Sobre la especificación y verificación del patrón de programación paralela PCR en TLA+

On the specification and verification of the PCR parallel programming pattern in TLA+

Sobre a especificação e verificação do padrão de programação paralela PCR em TLA+

José E. Solsona [1]

Recibido: 26/05/2023 Aceptado: 26/05/2023

Resumen. - Limitaciones físicas en el diseño de microprocesadores han hecho que la industria de computadoras pase de mejorar la velocidad de un solo procesador a aumentar el número de unidades centrales de procesamiento. Pero el diseño de software para explotar la potencia de procesamiento paralelo de manera correcta y efectiva es una tarea desafiante que requiere un alto grado de experiencia. En 2017, Pérez y Yovine propusieron un patrón de programación paralela agnóstico de plataforma llamado PCR, que facilita la escritura de código paralelo. En este trabajo, formalizamos el patrón PCR en términos de TLA+ - un lenguaje de especificación formal para sistemas concurrentes que se está utilizando en lugares como Intel, Amazon y Microsoft. Buscamos proporcionar un marco formal principalmente para (1) expresar diseños PCR de alto nivel y probar su corrección funcional en el sentido de que su computación paralela calcula una función matemática dada, y (2) poder relacionar formalmente diferentes diseños PCR. De esta manera, contribuimos al estado del arte en la verificación formal de programas paralelos aprovechando las herramientas asociadas a TLA+ para probar propiedades sobre algoritmos PCR de alto nivel, como su corrección funcional y refinamiento.

Palabras clave: Algoritmos paralelos; Patrones de diseño; Métodos formales; TLA+

Summary. - *Physical limitations in processor design have made the computer industry shift from improving the speed of a single processor to increasing the number of processing core units. But the design of software to exploit parallel processing power in a correct and cost-effective way is a challenging task requiring a high degree of expertise. In 2017, Pérez and Yovine proposed a platform-agnostic parallel programming pattern called PCR, that eases writing parallel code. In this work, we formalize the PCR pattern in terms of TLA+ - a formal specification language for concurrent systems that is being used at places such as Intel, Amazon and Microsoft. We seek to provide a formal framework mainly for (1) expressing high level PCR designs and prove their functional correctness in the sense that their parallel computation computes a given mathematical function, and (2) being able to formally relate different PCR designs. In this way, we contribute to the state of the art in formal verification of parallel programs by leveraging TLA+-related tools to*

[1] Master en Ingeniería orientación Ciencias de la Computación. Universidad ORT Uruguay, solsona@ort.edu.uy, ORCID iD: https://orcid.org/0000-0001-8626-9569

proving properties about high-level PCR-based algorithms such as their functional correctness and refinement.

Keywords: *Parallel algorithms; Design patterns; Formal Methods; TLA+*

Resumo. **-** *As limitações físicas no design do processador fizeram com que a indústria de computadores mudasse de melhorar a velocidade de um único processador para aumentar o número de unidades centrais de processamento. Mas o design de software para explorar o poder de processamento paralelo de maneira correta e econômica é uma tarefa desafiadora que requer um alto grau de especialização. Em 2017, Pérez e Yovine propuseram um padrão de programação paralela independente de plataforma chamado PCR, que facilita a escrita de código paralelo. Neste trabalho, formalizamos o padrão PCR em termos de TLA+ - uma linguagem de especificação formal para sistemas concorrentes que está sendo utilizada em locais como Intel, Amazon e Microsoft. Procuramos fornecer uma estrutura formal principalmente para (1) expressar designs de PCR de alto nível e provar sua correção funcional no sentido de que sua computação paralela calcula uma determinada função matemática e (2) ser capaz de relacionar formalmente diferentes designs de PCR. Dessa forma, contribuímos para o estado da arte na verificação formal de programas paralelos, aproveitando as ferramentas relacionadas ao TLA+ para provar propriedades sobre algoritmos baseados em PCR de alto nível, como correção e refinamento funcional.*

Palavras-chave: *Algoritmos paralelos; Padrões de design; Métodos Formais; TLA+*.

1. Introducción. - Limitaciones físicas en el diseño de los procesadores han hecho que la industria de computadoras pase, desde 2005, de mejorar la velocidad de un solo procesador a aumentar el número de unidades de procesamiento [1]. Este cambio de paradigma plantea para la ingeniería de software el desafío de proporcionar las herramientas apropiadas para construir software de manera efectiva que explote de manera correcta y eficiente el poder de procesamiento paralelo. Además de solucionar los conocidos problemas de la programación concurrente, como *deadlocks*, *livelocks* y *data races*, que son la causa de numerosos errores, el desarrollo de software para hardware paralelo exige considerar diferentes modelos de ejecución, tener conocimiento de las características de la plataforma subyacente e integrar código heredado que no siempre se puede reescribir fácilmente. Esta complejidad dificulta la ingeniería de software paralelo correcto y eficiente, requiriendo un alto grado de especialización.

