan importantes como asumir que dos procesos pueden ser equivalentes (o bisimilares) cuando en realidad no lo son si se tman en cuenta las acciones omitidas por τ. Por ejemplo, el proceso a es equivalente a $a + \delta$, ya que por la definición misma de δ, siempre se elegirá la acción a sobre ella, sin embargo, $a + \tau\delta$ no lo es, ya que al ejecutarse τ habilita la ejecución de δ y por tanto el sistema caerá en deadlock, esto significa que τ es relevante en el comportamiento de mas alto nivel del sistema. Debido a estas peculiaridades existen ciertas igualdades incluídas en la semántica del álgebra de procesos que permiten el mejor uso de τ, o como Milner la llama, el *Silent Step*.

También es posible especificar las acciones omitidas por τ utilizando el operador unario τ_I, donde

$I \subset A$, el cual renombra todas las acciones que contiene y las hace τ. El comportamiento de este operador es el siguiente, si la acción $v \notin I$ se ejecuta el sistema realiza una transición como cúnmente ocurre, pero si $v \in I$ entonces la acción no estará disponible para su ejecución y se realizará de forma interna al ejecutarse τ_I. Un ejemplo representativo de el uso de τ es precisament eel problema 2 ya que en el enunciado inicial no se habla de una comunicación interna, eso es producto de la definición del sistema por lo tanto la transmisión de datos por el canal C no es un comportamiento que defina el sistema pero si es proceso interno necesario, es decir, se debe definir. Por lo tanto se puede cambiar la definición A.10 del proceso R por la definición A.11, tomando en cuenta la comunicación $s_C(d)$ y Δ de la definición A.9.

$$R \triangleq \tau_{\{s_C(d)|d\in\Delta\}}(\partial_{\{e_C(d),r_C(d)|\forall d\in\Delta\}}(P||Q)) \tag{A.11}$$

Procesos infinitos

La definición A.9 desglosa los procesos involucrados en la solución del problema 2, sin embargo, al evolucionar los procesos P y Q se debe hacer notar que en cuanto se hace una comunicación entre ellos y se transmite el primer dato ambos procesos terminan. Sin embargo, al tratarse de buffers se requiere que sigan transmitiendo datos de forma continua, por lo tanto, la definición de los procesos antes mencionados se pueden modificar para dar lugar a ecuaciones recursivas (definición A.12) que modelen el comportamiento deseado. Es importante notar que esta especificación recursiva es por medio de ecuaciones custodiadas y por ello, se modela una solución única módulo Bisimulación.

$$\begin{aligned} P &\triangleq \sum_{d\in\Delta} r_R(d)\cdot a \cdot \sum_{d\in\Delta} e_C(d)\cdot P \\ Q &\triangleq (r_C(0)+r_C(1))\cdot b\cdot(e_E(0)+e_E(1))\cdot Q \end{aligned} \tag{A.12}$$

B. Axiomatización para ACP

BPA

A01	$x + y = y + x$
A02	$(x + y) + z = x + (y + z)$
A03	$x + x = x$
A04	$(x + y) \cdot z = x \cdot z + y \cdot z$
A05	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$

PAP

A06	$p \| \| q = p \lfloor \lfloor q + q \lfloor \lfloor p + p \| q$
A07	$v \lfloor \lfloor y = v \cdot y$
A08	$(v \cdot x) \lfloor \lfloor y = v \cdot (x \| \| y)$
A09	$(x + y) \lfloor \lfloor z = x \lfloor \lfloor z + y \lfloor \lfloor z$
A10	$v \| w = \gamma(v, w)$
A11	$v \| (w \cdot y) = \gamma(v, w) \cdot y$
A12	$(v \cdot x) \| w = \gamma(v, w) \cdot x$
A13	$(v \cdot x) \| (w \cdot y) = \gamma(v, w) \cdot (x \| \| y)$
A14	$(x + y) \| z = x \| z + y \| z$

A15 $x|(y+z) = x|y + x|z$

Deadlock y Encapsulación

A16 $x + \delta = x$

A17 $\delta \cdot x = \delta$

A18 $\partial_H(v) = v \ \texttt{si} \ v \notin H$

A19 $\partial_H(v) = \delta \ \texttt{si} \ v \in H$

A20 $\partial_H(\delta) = \delta$

A21 $\partial_H(x + y) = \partial_H(x) + \partial_H(y)$

A22 $\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$

A23 $\delta \lfloor\lfloor x = \delta$

A24 $\delta|x = \delta$

A25 $x|\delta = \delta$

Comportamiento infinito

A26 $\langle X_i|E\rangle = t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle \ \texttt{para} \ i \in \{1 \ldots n\}$

A27 $\texttt{Si} \ y = t_i(y_1, \ldots, y_n) \ \texttt{para} \ i \in \{1 \ldots n\} \ \texttt{entonces} \ y_i = \langle X_i|E\rangle \ \texttt{para} \ i \in \{1 \ldots n\}$

A28 $Y = t_i \cdot X \ \texttt{si} \ Y \triangleq t_i \cdot X$

Silent step

A29 $v \cdot \tau = v$

A30 $v \cdot (\tau \cdot (x + y) + x) = v \cdot (x + y)$

A31 $\tau_I(v) = v \ \texttt{si} \ v \notin I$

A32 $\tau_I(v) = \tau$ `si` $v \in I$

A33 $\tau_I(\delta) = \delta$

A34 $\tau_I(x + y) = \tau_I(x) + \tau_I(y)$

A35 $\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$

C. Desarrollo algebraico de los problemas planteados

C.1. Convenciones

A continuación se describen algunas convenciones cuya definición tiene el propósito de lograr una definición mas clara de los problemas planteados.

- Los datos serán representados por la letra d, con la característica $d \in \Delta$.
- Las acciones atómicas por letras minúsculas a partir de la f.
- Los canales de comunicación serán representados por subíndices numéricos, por lo tanto
- $a_{canal}(d)$ Envía del dato d por el canal, $d \in \Delta$.
- $b_{canal}(d)$ Recibe el dato d por el canal, $d \in \Delta$.
- $c_{canal}(d)$ Definición de acción de comunicación del dato d por el canal, $d \in \Delta$.

