**FICHA PARA LA INSCRIPCIÓN DE PERFILES DE PROYECTOS**[[1]](#footnote-1)

1. **IDENTIFICACIÓN DEL PERFIL**

|  |  |
| --- | --- |
| Título de proyecto / trabajo de grado: | Expressive Programming Language for Mathematics  Lenguaje de Programación Expresivo para las Matemáticas |
| Estudiante (s): | Juan Felipe Cañizares Corrales, Lukas Restrepo Suárez |
| Docente – Director: | Daniel Cañizares Corrales, Mario Zuluaga Tobón |
| Programa académico: | Ingeniería de Sistemas |
| Nivel de formación (semestre): | Décimo semestre |
| Fecha de formulación: | Septiembre 28 de 2015 |

1. **DESCRIPCIÓN DEL PERFIL[[2]](#footnote-2)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Antecedentes:** *constituyen la explicación de los desarrollos previos relacionados con el tema, de orden tecnológico, científico, histórico, político, metodológico, social, ambiental, entre otros. Pueden incluir elementos como revisión de bibliografía que permita analizar el estado actual de la situación y opinión de personas expertas en el tema o experiencias previas (referenciados en el ámbito local, regional, nacional e internacional).* | | | |
| Actualmente existen herramientas de programación que están orientadas a resolver problemas matemáticos como MatLab y Mathematica de Wolfram, aunque poseen numerosas librerías, son soluciones de un costo elevado y poca expresividad en el campo de las matemáticas. Expresividad entendida como la posibilidad de especificar las soluciones lo más cercano posible al dominio del problema.  Por otro lado, se encuentran herramientas más simples como AMPL y MathPiper; éstas permanecen alejadas de facilitar al programador un lenguaje de programación que rompa los paradigmas clásicos y los acerque a un punto en el que sea posible manipular objetos matemáticos con naturalidad.  Haskell, el más reconocido de los lenguajes funcionales, en algunos sentidos logra mayor expresividad matemática, sin embargo la sutil sintaxis y el sofisticado sistema de tipos de Haskell son un arma de doble filo – altamente apreciado por los programadores experimentados, pero también una fuente de frustración entre los principiantes(Heeren, Leijen, & van IJzendoorn, 2003). | | | |
| **Planteamiento del problema:** *el problema se puede entender como una situación no deseada, que ocurre en un momento dado y que impide alcanzar un estado normal o una condición positiva desde la perspectiva de un grupo social o de un entorno específico. Según su tipología los problemas pueden ser económicos, sociales, tecnológicos, técnicos, culturales, ambientales, legales, entre otros. Por lo tanto, para la identificación y definición de un problema pueden revisarse estos distintos componentes.* | | | |
| Actualmente no existen lenguajes de programación gratuitos y expresivos en el campo de las matemáticas que permitan abordar problemas de forma natural e intuitiva. | | | |
| **Pregunta de investigación:** *plantear a través de una o más preguntas el problema que se estudiará. Estas deben ser claras y delimitadas para esbozar problema y sugerir actividades pertinentes para la investigación.* | | | |
| ¿Se puede construir un lenguaje de programación que permita trabajar con objetos matemáticos con la facilidad de uso de un lenguaje imperativo, manteniendo el poder computacional del cálculo lambda de un lenguaje funcional? | | | |
| **Justificación:** *se exponen claramente las circunstancias que dieron lugar a la elección del tema; es decir, explicar por qué es necesario, conveniente y útil el desarrollo de la investigación desde el orden social, económico y científico.* | | | |
| Tanto en el ámbito científico como académico, se requiere el uso constante de las matemáticas para solucionar o plantear problemas. Muchas veces se obtienen complejas funciones que sin ayuda de un ordenador tomaría años para computar un resultado. El modelo computacional de Turing y la arquitectura Von Newman conllevan a que el común denominador de los lenguajes más usados pertenezcan al conjunto del paradigma imperativo, pero al intentar expresar modelos matemáticos bajo este paradigma se requiere numerosas líneas de código (que tiende a volverse oscuro y complejo).  Por su parte, existe otro modelo llamado cálculo lambda, que también permite expresar cualquier función matemática computable (Turing Complete). Los lenguajes que hacen uso del cálculo lambda están dentro del conjunto de los llamados lenguajes funcionales. El gran problema de estos últimos, es su intrincada gramática y la difícil representación diferentes tipos de problemas, un claro ejemplo sería Haskell.  Actualmente existe otro conjunto de herramientas como MatLab o Mathematica de Wolfram que hacen uso del paradigma imperativo y declarativo. A simple vista parece resuelto el problema, pero no es así, ambos lenguajes sacrifican demasiado el poder computacional del cálculo lambda y terminan siendo lenguajes con una gran cantidad de librerías que permiten el uso de funciones matemáticas pero que pierden la expresividad, lo cual resulta en soluciones complicadas y alejadas del planteamiento matemático inicial. | | | |