En [2], Pérez y Yovine argumentaron que un enfoque agnóstico de plataforma de alto nivel basado en patrones de diseño ayudaría a abordar esas dificultades. Sin embargo, también señalaron que la mayoría de los enfoques existentes para el diseño basado en patrones carecía de semántica formal, lo que socavaba la posibilidad de abordar adecuadamente el aspecto de la corrección. Por lo tanto, propusieron el patrón de programación paralela PCR, para el cual dieron una semántica basada en FXML, un lenguaje formal teórico que utiliza órdenes parciales para interpretar computaciones paralelas, y mostraron cómo podría implementarse en un modelo de ejecución concreto. Pudieron respaldar a través de evidencia empírica que era posible generar un código paralelo eficiente basado en el enfoque mencionado. En cuanto a la corrección, aunque demostraron que las transformaciones realizadas por una herramienta prototipo eran correctas, no se preocuparon por probar la corrección de los PCR en sí mismos. Además, FXML no tiene herramientas asociadas que puedan usarse para ayudar en la verificación formal.

En este trabajo, buscamos proporcionar un marco formal principalmente para (1) expresar diseños de PCR de alto nivel y probar su corrección funcional en el sentido de que su computación paralela calcula una función matemática dada, y (2) ser capaz de relacionar formalmente diferentes diseños de PCR. En particular, que un PCR con "más paralelismo" implemente otro PCR con "menos paralelismo", es decir, el primero es un refinamiento del segundo. Por razones prácticas, deseamos utilizar herramientas estándar para la verificación mecánica de las propiedades antes mencionadas, y creemos que es especialmente valioso que estas herramientas incluyan alguna forma de verificación automatizada (por ejemplo, *model-checking*) que permita una metodología rigurosa pero todavía liviana y apropiada para ser usada por los profesionales de la industria.

2. El patrón PCR: productor, consumidor y reductor.- Nuestra principal preocupación es el diseño de algoritmos paralelos correctos, con el fin de alcanzar los beneficios potenciales de la computación paralela sin sacrificar la corrección. Seguimos un enfoque basado en patrones de diseño, según el cual se deben desarrollar algoritmos paralelos sobre la base de algunos patrones de diseño específicos y bien entendidos susceptibles de análisis formal. El patrón PCR describe las computaciones concurrentes realizadas por productores, consumidores y reductores que se comunican entre sí, donde cada uno es una función básica, es decir, una función proporcionada por el usuario implementada en algún lenguaje anfitrión, o un PCR anidado. Aquí, presentamos una explicación informal e ilustrativa del patrón PCR.

2.1 Descripción de alto nivel.- El patrón PCR tiene como objetivo expresar computaciones en las que un productor recibe elementos de datos de entrada y genera, para cada uno de ellos, posiblemente muchas salidas que son consumidas por varios consumidores que trabajan en paralelo. Las salidas de todos ellos se agregan en un solo resultado por un reductor.

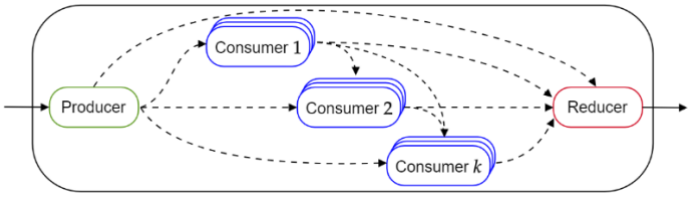

Figura I.- Vista pictórica del patrón de PCR.

El objetivo del patrón es enfatizar la independencia entre los diferentes cálculos, para exponer todas las oportunidades de paralelización. La Figura I representa la forma general de una PCR. Las flechas representan conexiones de datos en un PCR. Las flechas completas modelan la fuente de entrada externa y el canal de salida al entorno externo. La entrada externa está disponible para cualquier componente interno. Las flechas discontinuas indican canales de datos internos. Los ciclos de datos entre componentes internos no están permitidos: la red es en sí misma un grafo acíclico dirigido (DAG) del cual cualquier clasificación topológica tiene al productor y al reductor como el primer y último elemento, respectivamente. Las lecturas en los canales de datos no son destructivas; el mismo valor puede ser leído por cualquier consumidor y por el reductor.

El flujo de información dentro de un PCR es el siguiente: (1) Para cada elemento de datos de entrada, el componente productor genera un conjunto de valores de salida, cada uno de los cuales está inmediatamente disponible para su lectura. (2) Los componentes de consumo leen valores del alcance externo y de los canales de datos privados para realizar sus cálculos. (3) Un componente reductor combina valores de una o más fuentes de datos provenientes del productor y uno o más consumidores, generando un solo elemento de salida para cada elemento de entrada externo procesado por el productor.