C.2. Problema 1

C.2.1. Definición

$$
\begin{aligned}
\Delta &\triangleq \{0,1\} \\
H &\triangleq \{a_2(d), b_2(d) | \forall d \in \Delta\} \\
I &\triangleq \{c_2(d) | d \in \Delta\} \\
f &\triangleq \text{Invierte un número binario} \\
g &\triangleq \text{Concatena a la izquierda de N un número binario y separa el de la derecha} \\
b_1(d) &\triangleq \text{Recibe el dato d por el canal R}(d \in \Delta) \\
b_2(d) &\triangleq \text{Recibe el dato d por el canal C}(d \in \Delta) \\
a_2(d) &\triangleq \text{Envía el dato d por el canal C}(d \in \Delta) \\
a_3(d) &\triangleq \text{Envía el dato d por el canal E}(d \in \Delta) \\
c_2(d) &\triangleq \gamma(a_2(d), b_2(d)) \\
P &\triangleq \sum_{d \epsilon \Delta} b_1(d) \cdot f \cdot \sum_{d \epsilon \Delta} a_2(d) \cdot P \\
Q &\triangleq \sum_{d \epsilon \Delta} b_2(d) \cdot b \cdot \sum_{d \epsilon \Delta} a_3(d) \cdot Q \\
R &\triangleq \tau_I(\partial_H(P||Q))
\end{aligned}
\tag{C.1}
$$

C.2.2. Verificación

Como primer paso, se derivará el término $P||Q$, tomando en cuenta que si una comunicación entre dos términos no se encuentra definida siempre es igual a δ.

$$
\begin{aligned}
R =& \tau_I(\partial_H(P||Q)) = \tau_I(\partial_H(P \lfloor\lfloor Q + Q \lfloor\lfloor P + P|Q)) \\
=& \tau_I(\partial_H(P \lfloor\lfloor Q + Q \lfloor\lfloor P + (b_1(d) \cdot f \cdot a_2(d) \cdot P)|(b_2(d) \cdot g \cdot a_3(d) \cdot Q))) \\
=& \tau_I(\partial_H(P \lfloor\lfloor Q + Q \lfloor\lfloor P + b_1(d) \cdot (f \cdot a_2(d) \cdot P)|b_2(d) \cdot (g \cdot a_3(d) \cdot Q))) \\
=& \tau_I(\partial_H(P \lfloor\lfloor Q + Q \lfloor\lfloor P + \delta \cdot ((f \cdot a_2(d) \cdot P)|(\cdot g \cdot a_3(d) \cdot Q)))) \\
=& \tau_I(\partial_H(P \lfloor\lfloor Q + Q \lfloor\lfloor P + \delta)) \\
=& \tau_I(\partial_H(P \lfloor\lfloor Q + Q \lfloor\lfloor P)) \\
=& \tau_I(\partial_H((b_1(d) \cdot f \cdot a_2(d) \cdot P) \lfloor\lfloor Q + (b_2(d) \cdot g \cdot a_3(d) \cdot Q) \lfloor\lfloor P)) \\
=& \tau_I(\partial_H(b_1(d) \cdot (f \cdot a_2(d) \cdot P||Q) + b_2(d) \cdot (g \cdot a_3(d) \cdot Q||P))) \\
=& \tau_I(\partial_H(b_1(d)) \cdot \partial_H(f \cdot a_2(d) \cdot P||Q) + \partial_H(b_2(d)) \cdot \partial_H(g \cdot a_3(d) \cdot Q||P)) \\
=& \tau_I(b_1(d) \cdot \partial_H(f \cdot a_2(d) \cdot P||Q) + \delta \cdot \partial_H(g \cdot a_3(d) \cdot Q||P)) \\
=& \tau_I(b_1(d) \cdot \partial_H(f \cdot a_2(d) \cdot P||Q) + \delta) \\
=& \tau_I(b_1(d)) \cdot \tau_I(\partial_H(f \cdot a_2(d) \cdot P||Q)) \\
=& b_1(d) \cdot \tau_I(\partial_H(f \cdot a_2(d) \cdot P||Q))
\end{aligned}
\tag{C.2}
$$

Si se ejecuta la acción $b_1(d)$ se tiene:

$$
R_1 = \tau_I(\partial_H(f \cdot a_2(d) \cdot P||Q)) = f \cdot \tau_I(\partial_H(a_2(d) \cdot P||Q)) \tag{C.3}
$$

Ejecutando f se tiene:

$$
R_2 = \tau_I(\partial_H(a_2(d) \cdot P||Q)) = \tau_I(c_2(d)) \cdot \tau_I(\partial_H(P||g \cdot a_3(d) \cdot Q)) \tag{C.4}
$$

En el desarrollo C.4 se realiza la comunicación entre procesos, sin embargo desde el punto de vista del comportamiento del sistema esto ocurre de forma interna, por ello el término $\tau_I(c_2(d))$ desaparece. Este paso es el equivalente a una transición ϵ de un autómata no determinista, por lo tanto, el resultado cambia:

$$
R_2 = \tau_I(\partial_H(P||b \cdot a_3(d) \cdot Q)) = g \cdot \tau_I(\partial_H(P||a_3(d) \cdot Q)) \tag{C.5}
$$

Ejecutando g se tiene:

$$
R_3 = \tau_I(\partial_H(P||a_3(d) \cdot Q)) = b_1(d) \cdot \tau_I(\partial_H(f \cdot a_2(d) \cdot P||a_3(d) \cdot Q)) + a_3(d) \cdot \tau_I(\partial_H(P||Q)) \tag{C.6}
$$