| **Objetivo General.** *Describe qué se propone al hacer el proyecto. El objetivo general es la razón para ejecutar el proyecto a través de la producción de resultados tangibles. Se refiere al efecto anticipado que se espera como producto de alcanzar los resultados*  **Objetivo Específicos***. Se derivan del objetivo general, los cuales indican los fines inmediatos de cada una de las etapas de la investigación, estos no se deben confundir con actividades o procedimientos metodológicos. Son expresados de manera clara y concreta; que sean alcanzables y medibles.* | | | |
| Objetivo general: Indagar si la teoría actual de lenguajes formales permite definir un lenguaje de programación en el área de la matemática computacional.  Objetivos específicos:   1. Sintetizar aspectos relevantes de la teoría de compiladores y lenguajes formales. 2. Definir un lenguaje que resuelva la problemática de trabajar con objetos matemáticos. 3. Determinar si la teoría actual soporta el lenguaje definido. | | | |
| **Marco teórico, referencial o conceptual:** *es precisar, organizar y exponer las ideas, conceptos y teorías que sirvan de sustento para el análisis del problema o fenómeno a investigar, apoyado en los resultados teóricos de la revisión bibliográfica y documental.* | | | |
| **Matemática Computacional**  La matemática computacional es el conjunto de programas de ordenador, algoritmos, técnicas y teorías necesarios para resolver modelos matemáticos de problemas que surgen en la ciencia y la tecnología. Ciencia entendida en un sentido global, incluyendo tanto la economía, las ciencias sociales y de la salud, como las ciencias clásicas – física, química, biología, geología(Fasshauer, 2011). La matemática computacional debe proporcionar soluciones prácticas y concretas a los problemas. En ese sentido se basa en disciplinas matemáticas clásicas y, por supuesto, en el análisis numérico, pero va más allá de ellas. A la matemática computacional no le basta con desarrollar y analizar nuevos algoritmos, debe proporcionar programas que cumplan con tres requisitos: precisión, eficiencia, robustez y estabilidad(Sterck & Ullrich, 2006).  **Programación imperativa**  El paradigma imperativo se fundamenta en el modelo matemático presentado por el inglés Alan Turing, en su trabajo sobre la “máquina automática” conocida hoy como máquina de Turing, un modelo comúnmente aceptado en las ciencias de la computación ya que “la mayoría de los lenguajes de programación pueden simular una máquina de Turing”(Arora & Barak, 2009). La máquina tiene una cinta infinita dividida en celdas que contienen un símbolo, hay un número finito de símbolos y uno especial en blanco pertenecientes a un lenguaje. Existe una tabla de reglas finita con instrucciones, dada una entrada de la cinta, se modifica o no el estado de la máquina, el estado se guarda en un registro para el próximo computo.(Kleinberg, 2012)  Internamente, el procesador de un computador está estructurado bajo el paradigma imperativo, por eso es tan común en nuestros tiempos.  Los lenguajes imperativos se basan en mover bits y cambiar el estado de la máquina. Su unidad básica de abstracción es el procedimiento, el cual tiene un conjunto secuencial de instrucciones. Las variables asumen un rol muy importante pues son las que cambian el estado de la máquina, y sirven como abstracción a las celdas de la memoria del hardware(Bellaachia, n.d.). La mayoría de lenguajes que se usan hoy en día son basados en este paradigma.  *En el paradigma imperativo un algoritmo especifica como debe resolverse el problema usando una secuencia de instrucciones.*  **Programación funcional**  La programación funcional puede verse como la versión de propósito general del paradigma declarativo, es decir funcional es un paradigma de programación declarativo.  El paradigma declarativo obedece al modelo del cálculo lambda donde una función toma como parámetros otras funciones y retorna funciones como resultado(Thompson, 2011). Se dice que la función es el *objeto de primera clase,*en este sistema a la función se le denota con letra griega λ (lambda)(Thompson, 1999).  En programación funcional no hay variables (o las variables son inmutables) dado que a las funciones solo les interesan hacer procesamiento sobre datos. En principio, no hay interés en guardar el estado de nada, cada función opera sobre sus datos y no depende de otra función, por lo cual no hay una secuencia de ejecución(Evans, 2011; Hughes, 1989).  *En el paradigma declarativo se describe el problema sin importar el algoritmo necesario para conseguir la solución.