Productor, consumidores y reductor trabajan en paralelo sujeto a las dependencias de datos existentes: todos los elementos de entrada deben estar disponibles para una instancia de productor, consumidor o reductor para realizar su cómputo. Cada productor, consumidor y reductor puede potencialmente generar tantas instancias de ejecución paralela como sea necesario para cualquier carga de trabajo específica. Se supone que tanto la naturaleza de una instancia de ejecución (proceso o subproceso local y/o remoto) como la política de programación están definidas por la implementación subyacente.

2.2 Ejemplo: contar los primeros números Fibonacci primos.- Consideremos el problema de contar los números primos entre los primeros N números de Fibonacci. La Figura II ilustra gráficamente cómo dos PCR pueden cooperar en una forma de composición para resolver este problema.

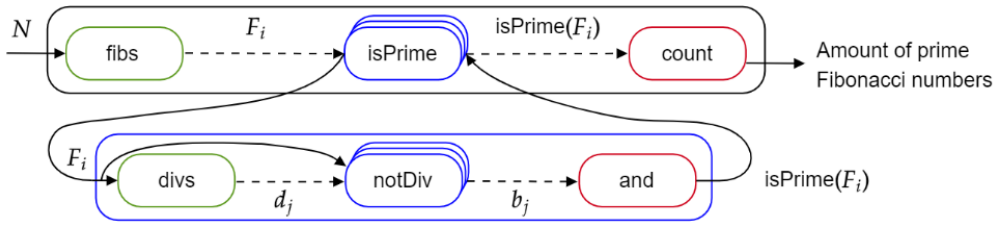

Figura II.- Solución PCR para contar números Fibonacci primos.

La solución PCR está diseñada para funcionar de la siguiente manera:

1. En el PCR externo, el productor `fibs` genera la secuencia $F_1, F_2, \ldots, F_N$ de los números de Fibonacci.
2. Para cada F_i, una instancia independiente del consumidor `isPrime` puede comprobar, en paralelo, su primalidad generando un resultado booleano. Esto constituye una oportunidad de anidamiento, en la que otro PCR interno puede paralelizar la subtarea de prueba de primalidad de la siguiente manera: (2.1) El productor `divs` genera todos los divisores posibles d_j de F_i. (2.2) Para cada d_j, una instancia independiente del consumidor `notDiv` puede verificar, en paralelo, la (no)divisibilidad de F_i por d_j generando un resultado booleano b_j. Este es un ejemplo de un consumidor que lee la salida del productor y también la entrada del PCR. (2.3) El reductor `and` calcula el resultado como la conjunción de todas las salidas b_j del consumidor.
3. El reductor `count` cuenta el número de salidas del consumidor que son verdaderas.

Esta solución admite la ejecución en paralelo en varios niveles. Muchas instancias de `isPrime` podrían ejecutarse simultáneamente según lo permitan las unidades de procesamiento disponibles y la tasa de producción de F_i. Además, cada instancia puede ser computada por un PCR interno con sus propias capacidades paralelas. Dado que las operaciones de reducción son asociativas (y conmutativas), también se pueden paralelizar.

3. Modelo abstracto para el patrón PCR.- Ahora presentamos un modelo abstracto de PCR, así como una sintaxis asociada. Este modelo debe ser fiel a la descripción informal, servir como una base rigurosa para razonar sobre PCR, pero lo suficientemente simple como para abstraerse de detalles como la arquitectura del hardware, *scheduling* de tareas o estrategias de implementación de reducción. Aquí nuestro objetivo es distinguir claramente, de la manera más abstracta y elemental posible, cómo computa un PCR (la concurrencia) de lo que computa el PCR (la entrada/salida esperada).

3.1 El PCR básico.- La computación de un PCR se rige por un espacio de iteración, un conjunto de índices (números naturales) posiblemente dependiente de la entrada sobre el que deberían actuar los componentes internos. Más precisamente, el espacio de iteración indexa los datos producidos y consumidos por los componentes del PCR (por ejemplo, para el ejemplo de Fibonacci es $\{1, 2, \ldots, N\}$). El resultado de cada operación de componente interno en cada índice se escribe en una variable de salida asociada. Específicamente, las variables de salida para el productor y los consumidores describen la historia completa de asignaciones en el espacio de iteración, modelando los canales de datos internos del PCR. Por lo tanto, pensamos en estas variables como flujos de valores matemáticamente representados por funciones parciales $\mathbb{N} \to T$, abreviado $\vec{T}$, para algún tipo de rango T. Si v es una variable de flujo, denotamos su i-ésimo elemento por v^i y por $wrt(v^i)$ la condición de que se ha escrito v^i. La escritura en una variable de flujo puede ocurrir en cualquier orden. La naturaleza de flujo de las variables se puede aprovechar en un mecanismo sintáctico que permite que las operaciones de flujo *look-ahead*/*look-behind* se utilicen en variables mediante la indexación. El siguiente es el esquema general para un PCR básico.