En el desarrollo C.6 se tienen dos alternativas para seguir con la ejecución: Si se ejecuta $b_1(d)$ se tiene:

$$R_4 = \tau_I(\partial_H(f \cdot a_2(d) \cdot P || a_3(d) \cdot Q)) = f \cdot \tau_I(\partial_H(a_2(d) \cdot P || a_3(d) \cdot Q)) + a_3(d) \cdot \tau_I(\partial_H(Q || a \cdot a_2(d) \cdot P)) \tag{C.7}$$

Ejecutando f se tiene:

$$R_5 = \tau_I(\partial_H(a_2(d) \cdot P || a_3(d) \cdot Q)) = a_3(d) \cdot \tau_I(\partial_H(Q || a_2(d) \cdot P)) \tag{C.8}$$

Ejecutando $a_3(d)$ del paso C.7 el sistema pasa al estado resultante del desarrollo C.3.

$$\begin{aligned} R_6 &= \tau_I(\partial_H(Q || f \cdot a_2(d) \cdot P)) \\ R_6 &= \tau_I(\partial_H(f \cdot a_2(d) \cdot P || Q)) = R_1 \end{aligned} \tag{C.9}$$

La otra alternativa es ejecutar $a_3(d)$ del paso C.6, es decir si se envía un dato, el sistema pasa al estado inicial(C.2).

$$R_7 = \tau_I(\partial_H(P || Q)) = R \tag{C.10}$$

C.3. Problema 2

C.3.1. Definición

$$
\begin{aligned}
f &\triangleq \text{Función de transición local} \\
\Delta &\triangleq \{d \mid d \quad \text{Puede ser transmitido por los procesos P1,P2,P3 y P4}\} \\
b_i(d) &\triangleq \text{Recibe el dato d por el canal i} \quad (d \in \Delta,\ i \in \{1,2,3,4,5,6,7,8\}) \\
a_i(d) &\triangleq \text{Envía el dato d por el canal i} \quad (d \in \Delta,\ i \in \{1,2,3,4,5,6,7,8\}) \\
c_i(d) &\triangleq \gamma(a_i(d), b_i(d)) \quad (d \in \Delta,\ i \in \{1,2,3,4,5,6,7,8\}) \\
H &\triangleq \{b_i(d), a_i(d)\} \\
P1 &\triangleq (a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) + a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d)) \cdot f \cdot P1 \\
C1 &\triangleq (b_1(d) \cdot a_2(d) + b_2(d) \cdot a_1(d)) \cdot C1 \\
P2 &\triangleq (a_3(d) \cdot b_2(d) \cdot a_2(d) \cdot b_3(d) + a_2(d) \cdot b_3(d) \cdot a_3(d) \cdot b_2(d)) \cdot f \cdot P2 \\
C2 &\triangleq (b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2 \\
P3 &\triangleq (a_5(d) \cdot b_4(d) \cdot a_4(d) \cdot b_5(d) + a_4(d) \cdot b_5(d) \cdot a_5(d) \cdot b_4(d)) \cdot f \cdot P3 \\
C3 &\triangleq (b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3 \\
P4 &\triangleq (a_7(d) \cdot b_6(d) \cdot a_6(d) \cdot b_7(d) + a_6(d) \cdot b_7(d) \cdot a_7(d) \cdot b_6(d)) \cdot f \cdot P4 \\
C4 &\triangleq (b_7(d) \cdot a_8(d) + b_8(d) \cdot a_7(d)) \cdot C4 \\
R &\triangleq \partial_H(P1||C1||P2||C2||P3||C3||P4||C4)
\end{aligned}
\tag{C.11}
$$

C.3.2. Verificación

Para comenzar con el proceso de derivación se puede elegir cualquier par de procesos, por ejemplo, $C2||P3$, $P1||C4$ o inclusive $P2||P4$, sin embargo se puede observar que el primer elemento a ejecutar de los procesos es un elemento de comunicación, por lo que cualquier par de procesos que no comiencen con el par de elementos $a_i(d), b_i(d)$ caerán en deadlock, como se verá a continuación (ya que las acciones $a_i(d)$ y $b_i(d)$ forman parte del conjunto H).

Una vez elegido el par de procesos se realiza la derivación hasta lograr ejecutar una acción atómica o que el sistema caiga en deadlock. Este proceso se repite con los elementos restantes (por pares) y al final se unen los resultados y las acciones atómicas que se ejecutaron forman parte de la traza del sistema.

Agrupando los términos de R por pares.

$$R \triangleq \partial_H(P1||C1)|||(P2||C2)|||(P3||C3)||(P4||C4) \tag{C.12}$$

Derivando $(P1||C1)$

$$\begin{aligned}
P1||C1 =&((a_1(d)\cdot b_8(d)\cdot a_8(d)\cdot b_1(d)+a_8(d)\cdot b_1(d)\cdot a_1(d)\cdot b_8(d))\cdot f\cdot P1)\\
&||((b_1(d)\cdot a_2(d)+b_2(d)\cdot a_1(d))\cdot C1)\\
P1||C1 =&(a_1(d)\cdot b_8(d)\cdot a_8(d)\cdot b_1(d)\cdot f\cdot P1+a_8(d)\cdot b_1(d)\cdot a_1(d)\cdot b_8(d)\cdot f\cdot P1)\\
&||(b_1(d)\cdot a_2(d)\cdot C1+b_2(d)\cdot a_1(d)\cdot C1)\\
P1||C1 =&(a_1(d)\cdot b_8(d)\cdot a_8(d)\cdot b_1(d)\cdot f\cdot P1+a_8(d)\cdot b_1(d)\cdot a_1(d)\cdot b_8(d)\cdot f\cdot P1)\\
&\lfloor\lfloor(b_1(d)\cdot a_2(d)\cdot C1+b_2(d)\cdot a_1(d)\cdot C1)\\
&+(b_1(d)\cdot a_2(d)\cdot C1+b_2(d)\cdot a_1(d)\cdot C1)\rfloor\rfloor\\
&(a_1(d)\cdot b_8(d)\cdot a_8(d)\cdot b_1(d)\cdot f\cdot P1+a_8(d)\cdot b_1(d)\cdot a_1(d)\cdot b_8(d)\cdot f\cdot P1)\\
&+(a_1(d)\cdot b_8(d)\cdot a_8(d)\cdot b_1(d)\cdot f\cdot P1+a_8(d)\cdot b_1(d)\cdot a_1(d)\cdot b_8(d)\cdot f\cdot P1)\\
&|(b_1(d)\cdot a_2(d)\cdot C1+b_2(d)\cdot a_1(d)\cdot C1)
\end{aligned} \tag{C.13}$$

