Verificación Formal
Posgrado en Ciencia e Ingeniería de la Computación
UNAM

Dr. Favio Ezequiel Miranda Perea
favio@ciencias.unam.mx
Facultad de Ciencias UNAM

26 de octubre de 2020

1. Sinopsis

El análisis y la verificación de sistemas de software es un asunto importante, los errores en sistemas cuya seguridad es crítica pueden ser desastrosos y causar pérdidas financieras enormes o incluso muertes, por lo que los métodos para evitar estos escenarios son esenciales. Un proceso de pruebas unitarias preliminares puede identificar problemas, si se realiza de manera rigurosa, pero por lo general no es suficiente para garantizar un nivel satisfactorio de calidad. Por otra parte las teorías de especificación y verificación formal ofrecen metodologías que van desde la descripción de requerimientos, lo cual permite razonar rigurosamente sobre ellos, hasta sofisticadas técnicas interactivas para la verificación de software. El objetivo de este curso es proporcionar un panorama de algunas metodologías de verificación formal mediante el uso del asistente de pruebas Coq para desarrollar tareas de verificación en estructuras de datos, lenguajes de programación y/o matemáticas dependiendo de los intereses particulares de la audiencia.

2. Prerrequisitos

El curso intentará ser autocontenido. Es deseable tener nociones de lógica computacional y programación funcional, aunque se revisarán a detalle los conceptos necesarios.

3. Temario

1. Introducción: discusión y estudio de casos simples como son:
   - Estructuras de datos: equivalencia de distintas representaciones de números naturales; operaciones en listas finitas y árboles
   - Lenguajes de programación: equivalencia de semánticas, optimización en el proceso de evaluación.
   - Matemáticas formalizadas: razonamiento en teoría de órdenes, álgebras booleanas y/o grupos.
   - Asistentes de pruebas: ideas generales e introducción a Coq. http://coq.inria.fr

2. Nociones de programación funcional:
    - Definición de estructuras de datos
    - Polimorfismo
    - Funciones de orden superior.
3. Nociones de lógica para la práctica:
    - Sintaxis
    - Lógica ecuacional
    - Cálculo de secuentes
    - Tácticas y razonamiento hacia atras.
4. Estudio de casos particulares:
    - Semánticas operacionales en lenguajes de programación.
    - Árboles binarios de búsqueda.
    - Arreglos flexibles.
    - Latices y conexiones de Galois
    - Casos de interes particular desarrollados por los alumnos.

4. Logística

El curso utilizará la plataforma Google Classroom/Meet y será mayormente autogestivo. Las sesiones síncronas serán empleadas para discutir dudas particulares. Se proporcionarán materiales para cada uno de los temas (notas, presentaciones o cuadernos interactivos (jupyter notebooks)).

5. Evaluación

La calificación final se obtendrá mediante la evaluación de las siguientes actividades.

- Ejercicios: uno cada dos o tres semanas: 40 %
- Proyecto de medio semestre con complejidad intermedia: 25 %
- Proyecto de final de semestre con complejidad alta: 35 %

Los proyectos son de libre elección y se califican en tres aspectos: el documento asociado; la exposición; y el contenido y calidad del desarrollo en el asistente de pruebas. Se sugiere elegir en cada caso un tema del particular interés de cada alumno, quizás relacionado con su trabajo de grado.

6. Bibliografía

La bibliografía es sólo de apoyo y se actualizará con artículos particulares durante el avance del curso.

- Yves Bertot, Pierre Casteran. Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. An EATCS Series. Springer 2004.
- Adam Chlipala. Certified Programming with Dependent Types. `http://adam.chlipala.net/cpdt/`
- Assia Mahboubi, Enrico Tassi. Mathematical Components. `https://math-comp.github.io/mcb/`
- Software Foundations book series. `https://softwarefoundations.cis.upenn.edu/`