Sea A un PCR básico con k consumidores. La computación de A sobre la entrada x consiste en las siguientes operaciones (asignaciones) para cada $i \in I_x$:

$$\begin{aligned} \textbf{Productor:} \quad & p^i := f_p(x, p, i) \\ \textbf{Consumidor 1:} \quad & c_1^i := f_{c1}(x, p, i) \\ \vdots \quad & \quad \vdots \\ \textbf{Consumidor } \boldsymbol{k}\textbf{:} \quad & c_k^i := f_{ck}(x, p, c_1, \ldots, c_{k-1}, i) \\ \textbf{Reductor:} \quad & r := r \otimes f_r(x, p, c_1, \ldots, c_k, i) \end{aligned}$$

donde

- $I_x \subseteq \mathbb{N}$ es un conjunto de índices que representa el *espacio de iteración*. En general, depende de la entrada x y tiene la forma $\{i \in lBnd(x) \ldots uBnd(x) : prop(i)\}$ donde $lBnd$ y $uBnd$ denotan su límite inferior y superior respectivamente, y $prop$ es una condición que actúa como filtro.
- p, c_1, ..., c_{k-1} y c_k son variables que representan las historias de valores calculados por el productor y los consumidores.
- f_p, f_{c1}, ..., f_{ck} y f_r son funciones básicas asociadas al productor, consumidores y reductor. En general, cada función básica puede consumir valores de variables de salida anteriores, lo que naturalmente induce un orden parcial que representa las dependencias de datos entre las operaciones de PCR.
- $\otimes$ es un combinador binario. r es una variable que representa el valor combinado (parcial), la salida del reductor. Su valor inicial viene dado por $r_0(x)$, al que a veces nos referimos como r_0. A diferencia de las variables de salida de los otros componentes, r es un escalar (es decir, no es una variable de flujo) y el orden de reducción con respecto a I_x no está especificado.

Se asume que todas las asignaciones, funciones básicas y operaciones de reducción son atómicas. Excepto cuando se indique lo contrario, asumimos que (1) I_x es finito, (2) el combinador $\otimes$ es una operación asociativa y conmutativa, y (3) $\otimes$ tiene un elemento de identidad $\mathrm{id}_\otimes$, de modo que $r_0(x) = \mathrm{id}_\otimes$ y este es también el valor predeterminado en caso de que el espacio de iteración esté vacío. Decimos que un PCR básico *A termina* en la entrada x si la operación de reducción se ha realizado para todo $i \in I_x$. Escribimos end_A para denotar esta condición. De las definiciones dadas, se sigue que la salida del PCR A es el valor de r cuando se cumple end_A.

3.2 Sintaxis básica.- En la Tabla I, presentamos una sintaxis simple para describir PCRs. Para cualquier variable de flujo de productor/consumidor `v`, `v[0]` refiere al valor en el índice de iteración actual *i* (es decir, v^i) y esto se puede escribir más convenientemente como `v`. Además, las funciones básicas tienen acceso opcional al índice parámetro *i* si necesitan actuar sobre su valor. Aquí no nos detenemos en la sintaxis o semántica del lenguaje anfitrión ya que no asumimos ninguno en particular.

Tipo	**Sintaxis**	**Descripción**
Espacio de iteración	**lbnd** $A(x)$ **ubnd** $A(x)$ **prop** $A(i)$	A es el nombre del PCR, y **lbnd**, **ubnd** y **prop** especifican el espacio de iteración.
Productor	$p =$ **produce** $f_p\ x\ p$	f_p es una función básica y x es la

		variable de entrada del PCR.
Consumidor	$c_k = \textbf{consume}\ f_c\ x\ p\ c_1 \ldots\ c_{k-1}$	f_c es una función básica, x es la variable de entrada del PCR, p es la variable de salida del productor y c_1, ..., c_{k-1} son las variables de salida de los consumidores previos.
Reductor	$r = \textbf{reduce} \otimes (r_0\ x)\ (f_r\ x\ p\ c_1 \ldots\ c_k)$	$\otimes$ es el combinador del reductor (con valor inicial r_0) y f_r es una función básica sobre la entrada del PCR y las salidas del productor y los consumidores.

Tabla I.- Sintaxis PCR.