$$
\begin{aligned}
P1||C1 =& a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 \lfloor\lfloor b_1(d) \cdot a_2(d) \cdot C1 \\
&+ a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d) \cdot f \cdot P1 \lfloor\lfloor b_2(d) \cdot a_1(d) \cdot C1 \\
&+ a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 \lfloor\lfloor b_2(d) \cdot a_1(d) \cdot C1 \\
&+ a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d) \cdot f \cdot P1 \lfloor\lfloor b_1(d) \cdot a_2(d) \cdot C1 \\
&+ b_1(d) \cdot a_2(d) \cdot C1 \lfloor\lfloor a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 \\
&+ b_2(d) \cdot a_1(d) \cdot C1 \lfloor\lfloor a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d) \cdot f \cdot P1 \\
&+ b_2(d) \cdot a_1(d) \cdot C1 \lfloor\lfloor a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 \\
&+ b_1(d) \cdot a_2(d) \cdot C1 \lfloor\lfloor a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d) \cdot f \cdot P1 \\
&+ a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 | b_1(d) \cdot a_2(d) \cdot C1 \\
&+ a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d) \cdot f \cdot P1 | b_2(d) \cdot a_1(d) \cdot C1 \\
&+ a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 | b_2(d) \cdot a_1(d) \cdot C1 \\
&+ a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d) \cdot f \cdot P1 | b_1(d) \cdot a_2(d) \cdot C1
\end{aligned}
\tag{C.14}
$$

En el desarrollo C.49 se puede ver que la primera acción atómica de cada elemento de la izquierda en los left merge ($\lfloor\lfloor$) es un elemento de comunicación, ya sea $a_i(d)$ o $b_i(d)$, por lo tanto todos esos términos van a deadlock.

$$
\begin{aligned}
P1||C1 =& a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 | b_1(d) \cdot a_2(d) \cdot C1 \\
&+ a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d) \cdot f \cdot P1 | b_2(d) \cdot a_1(d) \cdot C1 \\
&+ a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 | b_2(d) \cdot a_1(d) \cdot C1 \\
&+ a_8(d) \cdot b_1(d) \cdot a_1(d) \cdot b_8(d) \cdot f \cdot P1 | b_1(d) \cdot a_2(d) \cdot C1
\end{aligned}
\tag{C.15}
$$

En C.52 se tienen 4 acciones de comunicación que son posibles solo si el primer elemento de cada término del | son los componentes correctos, es decir, tienen el mismo subíndice, por lo tanto solo sobrevive uno. Cabe destacar que el subindice del término resultante es el canal que los conecta.

$$
\begin{aligned}
P1||C1 =& a_1(d) \cdot b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 | b_1(d) \cdot a_2(d) \cdot C1 \\
P1||C1 =& c_1(d) \cdot (b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 || a_2(d) \cdot C1)
\end{aligned}
\tag{C.16}
$$

Sustituyendo en R.

$$\begin{aligned} R =& c_1(d) \cdot (\partial_H(b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 || a_2(d) \cdot C1) || (P2 || C2) \\ & || (P3 || C3) || (P4 || C4)) \end{aligned} \tag{C.17}$$

Ejecutando $c_1(d)$.

$$\begin{aligned} R_1 =& \partial_H(b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 || a_2(d) \cdot C1 || (P2 || C2) \\ & || (P3 || C3) || (P4 || C4) \end{aligned} \tag{C.18}$$

A continuación se toma el par de procesos $(P2||C2)$ y se deriva.

$$\begin{aligned} (P2||C2) =& (a_3(d) \cdot b_2(d) \cdot a_2(d) \cdot b_3(d) + a_2(d) \cdot b_3(d) \cdot a_3(d) \cdot b_2(d)) \cdot f \cdot P2 \\ & || (b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2 \end{aligned} \tag{C.19}$$

Como en el caso anterior, solo sobrevivirá el elemento de comunicación correspondiente al canal que une a ambos procesos.

$$(P2||C2) = c_3(d) \cdot (b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2 || a_4(d) \cdot C2) \tag{C.20}$$

Sustituyendo en R_1.

$$\begin{aligned} R_1 =& c_3(d) \cdot (\partial_H((b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 || a_2(d) \cdot C1) \\ & || b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2 || a_4(d) \cdot C2 \\ & || (P3 || C3) || (P4 || C4)) \end{aligned} \tag{C.21}$$

Ejecutando $c_3(d)$.

$$\begin{aligned} R_2 =& \partial_H(b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1 || a_2(d) \cdot C1 \\ & || b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2 || a_4(d) \cdot C2 \\ & || (P3 || C3) || (P4 || C4)) \end{aligned} \tag{C.22}$$

A continuación se toma el par de procesos $(P3||C3)$ y se deriva resultando un comportamiento similar a los otros pares.

$$(P3||C3) = c_5(d) \cdot (b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_6(d) \cdot C3) \tag{C.23}$$

Sustituyendo en R_2.

$$\begin{aligned} R_2 =& c_5(d) \cdot (\partial_H((b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_2(d) \cdot C1) \\ & ||b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_4(d) \cdot C2 \\ & ||b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_6(d) \cdot C3 \\ & ||(P4||C4)) \end{aligned} \tag{C.24}$$

Ejecutando $c_5(d)$.

$$\begin{aligned} R_3 =& \partial_H((b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_2(d) \cdot C1) \\ & ||b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_4(d) \cdot C2 \\ & ||b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_6(d) \cdot C3 \\ & ||(P4||C4)) \end{aligned} \tag{C.25}$$

