UNC Universidad Nacional de Córdoba | FAMAF Facultad de Matemática, Astronomía y Física

EXP - UNC: 64763/2014

Córdoba, 30 de Marzo de 2015

VISTO

La Res. CD N° 434/2014 que fija el programa, correlativas y carga horaria de la materia optativa de la Licenciatura en Ciencias de la Computación "Diseño e Implementación de Compiladores"; y

CONSIDERANDO

Que se ha cometido un error involuntario en la carga horaria de la materia.

Por ello,

EL CONSEJO DIRECTIVO
DE LA FACULTAD DE MATEMÁTICA, ASTRONOMÍA Y FÍSICA

R E S U E L V E:

ARTÍCULO 1°: Reemplazar el Anexo I de la Res. CD N° 434/2014 por el Anexo I que forma parte de la presente Resolución.

ARTÍCULO 2°: Comuníquese y archívese.

DADA EN LA SALA DE SESIONES DEL CONSEJO DIRECTIVO DE LA FACULTAD DE MATEMÁTICA, ASTRONOMÍA Y FÍSICA A TREINTA DÍAS DEL MES DE MARZO DE DOS MIL QUINCE.

RESOLUCIÓN CD Nº 113/2015

Dra. NESVIT CASTELLANO
VICEDECANA
FaMAF

Dra. Ing. MIRTA IRIONDO
DECANA
FaMAF

EXP-UNC 64763/2014

Res. CD N° 113/2015

ANEXO I
PROGRAMA DE ASIGNATURA

ASIGNATURA: Diseño e implementación de Compiladores	**AÑO:** 2015
CARÁCTER: Optativa	
CARRERA: Licenciatura en Ciencias de la Computación	
RÉGIMEN: Cuatrimestral	**CARGA HORARIA:** 120 hs
UBICACIÓN en la CARRERA: Cuarto año – Primer Cuatrimestre	

FUNDAMENTACIÓN Y OBJETIVOS

Los temas involucrados en el diseño e implementación de Compiladores conforman una parte muy relevante en la currícula de Ciencias de la Computación. El desarrollo de un compilador involucra gran cantidad de técnicas y métodos que permite aplicar una cantidad considerable de conceptos teóricos y de implementar técnicas y algoritmos en casos específicos.

Es muy importante en la formación de estudiantes, programadores y diseñadores adquirir conocimientos avanzados sobre el diseño y la implementación de algoritmos y técnicas de análisis de programas como las usadas en los compiladores para la optimizaciones y/o transformaciones de código. Como así también, aquellas usadas para encontrar *bugs* y/o para verificar propiedades del código.

El campo abarcado por la compilación y el análisis estático de programas cubre una amplia variedad de técnicas muy conocidas y difundidas que en los últimos años han despertado interés en distintas áreas tales como verificación y re-ingeniería de software. En especial, el análisis estático permite extraer propiedades sobre un programa antes de la ejecución real del mismo. Esta información puede ser luego utilizada para realizar transformaciones sobre el mismo (compilación, optimizaciones, etc) y/o realizar pruebas sobre su corrección. En este curso se hará especial énfasis en las técnicas y conceptos básicos del análisis estático de código y sus principales aplicaciones. En particular se estudiará como estas técnicas se aplican a la generación, optimización y verificación automática de software.

El objetivo esencial del curso es lograr que los alumnos se familiaricen con las técnicas de compilación y análisis estático de software, que comprendan los fundamentos teóricos que hacen posibles la aplicación de estas técnicas en la práctica, y sepan aplicarlas en situaciones concretas con el fin de poder apreciar su importancia en el proceso de compilación aplicado por los compiladores modernos (optimización, análisis, verificación).
Los estudiantes deben ser capaces de:

- Entender el proceso de compilación; y
- Entender y aplicar análisis estáticos y optimizaciones a programas.

Objetivos Específicos: Los estudiantes que cumplan con los objetivos del curso serán capaces de:

- Explicar los conceptos introducidos;
- Explicar la complejidad y los límites teóricos de los análisis introducidos;
- Describir el comportamiento de las fases de compilación;
- Explicar análisis concretos y aplicar estos a programas simples;
- Realizar análisis semánticos incluyendo chequeos estáticos y representaciones intermedias;
- Realizar optimzaciones y verificaciones usando diversos análisis estáticos.

CONTENIDO

Unidad 1. Introducción. Compiladores e interpretes. Fases del compilador. Herramientas. Diseño de un compilador. Optimización. Análisis de código para optimización de performance. Análisis de código para verificación y seguridad.

Unidad 2. Lenguajes regulares y gramáticas libres de contexto. Definición de la sintaxis. Análisis sintáctico. Técnicas modernas de parsing. Sintaxis abstracta.

Unidad 3. Código Intermedio. Representación de código intermedio: arboles sintácticos abstractos, grafos de control de flujo, código de direcciones, static single assignament. Herramientas.

Unidad 4. Análisis de flujo de control. Dominadores y postdominadores. Análisis de flujo de datos. Análisis de dependencias. Análisis de alias. Análisis intraprocedural e interprocedural.