También es útil tener sintaxis para el manejo explícito de dependencias. En general, para las variables `v`, `w` y los índices `i`, `j`, la instrucción **dep** `v(i)` → `w(j)` significa que el valor de `w` en `j` (es decir, w^j) depende del valor de `v` en `i` (es decir, v^i). Para mirar hacia atrás sobre `v` en `k` (es decir, v^{i-k}) la sintaxis es `v[-k]`, y para mirar hacia adelante sobre `v` en `k` (es decir, v^{i+k}) la sintaxis es `v[+k]`. Ningún valor puede depender de los valores presentes o futuros de la misma variable, ya que esta situación puede introducir *deadlock*. El código de la Figura III captura el ejemplo informal ilustrado en la Figura II, donde el PCR externo es llamado `FibPrimes` y el PCR interno es llamado `IsPrime`. Aquí `IsPrime` es un PCR básico, mientras que `FibPrimes` es un caso de PCR compuesto con otro PCR a través del consumidor.

3.3 ¿Qué computa un PCR?.- Aquí centramos nuestra atención en lo que computa un PCR. Nos interesa el comportamiento funcional y para esto buscamos una forma explícita que incluya las funciones básicas que constituyen el PCR. Todas las operaciones PCR, excepto las reducciones, son

```
fun fibs(p,i)  = if i <= 2 then 1 else p[-1] + p[-2]
fun count(r,c) = r + if c then 1 else 0
dep p(i-1) → p(i)
dep p(i-2) → p(i)
lbnd FibPrimes(N) = 1
ubnd FibPrimes(N) = N

fun divs(j) = j
fun notDiv(F,d) = not (F % d == 0)
lbnd IsPrime(F) = 2
ubnd IsPrime(F) = floor(sqrt(F))

PCR FibPrimes(N)                  PCR IsPrime(F)
  p = produce fibs p                d = produce divs
  c = consume IsPrime p             b = consume notDiv F d
  r = reduce count 0 c              a = reduce and (F > 1) b
```

Figura III.- Solución PCR al problema de contar números Fibonacci primos.

asignaciones de la forma $v^i := f_v(x, u_1, \dots, u_k, i)$. Entonces, las variables v y u_j son flujos mientras que f_v es una función en los flujos u_j (así como en la entrada x y el índice i). Podemos tratar a f_v también como un flujo si definimos $\overrightarrow{f_v}$ como una versión "currificada" de f_v

$$\overrightarrow{f_v}\,(x, u_1, \dots, u_k) = i \in \mathbb{N} \;\longmapsto\; f_v(x, u_1, \dots, u_k, i)$$

de manera que $v^i = \overrightarrow{f_v}^i\,(x, u_1, \dots, u_k)$ para cada i tal que $wrt(v^i)$. Esto nos permite expresar el efecto de las operaciones PCR como una composición de flujos originada a partir de funciones básicas en un estilo *index-free*. De esta forma, una sola expresión matemática se puede usar para caracterizar funcionalmente una clase abstracta de PCRs sin importar las dependencias particulares entre sus componentes. En particular, para un PCR básico *A* con un solo consumidor, si denotamos por $A(x)$ la salida de A en la entrada x, tenemos:

$$A(x) = \otimes_{i \in I_x} \overrightarrow{f_r}^i \left(x, \overrightarrow{f_p}(x), \overrightarrow{f_c}(x, \overrightarrow{f_p}(x))\right).$$

4. Formalización del patrón PCR en TLA+.- TLA+ es un lenguaje de especificación desarrollado por Lamport [3] que se ha utilizado con éxito en lugares como Intel, Amazon y Microsoft para especificar y verificar principalmente sistemas concurrentes. Se puede analizar en dos fragmentos: (1) La Lógica Temporal de las Acciones (TLA), una variante de la lógica temporal original de Pnueli que hace práctico escribir una especificación como una sola fórmula, y (2) una teoría de primer orden basada en la teoría de conjuntos de Zermelo-Fraenkel con el axioma de elección (ZFC) y un operador de elección.

En el marco de TLA+, la computación se entiende como la evolución discreta del estado descrito por una fórmula lógica temporal, donde el estado está formado por estructuras lógicas descritas en las matemáticas basadas en la teoría de conjuntos. En TLA+, tanto el sistema como sus propiedades son fórmulas en la misma lógica. En consecuencia, un sistema *S* satisface la propiedad *P* si y solo si $S \Rightarrow P$ es una fórmula válida de la lógica. Además, el sistema S_2 es un refinamiento del sistema S_1 si y solo si $S_2 \Rightarrow S_1$ es una fórmula válida de la lógica.

