PROGRAMA de Parseo y Generación de Código

Carrera: Licenciatura en Informática

Asignatura: Parseo y Generación de Código

Núcleo al que pertenece: Avanzado

Profesor: Pablo Barenbaum

Asignaturas Correlativas: Lenguajes Formales y Autómatas – Características de Lenguajes de Programación

Objetivos:

1. Entender la arquitectura de herramientas existentes de análisis y síntesis de programas tales como intérpretes, compiladores y analizadores de código.
2. Concebir soluciones a nuevos problemas de análisis y síntesis de programas.
3. Desarrollar herramientas de análisis y síntesis de programas de pequeña escala para afianzar los conocimientos adquiridos.

Contenidos mínimos:

- Estructura de compiladores. Compilación vs. interpretación.
- Análisis léxico y sintáctico.
- Árboles de parsing y árboles de sintaxis abstracta.
- Análisis semántico.
- Generación de código.

Carga horaria semanal: 4 horas

Programa analítico:

Unidad 1

Estructura de un compilador. Fases de un compilador, compilación vs. interpretación.

Unidad 2

Análisis léxico. Lenguajes regulares, expresiones regulares, autómatas finitos.

Unidad 3

Análisis sintáctico. Gramáticas independientes del contexto, árboles de sintaxis, parsers descendentes y ascendentes, generadores de parsers.

Unidad 4

Interpretación. Intérpretes para lenguajes imperativos, funcionales, orientados a objetos.

Unidad 5

Análisis semántico. Tablas de símbolos, inferencia de tipos, unificación.

Unidad 6

Generación de código intermedio. Máquinas de pila, máquinas de registros, análisis y síntesis de atributos.

Unidad 7

Análisis estático. Grafos de flujo, análisis de flujo de datos.

Unidad 8

Optimización. Eliminación de código muerto, propagación de constantes, loop unrolling.

Unidad 9

Soporte en tiempo de ejecución. Representaciones tagged vs. tagless, registros de activación, garbage collection.

Unidad 10

Generación de código. Register allocation y spilling, selección de instrucciones.

Bibliografía (obligatoria y de consulta):

Bibliografía Obligatoria

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley, 2013.
- Andrew W. Appel. Modern Compiler Implementation in C. Cambridge University Press, 2004

Bibliografía de Consulta

- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. Introduction to Automata Theory, Languages and Computation. Pearson. 2013
- Simon Peyton Jones. The Implementation of Functional Programming Languages. Prentice Hall, 1987.
- Christian Queinnec. LiSP in Small Pieces. Cambridge University Press 2003.
- Guy L. Steele. RABBIT: A Compiler for SCHEME. MIT Technology, Cambridge, Mass., 1978
- Adele Goldberg, David Rodson. Smalltalk-80: The Language and Its Implementation. Longman Higher Education 1983
- Benjamin Pierce. Types and Programming Languages. The MIT Press, 1st edition, 2002
- Richard Jones, Antony Hosking, Eliot Moss. The Garbage Collection Handbook: The Art of Automatic Memory Management. Chapman and Hall/CRC, 1st edition. 2011

Organización de las clases:

Las clases tienen modalidad teórico-práctica. Su aspecto teórico consiste en motivar informalmente un problema computacional relacionado con el tema de la materia (construcción de un compilador), describir dicho problema de manera precisa usando el lenguaje lógico/matemático, y proponer una solución a ese problema. La solución típicamente tiene la forma de un algoritmo. Consideramos el comportamiento de esos algoritmos en términos de eficiencia temporal y espacial, y examinamos ejemplos usando diapositivas y pizarrón. El aspecto

práctico de las clases consiste en presentar problemas concretos relacionados en el ámbito de la materia, y discutir conjuntamente posibles soluciones que hacen uso de las herramientas teóricas presentadas.

Durante las clases prácticas también se presentan y se discuten los pormenores de la implementación de dos proyectos de software de pequeña escala que constituyen los trabajos prácticos de la materia.
Típicamente el primer TP consiste en implementar un analizador sintáctico para un lenguaje de programación cuya gramática está dada formalmente. El segundo TP consiste en implementar un generador de código cuyos lenguajes fuente y objeto se especifican como parte del enunciado del TP.

