Tema 1.1: Introducción

Manuel Soto Romero

Contenido

En este tema revisaremos:

1. **Contexto y definición de compilador**
2. **Conceptos auxiliares de la Teoría de Autómatas y Lenguajes Formales**

El curso de compiladores

- El curso de compiladores se encuentra ubicado dentro del séptimo semestre del plan de estudios de Ciencias de la Computación.

- Tiene como principal objetivo desmitificar a los compiladores desde un punto de vista teórico práctico.

Definición de compilador

Definición

Un **compilador** es un programa que traduce un lenguaje a otro. Toma como entrada un programa escrito en un ***lenguaje fuente*** y produce un programa equivalente escrito en un ***lenguaje objetivo***.

Programa fuente $\longrightarrow$ **Compilador** $\longrightarrow$ Programa objetivo

- El lenguaje fuente es un **lenguaje de alto nivel**.
- El lenguaje objetivo es **código objeto (máquina)**.

No tomar a la ligera la palabra programa pues no es para nada sencillo. La mayoría de computólogos nunca han escrito un compilador completo.

¿Qué necesito para diseñar un compilador?

- El curso te proporcionará todas las herramientas necesarias así como la experiencia práctica para diseñar y programar un compilador.

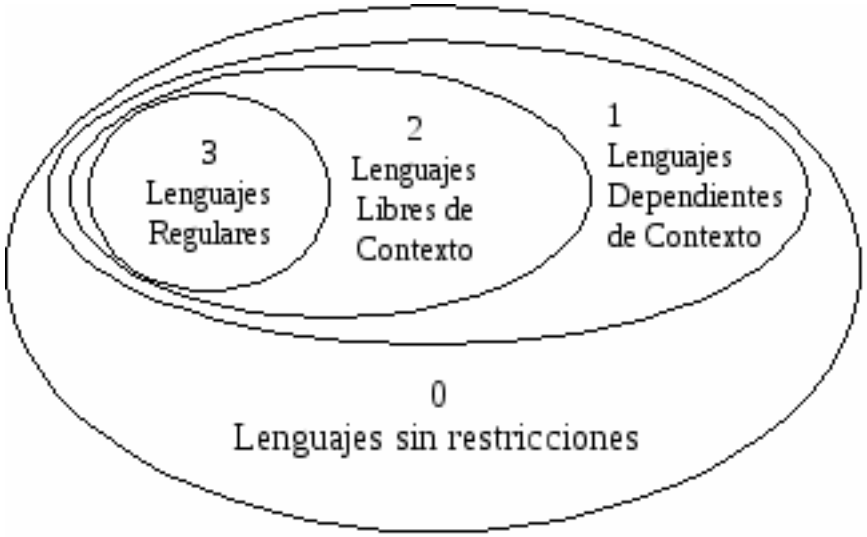
- Para alcanzar este objetivo será necesario estudiar algunas técnicas teóricas, principalmente las provenientes de la teoría de autómatas y lenguajes formales.

Probablemente estés familiarizadx con algunas de éstas. Sin embargo, para no asumir nada de ésto, repasaremos algunas así como su conexión con el diseño de compiladores.

La jerarquía de Chomsky

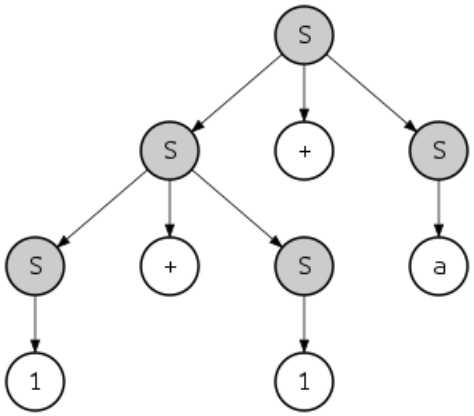
- Los estudios de Noam Chomsky condujeron a la clasificación de los lenguajes de acuerdo a la complejidad de sus **gramáticas** y la potencia de los algoritmos necesarios para reconocerlas.

- La **jerarquía de Chomsky** se compone de cuatro niveles de gramáticas, denominadas tipo 0, tipo 1, tipo 2 y tipo 3, cada una de las cuales es una especialización de la anterior.