Unidad 5. Introducción a la optimización. Optimizaciones: propagación de constantes, eliminación de redundancia, optimización en ciclos, variables inductivas, dependencia de control y de datos, dependencia de datos en ciclos, código muerto, asignación de registros, scheduling de código. Optimizaciones interprocedurales. Optimizaciones en cada fase del compilador.

Unidad 6. Métodos de generación de código: Basados en Pattern-Matching y métodos ad-hoc.

Unidad 7. Program Slicing: Introducción. Grafo de dependencias del programa (PDG). Forward y Backward. Intra e interprocedural. Aplicaciones.

BIBLIOGRAFÍA

BIBLIOGRAFÍA BÁSICA

- Compilers: Principles, Techniques, and Tools. Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman and Monica S. Lam. Segunda edición. 2006.
- Advanced Compiler Design and Implementation. Steven Muchnick. Morgan Kaufmann. 1997.
- Basics of Compiler Design. Torben Mogensen. University of Copenhagen. ISBN: 978-87-

993154-0-6. Edition 2010.

BIBLIOGRAFÍA COMPLEMENTARIA

- Principles of program analysis. Flemming Nielson, Hanne Riis Nielson and Chris Hankin. Springer Verlag, 2005.
- Building an Optimizing Compiler. Robert Morgan. Digital Press. 1998.

METODOLOGÍA DE TRABAJO

La asignatura está organizada como un taller o laboratorio. En ella mediante la realización de un proyecto (construcción de un compilador para un subconjunto de un lenguaje de programación), se aplican un número considerable de técnicas, muchas de estas técnicas son casos particulares de técnicas fundamentales.

El proyecto consiste en diseñar y desarrollar un compilador para un lenguaje de programación minimalista. El proyecto se divide en 6 etapas que abarcan todo el proceso de compilación: análisis léxico, análisis sintáctico, análisis semántico, generación de código intermedio, generación de código objeto (assembly) y optimización del código. Cada una de las etapas consta de diseño, implementación y testing de los artefactos desarrollados. Los contenidos básicos a desarrollar son aquellos relacionados con cada etapa: análisis léxico, análisis sintáctico, análisis semántico, código intermedio, código objeto, generación de código intermedio y objeto y optimización del código.

Se guía al alumno para que: (1) pueda vislumbrar que el campo de aplicación de las técnicas dadas es más amplio; y (2) reconozca que existen una gran cantidad de técnicas y algoritmos, no todas vistas en clase, para diseñar e implementar un compilador. Además se generan espacios de discusión para que el alumno pueda: (1) asociar la aplicación de determinada técnica en otros contextos; (2) determinar la generalización de la técnica; e (3) investigar otras alternativas de diseño e implementación.

Los contenidos fueron seleccionados con el fin de que el alumno cuente con los conocimientos necesarios para culminar exitosamente el proyecto. Las actividades se seleccionaron con el fin de cumplir con los objetivos propuestos, los cuales, todos contribuyen a fortalecer el perfil del egresado y su práctica profesional.

Se hace especial hincapié en desarrollar la autonomía del alumno para aprender y utilizar las herramientas involucradas en el desarrollo del proyecto.

EVALUACIÓN

FORMAS DE EVALUACIÓN

La forma de evaluación consta de tres partes: (1) seguimiento del desarrollo del proyecto (puntualidad en las entregas, desempeño, participación, calidad, claridad, metodologías usadas y justificación de las actividades realizadas); (2) evaluación, siguiendo los mismos lineamientos, de la entrega final; y (3) evaluación en el desempeño de los alumnos en la resolución de problemas.

- **Evaluaciones Parciales:** Entrega de cada una de las etapas del proyecto. En cada etapa se fundamentalmente la capacidad de resolución de problemas y de implementación de soluciones utilizando los contenidos introducidos. Como las etapas son incrementales en cada etapa se deben incluir las correcciones de la etapa anterior.

- **Evaluación Final:** Exposición y defensa del proyecto y un examen oral. Se evalúa fundamentalmente la adquisición de los conceptos fundamentales, su vinculación con el resto de la carrera y la capacidad de aplicarlos en diferentes situaciones.

Trabajos de Laboratorio:
Nº1: Análisis Léxico y Sintáctico
Nº2: Análisis Semántico
Nº3: Generación de Código Intermedio
Nº4: Optimizaciones sobre el código intermedio
Nº5: Generación de Código Objeto
Nº6: Optimizaciones sobre el Código Objeto

CONDICIONES PARA OBTENER LA REGULARIDAD
El alumno deberá aprobar el 60% de los Trabajos de Laboratorio.

CONDICIONES PARA OBTENER LA PROMOCIÓN
El alumno deberá:
- aprobar todos los Trabajos de Laboratorio con una nota no menor a 6 (seis), y obteniendo un promedio no menor a 7 (siete).
- aprobar un coloquio.

CORRELATIVIDADES
Para cursar:
- Algoritmos y Estructuras de Datos II (regularizada).
- Organización del Computador (regularizada).

Para rendir:
- Paradigmas de Programación (regularizada).