Facultad de Ciencias Exactas, Ingeniería y Agrimensura - UNR
Departamento de Ciencias de la Computación

Propuesta de Tesina

Postulante:

Bini, Valentina María

Legajo: B-5926/9

Director:

Rivas, Exequiel

13-12-2022

1. Situación de la postulante

Al día 10/12/2022 tengo aprobadas 33 materias de la carrera y sólo tengo pendiente realizar la tesina para finalizar la carrera de Licenciatura en Ciencias de la Computación. Me encuentro trabajando como docente en la Facultad de Ciencias Exactas, Ingeniería y Agrimensura (UNR) 20 horas semanales (2 cargos simples).

2. Título

??

3. Motivación y objetivo general

La ley de Moore postulaba que el número de transistores que se pueden poner un chip de computadora dobla (de manera aproximada) cada un par de años. De manera práctica, esto significaba que para obtener mejorías en la velocidad de ejecución de un programa simplemente había que esperar: después de un par de años, los programas que escribiríamos hoy ejecutarían más rápido entonces.

Sin embargo, esta ley ya no se verifica en la práctica. Por razones físicas, es cada vez más costoso aumentar el número de transistores de un chip, y este tipo de progreso ya no puede ser garantizado. Los programadores se ven obligados a encontrar otras maneras de mejorar el desempeño de sus programas, y una de las maneras más naturales que surgió fue pensar que podemos resolver varias tareas al mismo tiempo, y de esa manera evitar puntos ociosos donde solo estamos esperando el entorno (por ej. I/O bloqueante). En los últimos años, con la masificación de los microprocesadores con múltiples core, se hace aún más evidente la necesidad de proveer al programador con la capacidad de concurrencia.

Si bien los sistemas operativos modernos nos dan primitivas para manejar la concurrencia, no siempre los lenguajes de programación han incorporado estas características de manera natural. Los lenguajes de programación imperativos en general se basan en la idea de un modelo de ejecución secuencial, donde la computación se desarrolla siguiendo una única serie de pasos. Es así que lenguajes como C o Java pueden manejar concurrencia, pero sin proponer cambios radicales en la concepción del lenguaje: simplemente librerías que capturan esta capacidad de manera externa.

La misma situación se refleja en los lenguajes de programación funcionales, sobre todo cuando se intenta escribir programas con efectos. Los programas con efectos suelen representarse utilizando mónadas, de manera que los programas funcionales toman una apariencia fundamentalmente imperativa. Al igual que lo que sucede en la programación imperativa, al considerar la

concurrencia de los efectos, la manera usual de propuesta es mediante llamadas a funciones ad-hoc, y no usando primitivas bien fundadas que deriven de alguna estructura matemática, así como lo hacen las operaciones de las mónadas.

Las mónadas concurrentes fueron recientemente definidas buscando obtener primitivas bien fundadas para la concurrencia, extendiendo las mónadas con nuevos operadores. Si bien esta estructura es matemáticamente bien fundada, basada en monoides concurrentes, es considerablemente complicado encontrar y probar modelos válidos de esta estructura.

Una hipótesis particular es que la mónada de Delay, utilizada para modelizar el efecto de no terminación, puede ser dotada de estructura de mónada concurrente. En esta tesina desarrollaremos el concepto de mónada concurrente en el ámbito del lenguaje de pruebas Agda, y con el objetivo principal de probar o refutar la hipótesis de que a la mónada Delay se le puede dar una estructura de mónada concurrente.

4. Fundamentos y estado de conocimiento sobre el tema

El uso de mónadas para estructuras semánticas de lenguajes con efectos fue desarrollado originalmente por E. Moggi [3, 4]. Poco más tarde, P. Wadler [6] adaptó el concepto de manera interna en los lenguajes de programación funcional, dando origen a la programación con mónadas. Existe una gran variedad de efectos que pueden ser capturados usando mónadas internas, por ejejmplo, estado, excepciones, entornos, continuaciones, etc.

La mónada Delay fue introducida por Capretta [1] con el objetivo de capturar el efecto de no terminación de manera explícita y uniforme. Su estructura fue estudiada en varios artículos, entre los que podemos mencionar la tesis de N. Veltri.

Las mónadas concurrentes fueron recientemente introducidas en una preimpresión por M. Jaskelioff y E. Rivas [5]. Esta definición surge de categorificar la noción de monoide concurrente, definido por C. A. R. Hoare et al. [2] hace unos años.

Agda es un asistente de pruebas basado en teoría de tipos, con soporte induction-recursion, y que ya ha sido utilizado para estudiar la mónada Delay.

5. Metodología y plan de trabajo

1. Estudio de la estructura de mónada concurrente.
2. Introducción de la clase de mónada concurrente en Agda.
3. Estudio de la mónada Delay y su implementación en Agda.

4. Adaptación de las operaciones de la mónada Delay en el contexto de 2, y las pruebas de estructura de mónada y functor monoidal.

5. Prueba del axioma de intercambio de mónada concurrente en Agda para la mónada Delay.

Referencias

[1] Venanzio Capretta. General Recursion via Coinductive Types. *Logical Methods in Computer Science*, Volume 1, Issue 2, July 2005.

[2] C. A. R. Hoare, Akbar Hussain, Bernhard Möller, Peter W. O'Hearn, Rasmus Lerchedahl Petersen, and Georg Struth. On locality and the exchange law for concurrent processes. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 – Concurrency Theory*, pages 250–264, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[3] Eugenio Moggi. Computational lambda-calculus and monads. In *Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, 1989.

[4] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55 – 92, 1991.

[5] Exequiel Rivas and Mauro Jaskelioff. Monads with merging. working paper or preprint: hal-02150199, June 2019.

[6] Philip Wadler. The essence of functional programming. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, page 1–14, New York, NY, USA, 1992. Association for Computing Machinery.