A continuación se toma el par de procesos $(P4||C4)$ y se deriva resultando un comportamiento similar a los otros pares.

$$(P4||C4) = c_7(d) \cdot (b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_8(d) \cdot C4) \tag{C.26}$$

Sustituyendo en R_3

$$\begin{aligned} R_3 =& c_7(d) \cdot (\partial_H((b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_2(d) \cdot C1) \\ & ||b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_4(d) \cdot C2 \\ & ||b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_6(d) \cdot C3 \\ & ||b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_8(d) \cdot C4)) \end{aligned} \tag{C.27}$$

Ejecutando $c_7(d)$.

$$\begin{aligned} R_4 =& \partial_H(b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_2(d) \cdot C1||b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_4(d) \cdot C2 \\ & ||b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_6(d) \cdot C3||b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_8(d) \cdot C4) \end{aligned} \tag{C.28}$$

Una vez mas se agruparán los términos por pares de forma que se puedan realizar eventos de comunicación, el resultado es la ecuación C.29.

$$\begin{aligned} R_4 = \partial_H((b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_8(d) \cdot C4)||(b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_2(d) \cdot C1) \\ ||(b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_4(d) \cdot C2)||(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3)) \end{aligned} \tag{C.29}$$

A continuación se toma el par $b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_8(d) \cdot C4$.

$$(b_8(d) \cdot a_8(d) \cdot b_1(d) \cdot f \cdot P1||a_8(d) \cdot C4) = c_8(d) \cdot (a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4) \tag{C.30}$$

Sustituyendo en R_4.

$$\begin{aligned} R_4 = c_8(d) \cdot (\partial_H((a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4)||(b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_2(d) \cdot C1) \\ ||(b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_4(d) \cdot C2)||(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3))) \end{aligned} \tag{C.31}$$

Ejecutando $c_8(d)$.

$$\begin{aligned} R_5 = \partial_H((a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4)||(b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_2(d) \cdot C1) \\ ||(b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_4(d) \cdot C2)||(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3)) \end{aligned} \tag{C.32}$$

A continuación se toma el par $(b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_2(d) \cdot C1)$.

$$(b_2(d) \cdot a_2(d) \cdot b_3(d) \cdot f \cdot P2||a_2(d) \cdot C1) = c_2(d) \cdot (a_2(d) \cdot b_3(d) \cdot f \cdot P2||C1) \tag{C.33}$$

Sustituyendo en R_5.

$$\begin{aligned} R_5 = c_2(d) \cdot (\partial_H((a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4)||(a_2(d) \cdot b_3(d) \cdot f \cdot P2||C1) \\ ||(b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_4(d) \cdot C2)||(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3))) \end{aligned} \tag{C.34}$$

Ejecutando $c_2(d)$.

$$\begin{aligned} R_6 = \partial_H((a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4)||(a_2(d) \cdot b_3(d) \cdot f \cdot P2||C1) \\ ||(b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_4(d) \cdot C2)||(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3)) \end{aligned} \tag{C.35}$$

A continuación se toma el par $(b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_4(d) \cdot C2)$.

$$(b_4(d) \cdot a_4(d) \cdot b_5(d) \cdot f \cdot P3||a_4(d) \cdot C2) = c_4(d) \cdot (a_4(d) \cdot b_5(d) \cdot f \cdot P3||C2) \tag{C.36}$$

Sustituyendo en R_6.

$$\begin{aligned} R_6 = c_4(d) \cdot (\partial_H((a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4)||(a_2(d) \cdot b_3(d) \cdot f \cdot P2||C1) \\ ||(a_4(d) \cdot b_5(d) \cdot f \cdot P3||C2)||(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3))) \end{aligned} \tag{C.37}$$

Ejecutando $c_4(d)$.

$$\begin{aligned} R_7 = \partial_H((a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4)||(a_2(d) \cdot b_3(d) \cdot f \cdot P2||C1) \\ ||(a_4(d) \cdot b_5(d) \cdot f \cdot P3||C2)||(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3)) \end{aligned} \tag{C.38}$$

A continuación se toma el par $b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3)$.

$$(b_6(d) \cdot a_6(d) \cdot b_7(d) \cdot f \cdot P4||a_6(d) \cdot C3) = c_6(d) \cdot (a_6(d) \cdot b_7(d) \cdot f \cdot P4||C3) \tag{C.39}$$

Sustituyendo en R_7.

$$\begin{aligned} R_7 = c_6(d) \cdot (\partial_H((a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4)||(a_2(d) \cdot b_3(d) \cdot f \cdot P2||C1) \\ ||(a_4(d) \cdot b_5(d) \cdot f \cdot P3||C2)||(a_6(d) \cdot b_7(d) \cdot f \cdot P4||C3))) \end{aligned} \tag{C.40}$$

Ejecutando $c_6(d)$.

$$\begin{aligned} R_8 = \partial_H(a_8(d) \cdot b_1(d) \cdot f \cdot P1||C4||a_2(d) \cdot b_3(d) \cdot f \cdot P2||C1 \\ ||a_4(d) \cdot b_5(d) \cdot f \cdot P3||C2||a_6(d) \cdot b_7(d) \cdot f \cdot P4||C3) \end{aligned} \tag{C.41}$$

Para el siguiente paso se sustituye $C1, C2, C3$ y $C4$ por sus definiciones (ecuación C.42).

$$\begin{aligned} R_8 = \partial_H(a_8(d) \cdot b_1(d) \cdot f \cdot P1||(b_7(d) \cdot a_8(d) + b_8(d) \cdot a_7(d)) \cdot C4 \\ ||a_2(d) \cdot b_3(d) \cdot f \cdot P2||(b_1(d) \cdot a_2(d) + b_2(d) \cdot a_1(d)) \cdot C1 \\ ||a_4(d) \cdot b_5(d) \cdot f \cdot P3||(b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2 \\ ||a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3) \end{aligned} \tag{C.42}$$

Se colocan por pares.