TLA+ posee un buen soporte de herramientas, una característica necesaria para el uso industrial. En particular, admite la verificación automatizada (en modelos finitos) mediante *model-checking* con la herramienta TLC [4] y la verificación semiautomática mediante la demostración de teoremas con la herramienta TLAPS [5]. Además, existe un IDE que está muy bien integrado con las herramientas y generalmente se considera como la forma preferida de trabajar con TLA+ [6]. Ahora presentamos una formalización en TLA+ de los modelos PCR abstractos y las funciones que computan. Hay principalmente dos temas a tratar:

1. **La semántica entrada-salida.** El comportamiento de entrada-salida de un PCR se identificó previamente con ciertas funciones bajo algunos supuestos algebraicos. Para tratar esto formalmente, organizamos los conceptos matemáticos requeridos, las propiedades y pruebas a través de una colección de módulos TLA+ donde solo se usan matemáticas clásicas.
2. **La semántica concurrente.** Se proporciona una semántica concurrente formal para PCR usando el fragmento temporal de TLA+. Para tal fin, llevamos a cabo una traducción directa de las operaciones de PCR como acciones en el sentido de TLA+ para que una ejecución de PCR pueda verse como un sistema de transiciones etiquetado (y justo).

Estos dos puntos están conectados en TLA+ por las nociones de corrección parcial (una propiedad de *safety*) y terminación (una propiedad de *liveness*). Juntas, nos aseguran que la semántica concurrente debe calcular una función apropiada en caso de terminación.

Además, las especificaciones de diferentes modelos de PCR están conectadas por la noción de refinamiento bajo sustituciones apropiadas. Las especificaciones están parametrizadas para las

funciones básicas y otros elementos concretos que debe proporcionar el usuario en circunstancias normales. De esta manera, las clases de PCR concretos se pueden representar con una única especificación, de modo que instanciar los parámetros de la especificación resulta en un PCR concreto particular.

4.1 Supuestos iniciales.- Fijamos algunos supuestos básicos sobre los PCRs y su ejecución. Estas suposiciones se hacen principalmente en aras de la simplicidad, pero algunas de ellas también se deben a limitaciones en el soporte de herramientas.
Asumimos una semántica de ejecución por *interleaving*, que es el enfoque más común adoptado para modelar la concurrencia y una abstracción razonable para la mayoría de los propósitos prácticos.
De acuerdo con las definiciones dadas en la Sección 3, las funciones básicas pueden leer todas las variables anteriores. Una simplificación que hacemos es leer solo de la variable inmediatamente anterior. No hay pérdida de expresividad, ya que cada función básica podría reenviar los valores anteriores junto con su salida si fuera necesario. Además, solo consideramos PCRs de un solo consumidor. Nuevamente, no hay pérdida de expresividad, ya que podemos expresar el cómputo de múltiples consumidores con uno solo.
Para favorecer la uniformidad en nuestras especificaciones, nunca asumimos que el PCR principal que se especifica se encuentra en la raíz de la jerarquía de ejecución. En general, los índices son secuencias de números naturales, formalmente objetos de la forma $I \circ \langle i \rangle \in Seq(Nat)$ (que informalmente escribimos como I, i) donde I es el índice de la instancia (también el índice de la instancia del padre) e i es la asignación actual en el PCR interior. Entonces, el punto de vista que adoptamos es que el PCR principal que se especifica opera a partir de algún índice base $I_0 \in Seq(Nat)$, que puede ser el índice vacío $\langle\rangle$ (en cuyo caso podría considerarse como la raíz de ejecución), pero de lo contrario significaría que se ubica más profundo en la jerarquía de ejecución, lo que refleja una configuración un poco más general.

4.2 Formalización de la semántica entrada-salida.- El resultado de un PCR básico A en la entrada x está dado (asumiendo que termina) por la expresión $\bigotimes_{i \in I_x} \overrightarrow{f_A}$ para alguna función $\overrightarrow{f_A}: \mathbb{N} \to D$ construida a partir de las funciones básicas del PCR y suponiendo que $(D, \mathrm{id}_{\otimes}, \otimes)$ es un monoide abeliano. Aquí necesitamos una definición formal adecuada para este tipo de expresiones en el lenguaje TLA+, que también debería ser manejable por sus herramientas. Para ello formalizamos la extensión de una operación binaria $\otimes$ a intervalos finitos (no necesariamente consecutivos) de $\mathbb{N}$ bajo ciertas propiedades algebraicas. Actualmente, la biblioteca estándar de matemáticas que ofrece TLAPS no es muy grande. Sin embargo, hay material suficiente para nuestras necesidades. Nuestra formalización es completamente independiente del concepto PCR y está organizada en varios módulos. Algunos se resumen en la Tabla II.

Módulo	**Descripción**
`AbstractAlgebra`	Definiciones de álgebra abstracta elemental.
`MonoidBigOp`	Operación general sobre una estructura monoide.
`AbelianMonoidBigOp`	Operación general sobre una estructura monoide abeliana.