*  **Calculo Lambda**  Fue introducido en los 30's por Alonzo Church y Stephen Kleene con el objetivo de dar una teoría general de las funciones es un sistema formal diseñado para definir funciones, la forma de utilizarlas y la recursión. El cálculo lambda influye en lenguajes funcionales como Lisp, Haskell y ML. Es utilizado como un fundamento de lenguajes de programación porque aporta una sintaxis básica de programación, la semántica para el concepto de función en la transformación de argumentos en resultados y una forma de definir primitivas de programación(Barendregt & Barendsen, 1984).  El cálculo lambda es el lenguaje de programación más pequeño consiste en transformación simple (sustituir variables) y en definir funciones. El cálculo lambda se puede decir que es equivalente a las máquinas Turing porque es capaz de evaluar y expresar cualquier función computable.(Arora & Barak, 2009)  Church había querido hacer un sistema formal completo para modelar la matemática pero después separo el cálculo lambda y lo ideó para que estudiara la computabilidad.  **Compilador**  Un compilador es un programa que traduce un programa escrito en un lenguaje de alto nivel, el cuál es más fácil de entender para los humanos, y genera un nuevo programa escrito en código máquina que será ejecutado por el computador. El primer compilador fue inventado por Grace Hopper en la década de los 50’(Evans, 2011). John Backus es reconocido como el padre de la teoría de compiladores, quien con un equipo de IBM desarrollo el primer compilador completo que era capaz de traducir código escrito en el lenguaje FORTRAN al lenguaje máquina.  Un compilador no siempre genera como traducción lenguaje máquina, aunque suele ser lo más común. Es normal crear compiladores que generen otro tipo de salidas diferentes al lenguaje máquina. Por ejemplo, un *intérprete* puede procesar lenguaje de alto nivel. Para lograr esto, el compilador traduce el programa original a uno escrito en un lenguaje intermedio que entiende el intérprete (Aho, Sethi, & Ullman, 2008; Tobón Zuluaga, n.d.) | | | |
| **Diseño metodológico:** *son los procedimientos generales mediante los cuales se pretende lograr los objetivos de la investigación. Se debe definir el tipo de investigación, seleccionar el diseño de la investigación, tipo de variables, técnicas de colecta de la información y precisar cuáles son las técnicas de análisis de la información.* | | | |
| Investigación cuantitativa, se recolectará la teoría que soporta los lenguajes de programación actuales, y se implementará el prototipo inicial del compilador. | | | |
| **Referencias bibliográficas[[3]](#footnote-3):** *normas APA, de acuerdo con las indicaciones del MEI* | | | |
| Bibliografía  Aho, A., Sethi, R., & Ullman, J. (2008). *Compiladores: principios, técnicas y herramientas* (Segunda.). Pearson Educación. Retrieved from http://books.google.com/books?hl=en&lr=&id=yG6qJBAnE9UC&oi=fnd&pg=PA1&dq=Compiladores+principios,+t%C3%A9cnicas+y+herramientas&ots=rsWJQ-a0fP&sig=lDXyXmKQ5DtFJi9sU\_0QtMLWIL0  Arora, S., & Barak, B. (2009). *Computational Complexity a modern approach*.  Barendregt, H., & Barendsen, E. (1984). Introduction to lambda calculus. *Nieuw Archief Voor …*, (March). Retrieved from http://homepages.nyu.edu/~cb125/Lambda/barendregt.94.pdf  Bellaachia, A. (n.d.). Imperative Programming Languages ( IPL ). *George Washington University*.  Evans, D. (2011). *Introduction to Computing*. University of Virginia.  Fasshauer, G. (2011). MATH 350: Introduction to Computational Mathematics. *Department of Applied Mathematics Illinois Institute of Tecnology*. Retrieved from http://amadeus.math.iit.edu/~fass/Notes350\_Ch3.pdf  Heeren, B., Leijen, D., & van IJzendoorn, A. (2003). Helium, for learning Haskell. *Proceedings of the ACM SIGPLAN Workshop on Haskell - Haskell ’03*. doi:10.1145/871895.871902  Hughes, J. (1989). Why functional programming matters. *The Computer Journal*. Retrieved from http://comjnl.oxfordjournals.org/content/32/2/98.short  Kleinberg, B. (2012). Definition of a Turing machine. *Cornell University*.  Sterck, H. De, & Ullrich, P. (2006). INTRODUCTION TO COMPUTATIONAL MATHEMATICS Course Notes for. *Department of Applied Mathematics University of Waterloo*.  Thompson, S. (1999). *Type theory and functional programming*. Retrieved from https://izzyderpderp.com/Gentoomen/Programming/Functional Programming/Type Theory and Functional Programming.pdf  Thompson, S. (2011). *The Craft of Functional Programming Second edition*. (Addison-Wesley, Ed.) (Third.).  Tobón Zuluaga, M. (n.d.). *Elementos basicos de compiladores*. (M. Zuluaga Tobón, Ed.). | | | |
| Firma estudiante Lukas Restrepo Suárez | Firma asesor  Mario Zuluaga Tobón | Fecha aprobación | V° B° Comité de Currículo y/o Consejo de Facultad |
| Firma estudiante Juan Felipe Cañizares Corrales | Firma asesor  Daniel Cañizares Corrales |  |  |

1. La extensión no debe exceder de tres (3) páginas, en letra Arial 10. Debe ser enviado impreso y en medio magnético a la Facultad respectiva. [↑](#footnote-ref-1)
2. Tomado de: MEI. 2013. Universidad Católica de Oriente. [↑](#footnote-ref-2)
3. Normas APA. [↑](#footnote-ref-3)