$$\begin{aligned} R_8 = \partial_H((a_8(d) \cdot b_1(d) \cdot f \cdot P1||(b_7(d) \cdot a_8(d) + b_8(d) \cdot a_7(d)) \cdot C4) \\ ||(a_2(d) \cdot b_3(d) \cdot f \cdot P2||(b_1(d) \cdot a_2(d) + b_2(d) \cdot a_1(d)) \cdot C1) \\ ||(a_4(d) \cdot b_5(d) \cdot f \cdot P3||(b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2) \\ ||(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3)) \end{aligned} \tag{C.43}$$

A continuación se toma el par $a_8(d) \cdot b_1(d) \cdot f \cdot P1||(b_7(d) \cdot a_8(d) + b_8(d) \cdot a_7(d)) \cdot C4$.

$$a_8(d) \cdot b_1(d) \cdot f \cdot P1||(b_7(d) \cdot a_8(d) + b_8(d) \cdot a_7(d)) \cdot C4 = c_8(d) \cdot (b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4) \tag{C.44}$$

Sustituyendo en R_8.

$$\begin{aligned} R_8 = c_8(d) \cdot (&\partial_H((b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4) \\ &||(a_2(d) \cdot b_3(d) \cdot f \cdot P2||(b_1(d) \cdot a_2(d) + b_2(d) \cdot a_1(d)) \cdot C1) \\ &||(a_4(d) \cdot b_5(d) \cdot f \cdot P3||(b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2) \\ &||(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3))) \end{aligned} \tag{C.45}$$

Ejecutando $c_8(d)$.

$$\begin{aligned} R_9 = \partial_H((&b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4) \\ &||(a_2(d) \cdot b_3(d) \cdot f \cdot P2||(b_1(d) \cdot a_2(d) + b_2(d) \cdot a_1(d)) \cdot C1) \\ &||(a_4(d) \cdot b_5(d) \cdot f \cdot P3||(b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2) \\ &||(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3)) \end{aligned} \tag{C.46}$$

A continuación se toma el par $(a_2(d) \cdot b_3(d) \cdot f \cdot P2||(b_1(d) \cdot a_2(d) + b_2(d) \cdot a_1(d)) \cdot C1)$.

$$a_2(d) \cdot b_3(d) \cdot f \cdot P2||(b_1(d) \cdot a_2(d) + b_2(d) \cdot a_1(d)) \cdot C1 = c_2(d) \cdot (b_3(d) \cdot f \cdot P2||a_1(d) \cdot C1) \tag{C.47}$$

Sustituyendo en R_9.

$$\begin{aligned} R_9 = c_2(d) \cdot (&\partial_H((b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4) \\ &||(b_3(d) \cdot f \cdot P2||a_1(d) \cdot C1) \\ &||(a_4(d) \cdot b_5(d) \cdot f \cdot P3||(b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2) \\ &||(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3))) \end{aligned} \tag{C.48}$$

Ejecutando $c_2(d)$.

$$\begin{aligned} R_{10} = \partial_H((&b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4) \\ &||(b_3(d) \cdot f \cdot P2||a_1(d) \cdot C1) \\ &||(a_4(d) \cdot b_5(d) \cdot f \cdot P3||(b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2) \\ &||(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3)) \end{aligned} \tag{C.49}$$

A continuación se toma el par $(a_4(d) \cdot b_5(d) \cdot f \cdot P3||(b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2)$.

$$a_4(d) \cdot b_5(d) \cdot f \cdot P3||(b_3(d) \cdot a_4(d) + b_4(d) \cdot a_3(d)) \cdot C2 = c_4(d) \cdot (b_5(d) \cdot f \cdot P3||a_3(d) \cdot C2) \tag{C.50}$$

Sustituyendo en R_{10}.

$$\begin{aligned} R_{10} = c_4(d) \cdot (\partial_H((b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4) \\ ||(b_3(d) \cdot f \cdot P2||a_1(d) \cdot C1) \\ ||(b_5(d) \cdot f \cdot P3||a_3(d) \cdot C2) \\ ||(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3))) \end{aligned} \tag{C.51}$$

Ejecutando $c_4(d)$.

$$\begin{aligned} R_{11} = \partial_H((b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4) \\ ||(b_3(d) \cdot f \cdot P2||a_1(d) \cdot C1) \\ ||(b_5(d) \cdot f \cdot P3||a_3(d) \cdot C2) \\ ||(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3)) \end{aligned} \tag{C.52}$$

A continuación se toma el par $(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3)$.

$$(a_6(d) \cdot b_7(d) \cdot f \cdot P4||(b_5(d) \cdot a_6(d) + b_6(d) \cdot a_5(d)) \cdot C3) = c_6(d) \cdot (b_7(d) \cdot f \cdot P4||a_5(d) \cdot C3) \tag{C.53}$$

Sustituyendo en R_{11}.

$$\begin{aligned} R_{11} = c_6(d) \cdot (\partial_H((b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4) \\ ||(b_3(d) \cdot f \cdot P2||a_1(d) \cdot C1) \\ ||(b_5(d) \cdot f \cdot P3||a_3(d) \cdot C2) \\ ||(b_7(d) \cdot f \cdot P4||a_5(d) \cdot C3))) \end{aligned} \tag{C.54}$$

Ejecutando $c_6(d)$

$$\begin{aligned} R_{12} = \partial_H(b_1(d) \cdot f \cdot P1||a_7(d) \cdot C4 \\ ||b_3(d) \cdot f \cdot P2||a_1(d) \cdot C1 \\ ||b_5(d) \cdot f \cdot P3||a_3(d) \cdot C2 \\ ||b_7(d) \cdot f \cdot P4||a_5(d) \cdot C3) \end{aligned} \tag{C.55}$$

Se agrupa por pares.

$$R_{12} = \partial_H((b_1(d) \cdot f \cdot P1||a_1(d) \cdot C1)||(b_3(d) \cdot f \cdot P2||a_3(d) \cdot C2)|| \\ ||(b_5(d) \cdot f \cdot P3||a_5(d) \cdot C3)||(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4)) \tag{C.56}$$