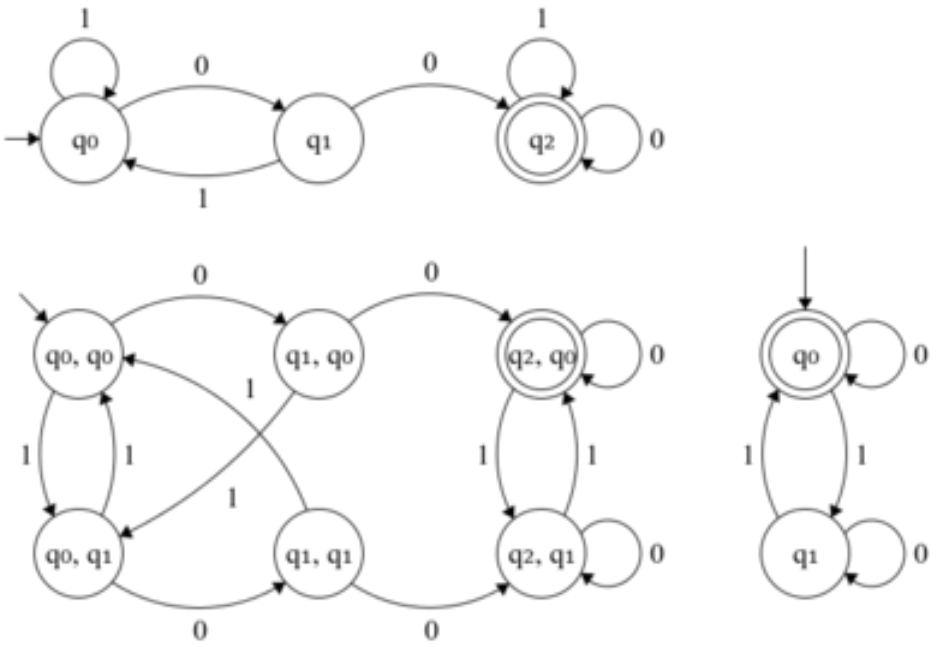
Gramáticas libres de contexto

- Las gramáticas de tipo 2, mejor conocidas como **gramáticas libres de contexto**, han demostrado ser las más útiles para especificar lenguajes de programación.
- El **problema de análisis sintáctico** consiste en encontrar algoritmos eficientes que reconozcan lenguajes libres de contexto.
- El estudio de este problema condujo al desarrollo de programas que automatizan su solución mediante los llamados **compiladores de compilador** hoy llamados **generadores de analizadores sintácticos**.
- El más conocido de estos programas es Yacc. Fue escrito por Steve Johnson en 1975 para el sistema Unix.



Autómatas finitos y expresiones regulares

- Estas técnicas, corresponden a las gramáticas tipo 3 y se encuentran relacionadas con las gramáticas libres de contexto.
- Su estudio condujo a métodos simbólicos para expresar la estructura de las palabras o **tokens** de un lenguaje de programación.
- Esto llevó al desarrollo de otra herramienta denominada **generador de analizadores léxicos**, cuyo representante más conocido es Lex.



¿Qué hay de la Teoría de Lenguajes de Programación?

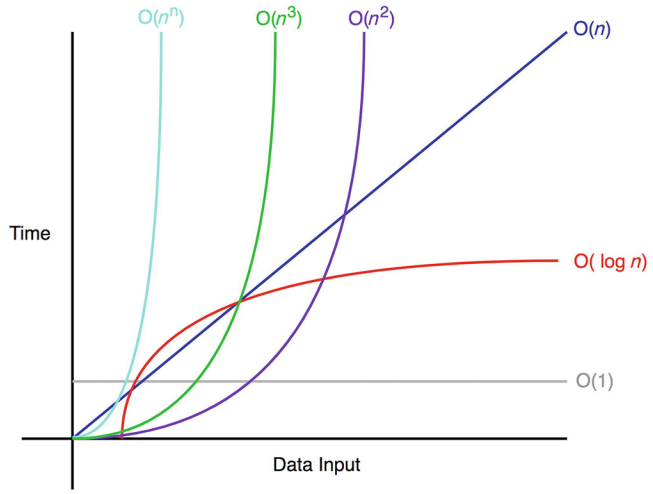
- Seguramente ya notaste que existe una interacción entre la estructura de un compilador y el diseño del lenguaje de programación que traduce.
- Revisaremos una gran cantidad de lenguajes de programación, sin embargo, únicamente hablaremos ocasionalmente de los aspectos de diseño de éstos.

Sugerencia: Repasa tus apuntes del curso de Lenguajes de Programación. Una liga con material de dicha materia se encuentra en la página del curso.

Técnicas de optimización

- Otro aspecto destacable en el área ha sido el desarrollo de métodos para la generación de código objeto eficaz.

- Estás técnicas suelen denominarse incorrectamente **técnicas de optimización**, pero en realidad deberían llamarse **técnicas de mejoramiento de código**, pues casi nunca producen un código objeto verdaderamente óptimo.



Otros avances

- Algoritmos para inferir y/o simplificar la información contenida en un programa, por ejemplo el algoritmo de unificación de tipos de Hindley-Milner.
- Integración con los llamados **entornos de desarrollo integrados** (IDE).

Conclusiones

- Podemos apreciar una cantidad significativa de actividades de investigación a lo largo de los años en varios ejes.
- A pesar de esto, los fundamentos en el diseño de compiladores no han cambiado mucho en los últimos años y son parte esencial de la enseñanza en los cursos de Ciencias de la Computación.
- Estudiaremos a detalle todos estos fundamentos con el fin de que puedas incursionarte en esta área o por si en algún momento los llegas a necesitar como herramienta para otro tipo de problemas.

Referencias

[1] Louden, K. C. (1997). *Compiler Construction: Principles and Practice*. Course Technology.