Tabla III.- Módulos de la especificación TLA+ para conceptos algebraicos y sus teoremas asociados.

4.3 Formalización de los modelos abstractos de PCR.- La Tabla III resume algunos de los módulos TLA+ que representan los modelos abstractos de PCR. Cada módulo incluye la semántica entrada-salida correspondiente y la semántica concurrente. A continuación, explicamos brevemente el PCR básico. Los demás módulos pueden considerarse extensiones o variaciones.

Módulo	**Descripción**	**Instancia concreta**
`PCR_A`	PCR básico.	`FibPrimes1, IsPrime`
`PCR_A_c_B`	PCR compuesto a través del consumidor con un PCR básico.	`FibPrimes2`
`PCR_DC`	PCR que implementa la estrategia d*ivide-and-conquer*.	`MergeSort, NQueens`

Tabla IIII.- Módulos de la especificación TLA+ para los modelos PCR abstractos e instancias concretas.

Elementos como funciones básicas, tipos y dependencias de datos se parametrizan mediante símbolos constantes. En particular, Dep_pp, Dep_pc y Dep_cr son pares de conjuntos *look-ahead/look-behind* de la forma $\langle\{b_1, b_2, \ldots\}, \{a_1, a_2, \ldots\}\rangle$. El estado de un PCR básico está representado por las siguientes variables: (1) X: La entrada del PCR, una función (parcial) de índices al tipo de entrada T. (2) p : La variable de salida del productor, una función mapeando índices a funciones parciales de Nat al tipo del productor T_p. (3) c : La variable de salida del consumidor, una función mapeando índices a funciones parciales de Nat al tipo del consumidor T_c. (4) r : La variable de salida del reductor, una función de índices al tipo de salida D. (5) red : Una variable auxiliar para registrar las reducciones hechas en cualquier momento, una función que mapea índices a funciones de Nat a $BOOL$EAN. Las variables p, c y red representan el comportamiento interno del PCR, mientras que las variables X y r representan el comportamiento visible externo. Otro PCR puede querer escribir en la entrada X o leer en la salida r. Por conveniencia, las variables internas son funciones "currificadas" que separan el índice padre $I \in Seq(Nat)$ de la asignación $i \in Nat$. En notación TLA+, a todas estas variables se accede con corchetes (múltiples) ya que son funciones (de funciones). Para variables internas tenemos, por ejemplo, que $p[I]$ denota el flujo productor en el índice I mientras que $p[I][i]$ denota el valor de la i-ésima asignación del flujo productor en el índice I. En lo que sigue, a veces escribimos en la notación de superíndice como era habitual en la Sección 3, por ejemplo, p^I y $p^{I,i}$ respectivamente. La especificación de la semántica entrada-salida proporciona una fórmula que caracteriza la función matemática asociada al PCR básico. Esta fórmula se usa para afirmar la corrección y también se puede usar para representar el PCR como un cómputo de un solo paso. El módulo `AbelianMonoidBigOp` se instancia con la firma $(D, \mathrm{id}_{\otimes}, \otimes)$ y el supuesto *H_Algebra* afirma que satisface las leyes de un monoide abeliano, lo que nos da acceso a los teoremas correspondientes para poder hacer demostraciones deductivas. Para PCRs concretos esta hipótesis debe verificarse.

$$M \triangleq \text{INSTANCE } AbelianMonoidBigOp$$
$$\text{AXIOM } H\_Algebra \triangleq AbelianMonoid(D, \mathrm{id}_{\otimes}, \otimes)$$

La especificación de la semántica concurrente proporciona una fórmula temporal en la llamada forma canónica, una conjunción de fórmulas de *safety* y *liveness*. Una especificación no justa de línea de base principal, denominada *Spec* como de costumbre, y su versión justa (con condiciones de *fairness*) se definen de acuerdo con la forma canónica

$$Spec \triangleq Init \wedge \Box\,[\text{Next}]_{vs}$$
$$FairSpec \triangleq Spec \wedge WF_{vs}(Step)$$

donde $vs = \langle X, p, c, r, red \rangle$. Las condiciones iniciales las establece el predicado de estado *Init*. La salida del reductor es inicialmente por defecto la identidad del monoide $\mathrm{id}_{\otimes}$. Las salidas de productor y consumidor son inicialmente indefinidas, y el historial de reducción es inicialmente

falso. La acción *Next* gobierna la evolución del estado y es la disyunción de dos sub-acciones posibles y mutuamente excluyentes *Step* y *Done*. La acción *Done* detecta la terminación cuando todas las instancias bien definidas del PCR (dadas por el conjunto *WDIndex*) han terminado y el estado del sistema no cambia, esto es

$$Done \triangleq (\forall\, I \in WDIndex :\ end(I)) \wedge vs' = vs$$

La acción *Step* representa cualquier operación posible de los componentes básicos del PCR. Para esto, las operaciones son modeladas por acciones atómicas: *P* para el productor, *C* para el consumidor y *R* para el reductor, las cuales están parametrizadas por el índice de instancia *I* y la asignación actual *i*. Por lo tanto, *Step* se define como un (doble) cuantificador existencial que introduce no determinismo para modelar la ejecución concurrente de posibles instancias múltiples del PCR básico y las acciones de sus componentes:

$$Step \triangleq \exists\, I \in WDIndex :\ \exists i \in It(X^I) :\ P(I, i) \vee C(I, i) \vee R(I, i)$$