A continuación se toma el par $(b_1(d) \cdot f \cdot P1||a_1(d) \cdot C1)$.

$$(b_1(d) \cdot f \cdot P1||a_1(d) \cdot C1) = c_1(d) \cdot (f \cdot P1||C1) \tag{C.57}$$

Sustituyendo en R_{12}.

$$R_{12} = c_1(d) \cdot (\partial_H((f \cdot P1||C1)||(b_3(d) \cdot f \cdot P2||a_3(d) \cdot C2) \\ ||(b_5(d) \cdot f \cdot P3||a_5(d) \cdot C3)||(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4))) \tag{C.58}$$

Ejecutando $c_1(d)$.

$$R_{13} = \partial_H((f \cdot P1||C1)||(b_3(d) \cdot f \cdot P2||a_3(d) \cdot C2) \\ ||(b_5(d) \cdot f \cdot P3||a_5(d) \cdot C3)||(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4)) \tag{C.59}$$

A continuación se toma el par $(b_3(d) \cdot f \cdot P2||a_3(d) \cdot C2)$.

$$(b_3(d) \cdot f \cdot P2||a_3(d) \cdot C2) = c_3(d) \cdot (f \cdot P2||C2) \tag{C.60}$$

Sustituyendo en R_{13}.

$$R_{13} = c_3(d) \cdot (\partial_H((f \cdot P1||C1)||(f \cdot P2||C2) \\ ||(b_5(d) \cdot f \cdot P3||a_5(d) \cdot C3)||(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4))) \tag{C.61}$$

Ejecutando $c_3(d)$.

$$R_{14} = \partial_H((f \cdot P1||C1)||(f \cdot P2||C2)|| \\ ||(b_5(d) \cdot f \cdot P3||a_5(d) \cdot C3)||(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4)) \tag{C.62}$$

A continuación se toma el par $(b_5(d) \cdot f \cdot P3||a_5(d) \cdot C3)$.

$$(b_5(d) \cdot f \cdot P3||a_5(d) \cdot C3) = c_5(d) \cdot (f \cdot P3||C3) \tag{C.63}$$

Sustituyendo en R_14

$$R_{14} = c_5(d) \cdot (\partial_H((f \cdot P1||C1)||(f \cdot P2||C2)||(f \cdot P3||C3) \\ ||(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4))) \tag{C.64}$$

Ejecutando $c_5(d)$.

$$R_{15} = \partial_H((f \cdot P1||C1)||(f \cdot P2||C2)||(f \cdot P3||C3) \\ ||(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4)) \tag{C.65}$$

A continuación se toma el par $(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4)$.

$$(b_7(d) \cdot f \cdot P4||a_7(d) \cdot C4) = c_7(d) \cdot (f \cdot P4||C4) \tag{C.66}$$

Sustituyendo en R_15.

$$R_{15} = c_7(d) \cdot (\partial_H((f \cdot P1||C1)||(f \cdot P2||C2)||(f \cdot P3||C3)||(f \cdot P4||C4))) \tag{C.67}$$

Ejecutando $c_7(d)$.

$$R_{16} = \partial_H((f \cdot P1||C1)||(f \cdot P2||C2)||(f \cdot P3||C3)||(f \cdot P4||C4)) \tag{C.68}$$

Agrupando por pares

$$R_{16} = \partial_H((f \cdot P1||C1)||(f \cdot P2||C2)||(f \cdot P3||C3)||(f \cdot P4||C4)) \tag{C.69}$$

A continuación se toma el par $(f \cdot P1||C1)$.

$$f \cdot P1||C1 = f \cdot (P1||C1) \tag{C.70}$$

Sustituyendo en r_{16}.

$$R_{16} = f \cdot (\partial_H((P1||C1)||(f \cdot P2||C2)||(f \cdot P3||C3)||(f \cdot P4||C4))) \tag{C.71}$$

Ejecutando f.

$$R_{17} = \partial_H((P1||C1)||(f \cdot P2||C2)||(f \cdot P3||C3)||(f \cdot P4||C4)) \tag{C.72}$$

A continuación se toma el par $(f \cdot P2||C2)$.

$$f \cdot P2||C2 = f \cdot (P2||C2) \tag{C.73}$$

Sustituyendo en R_{17}.

$$R_{17} = f \cdot (\partial_H((P1||C1)||(P2||C2)||(f \cdot P3||C3)||(f \cdot P4||C4))) \tag{C.74}$$

Ejecutando f.

$$R_{18} = \partial_H((P1||C1)||(P2||C2)||(f \cdot P3||C3)||(f \cdot P4||C4)) \tag{C.75}$$

A continuación se toma el par $(f \cdot P3||C3)$.

$$f \cdot P3||C3 = f \cdot (P3||C3) \tag{C.76}$$

Sustituyendo en R_{18}.

$$R_{18} = f \cdot (\partial_H((P1||C1)||(P2||C2)||(P3||C3)||(f \cdot P4||C4))) \tag{C.77}$$

Ejecutando f.

$$R_{19} = \partial_H((P1||C1)||(P2||C2)||(P3||C3)||(f \cdot P4||C4)) \tag{C.78}$$

A continuación se toma el par $(f \cdot P4||C4)$.

$$(f \cdot P4||C4) = f \cdot (P4||C4) \tag{C.79}$$

Sustituyendo en R_{19}.

$$R_{19} = f \cdot \partial_H((P1||C1)||(P2||C2)||(P3||C3)||(P4||C4))) \tag{C.80}$$

Ejecutando f.

$$R20 = \partial_H(P1||C1||P2||C2||P3||C3||P4||C4) = R \tag{C.81}$$

Bibliografía

[AFV99] Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In Handbook of Process Algebra, pages 197–292. Elsevier, 1999.

[Bae05] J.C.M. Baeten. Abrief history of process algebra. Theoretical Computer Science, page 131 – 146, 2005.