Guías de ejercitación:

Práctica 01 – Repaso de gramáticas independientes del contexto. Objetivos: usar gramáticas independientes del contexto como herramienta para describir lenguajes independientes del contexto; poder determinar si una gramática dada es o no ambigua.
Práctica 02 – Lenguajes regulares. Objetivos: usar autómatas finitos y expresiones regulares como herramientas para describir lenguajes regulares; convertir expresiones regulares en autómatas finitos y viceversa; determinar si un lenguaje es o no es regular recurriendo al lema de Pumping.
Práctica 03 – Normalización de gramáticas. Objetivos: aplicar técnicas para convertir gramáticas en otras equivalentes que cumplan con ciertas propiedades: sin símbolos anulables, sin ciclos, sin recursión a izquierda.
Práctica 04 – Análisis sintáctico descendente. Objetivos: entender el funcionamiento de los algoritmos de análisis sintáctico descendente, y en particular LL(1), aplicándolo sobre diversas gramáticas; aplicar técnicas de normalización de gramáticas para convertir una gramática en otra equivalente que sea LL(1).
Práctica 05 – Análisis sintáctico ascendente. Objetivos: entender el funcionamiento de los algoritmos de análisis sintáctico ascendente, y en particular LR(0), SLR, LR(1) y LALR, aplicándolos sobre diversas gramáticas; analizar y resolver conflictos *shift/reduce* y *reduce/reduce*.
Práctica 06 – Interpretación. Objetivos: escribir intérpretes para lenguajes de programación con diversas características, basados en entornos y usando las técnicas de pasaje de memorias y pasaje de continuaciones.
Práctica 07 – Tipos y unificación. Objetivos: entender los sistemas de tipos a través de ejercitación sobre una de sus manifestaciones más puras: el cálculo-lambda simplemente tipado; entender el algoritmo de unificación de primer orden; aplicar el algoritmo W de inferencia de tipos (Hindley-Milner) sobre ejemplos concretos.
Práctica 08 – Generación de código intermedio. Objetivos: entender diferentes formas de representar el código intermedio, en particular máquinas de pila y máquinas de registros, y cómo el código fuente de un lenguaje de programación de alto nivel se traduce a dicha representación.
Práctica 09 – Análisis de flujo de datos. Objetivos: aplicar técnicas de análisis estático, y en particular análisis de flujo de datos local (intraprocedural), para sobreaproximar o subaproximar estáticamente propiedades dinámicas de los programas.
Práctica 10 – Asignación de registros. Objetivos: estudiar posibles algoritmos, y en particular algoritmos basados en coloreo, destinados a asignar variables a registros y determinar cuándo debe hacerse el *spilling* a memoria principal.

Trabajos prácticos:

TP1 – Objetivos: aplicar las técnicas de análisis sintáctico estudiadas en la materia, implementando el analizador sintáctico de un lenguaje de programación de pequeña escala a partir de una gramática dada formalmente en notación BNF. En un trabajo típico, la implementación recibe un programa fuente a partir del cual debe generar un árbol de sintaxis abstracta, ya sea manualmente o usando alguna herramienta de generación de parsers como yacc o AntLR.

TP2 – Objetivos: aplicar las técnicas de generación de código estudiadas en la materia, implementando un traductor o compilador. En un trabajo típico, la implementación parte del árbol de sintaxis abstracta generado en el TP1 y debe convertirlo a alguna otra representación apta para su ejecución (ej. código en lenguaje ensamblador para alguna arquitectura física o virtual).

Modalidad de evaluación:

Los mecanismos de evaluación en modalidades libre y presencial de esta asignatura están reglamentados según los siguientes artículos del Régimen de estudios de la UNQ (Res. CS 201/18).
En la modalidad de libre, se evaluarán los contenidos de la asignatura con un examen escrito, un examen oral e instancias de evaluación similares a las realizadas en la modalidad presencial.

CRONOGRAMA TENTATIVO

Semana	Tema/unidad	Actividad*				Evaluación
		Teórico	Práctico			
			Res Prob.	Lab.	Otros Especificar	
1	Introducción a la materia. Intérpretes y compiladores. Introducción a los lenguajes formales.	x	x			
2	Lexers y autómatas.	x	x			
3	Lenguajes regulares y normalización de gramáticas.	x	x			
4	Análisis sintáctico descendente. [Presentación del TP1].	x	x			
5	Análisis sintáctico ascendente	x	x			
6	Repaso y consultas. [Entrega del TP1].		x	x		x
7	[Primer parcial].					x
8	Intérpretes.	x	x			
9	Inferencia de tipos.	x	x			
10	Generación de código intermedio. [Presentación del TP2].	x	x			
11	Análisis estático y optimización.	x	x			
12	Asignación de registros y manejo automático de memoria. [Entrega del TP2].	x	x	x		x
13	Repaso y consultas.		x	x		
14	[Segundo parcial].					x
15	[Recuperatorio del primer parcial].					x
16	[Recuperatorio del segundo parcial] [Última fecha reentrega de TPs].					x