La corrección parcial y la terminación se formulan de la siguiente manera:

$$Correctness \triangleq \Box\,(end(I_0) \Rightarrow r^{I_0} = A(X^{I_0}))$$
$$Termination \triangleq \Diamond end(I_0)$$

4.4 Verificación de propiedades. - Cada módulo tiene una sección que establece las propiedades de *safety* y *liveness* asociadas al PCR especificado en dicho módulo y, posiblemente, también propiedades de refinamiento relacionadas con PCRs especificados en otros módulos. Aquí, las propiedades de *safety* incluyen la corrección parcial del PCR y otras propiedades invariantes como la corrección del tipo, mientras que las propiedades de *liveness* son, más específicamente, la propiedad de terminación. En el marco de TLA+, la verificación mecánica se puede realizar de las siguientes maneras:

- ***Model-checking* con TLC.** Los parámetros de símbolos constantes deben instanciarse con definiciones concretas que puedan evaluarse, lo que da como resultado PCRs concretos. Esto significa que el *model-checking* se puede utilizar como método de verificación para PCRs concretos sobre tipos de datos finitos. TLC permite verificar propiedades de *safety*, *liveness* y refinamientos.
- **Demostración deductiva con TLAPS.** Para el enfoque deductivo no necesitamos instanciar parámetros de símbolos constantes. Esto significa que la verificación se puede realizar para el modelo de PCR abstracto sin restricciones de finitud. TLAPS actualmente permite verificar las propiedades de *safety* y el componente de *safety* en las propiedades de refinamiento (es decir, sin las condiciones de *fairness*).

5. Conclusión. -

En trabajos futuros, esperamos poder extender nuestro trabajo en otras direcciones. En particular, sabemos que en las próximas versiones de TLAPS será posible tratar formalmente la terminación que es una propiedad de *liveness*. Además, visualizamos una herramienta de traducción automática para PCR con el mismo espíritu de PlusCal, un lenguaje algorítmico que se traduce a TLA+ [7]. Actualmente hay trabajos que están investigando la extracción de código ejecutable desde PlusCal, creemos que esto podría beneficiarnos y, por lo tanto, no es poco realista apuntar a programas PCR ejecutables y correctos. En resumen, creemos que esta tesis contribuye al estado del arte en el refinamiento formal de programas paralelos a partir de modelos abstractos, especialmente partiendo de una caracterización alternativa del patrón general de PCR, y utilizando el marco teórico y práctico que ofrece TLA+. Esperamos haber mostrado esto como una interesante línea de investigación dentro de la búsqueda general de la programación de calidad.

6. Referencias. -

[1] Asanovic, K. et al; *A view of the parallel computing landscape*, Commun. ACM, 2009. 52(10) : p. 56–67.

[2] Pérez, G. y Yovine, S.; *Formal specification and implementation of an automated pattern-based parallel-code generation framework*, STTT, 2017. 21(2) : p. 183–202.

[3] Lamport, L.; *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, 2002, Addison-Wesley Longman Publishing.

[4] Yu, Y. et al; *Model Checking TLA+ Specifications*, Correct Hardware Design and Verification Methods, 1999. 1703 : p. 54–66.

[5] Cousineau, D. et al.; *TLA+ Proofs*, FM 2012: Formal Methods, 2012. 7436 : p. 147–154.

[6] Kuppe, M. et al; *The TLA+ Toolbox*, ArXiv, 2019. p. 50–62.

[7] Lamport, L.; *The pluscal algorithm language*, Theoretical Aspects of Computing-ICTAC, 2009. 5684 : p. 36–60.

Nota contribución de los autores:

1. Concepción y diseño del estudio

2. Adquisición de datos

3. Análisis de datos

4. Discusión de los resultados

5. Redacción del manuscrito

6. Aprobación de la versión final del manuscrito

JES ha contribuido en: 1, 2, 3, 4, 5 y 6.

Nota de aceptación: Este artículo fue aprobado por los editores de la revista Dr. Rafael Sotelo y Mag. Ing. Fernando A. Hernández Gobertti.