[BDSW14] Pontus Boström, Fredrik Degerlund, Kaisa Sere, and Marina Waldén. Derivation of concurrent programs by stepwise scheduling of event-b models. Formal Aspects of Computing, 26(2):281–303, 2014.

[BK89] J.A. Bergstra and J.W. Klop. Acpτ a universal axiom system for process specification. In Martin Wirsing and JanA. Bergstra, editors, Algebraic Methods: Theory, Tools and Applications, volume 394 of Lecture Notes in Computer Science, pages 445–463. Springer Berlin Heidelberg, 1989.

[BKS89] R.-J.R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. Distributed Computing, 3(2):73–87, 1989.

[Bon10] Maria Paola Bonacina. On theorem proving for program checking: Historical perspective and recent developments. In Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, PPDP '10, pages 1–12, New York, NY, USA, 2010. ACM.

[BS91] Ralph-Johan Back and Kaisa Sere. Stepwise refinement of action systems. Structured Programming, 12:17–30, 1991.

[But96] Michael J. Butler. Stepwise refinement of communicating systems. Science of Computer

Programming, 27(2):139 – 173, 1996.

[CD99] Jens Coldewey and Paul Dyson, editors. Proceedings of the 3rd European Conference on Pattern Languages of Programms (EuroPLoP '1998), Irsee, Germany, July 8-12, 1998. UVK - Universitaetsverlag Konstanz, 1999.

[CK89] C. C. Chang and H. J. Keisler. Model Theory. North Holland, Amsterdam, third edition edition, 1989.

[Dij75] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM, 18(8):453–457, August 1975.

[Dij97] Edsger Wybe Dijkstra. A Discipline of Programming. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1997.

[DM98] Leonardo Dagum and Ramesh Menon. Openmp: An industry-standard api for shared-memory programming. IEEE Comput. Sci. Eng., 5(1):46–55, January 1998.

[DS08] Fredrik Degerlund and Kaisa Sere. Refinement of Parallel Algorithms, pages 77–96. Computational Science Series. Chapman and Hall/CRC Press (Taylor and Francis Group), 2008.

[DYK05] Zhenhua Duan, Xiaoxiao Yang, and Maciej Koutny. Semantics of framed temporal logic programs. In Logic Programming, pages 356–370. Springer, 2005.

[Fok99] Wan Fokkink. Introduction to Process Algebra. Springer, Germany, 1999.

[FR96] Michael J. Flynn and Kevin W. Rudd. Parallel architectures. ACM Comput. Surv., 28(1):67–70, March 1996.

[Gri04] V.N. Grishin. Springer, encyclopedia of mathematics. `http://www.encyclopediaofmath.org/index.php/Equivalence_relation`, 2004. Consultado el 12-Ene-2015.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. Commun. ACM, 21(8):666–677, August 1978.

[KA99] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating di-

rected task graphs to multiprocessors. ACM Comput. Surv., 31(4):406–471, December 1999.

[Kar05] Jarkko Kari. Reversible cellular automata. In Clelia De Felice and Antonio Restivo, editors, Developments in Language Theory, volume 3572 of Lecture Notes in Computer Science, pages 57–68. Springer Berlin Heidelberg, 2005.

[Koz97] Dexter C. Kozen. Automata and Computability. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1997.

[LDM+12] Yun Liang, Huping Ding, Tulika Mitra, Abhik Roychoudhury, Yan Li, and Vivy Suhendra. Timing analysis of concurrent programs running on shared cache multi-cores. Real-Time Systems, 48(6):638–680, 2012.

[Mil82] R. Milner. A Calculus of Communicating Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[Mol90] Faron Moller. The importance of the left merge operator in process algebras. In MichaelS. Paterson, editor, Automata, Languages and Programming, volume 443 of Lecture Notes in Computer Science, pages 752–764. Springer Berlin Heidelberg, 1990.

[NW05] C.F. Ngolah and Yingxu Wang. An operational semantics for rtpa. In Electrical and Computer Engineering, 2005. Canadian Conference on, pages 1823–1826, May 2005.

[OA10] Jorge Luis Ortega-Arjona. Patterns for Parallel Software Design. Wiley Publishing, 1st edition, 2010.

[PB90] Cherri M. Pancake and D. Bergmark. Do parallel languages respond to the needs of scientific programmers? Computer, 23(12):13–23, Dec 1990.

[RS94] K. Ramamritham and J.A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. Proceedings of the IEEE, 82(1):55–67, Jan 1994.

[SOHL+98] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. MPI-The Complete Reference, Volume 1: The MPI Core. MIT Press, Cambridge, MA, USA, 2nd. (revised) edition, 1998.

[SSOG93] Jaspal Subhlok, James M. Stichnoth, David R. O'Hallaron, and Thomas Gross. Exploi-

ting task and data parallelism on a multicomputer. In Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP '93, pages 13–22, New York, NY, USA, 1993. ACM.

[Sut05] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobb's Journal, 30(3), March 2005.

[Tay83] RichardN. Taylor. Complexity of analyzing the synchronization structure of concurrent programs. Acta Informatica, 19(1):57–84, 1983.

[Wan02] Yingxu Wang. The real-time process algebra (rtpa). Annals of Software Engineering, 14(1-4):235–274, 2002.

[YCW+14] Junfeng Yang, Heming Cui, Jingyue Wu, Yang Tang, and Gang Hu. Making parallel programs reliable with stable multithreading. Commun. ACM, 57(3):58–69, March 2014.

[ZDT13] Nan Zhang, Zhenhua Duan, and Cong Tian. A cylinder computation model for many-core parallel computing. Theor. Comput. Sci., 497:68–83, July 2013.

[ZDT14] Nan Zhang, Zhenhua Duan, and Cong Tian. An axiomatization for cylinder computation model. In Zhipeng Cai, Alex Zelikovsky, and Anu Bourgeois, editors, Computing and Combinatorics, volume 8591 of Lecture Notes in Computer Science, pages 71–83. Springer International Publishing, 2014.