Autómatas y Lenguajes Formales
Nota 01. Introducción

Noé Salomón Hernández S.

1. Resumen del curso

Meta I

Se pretende desarrollar modelos matemáticos y abstractos de computación, tales como:

- máquinas de estados finitos,
- autómatas de pila, y
- máquinas de Turing.

Debemos notar que las computadoras reales, físicas, que tenemos corresponden al modelo más sencillo, máquinas de estados finitos. Sin embargo, el resto son abstracciones útiles de máquinas que podríamos construir en principio, dados los recursos limitados.

Meta II

Comprender las propiedades y expresividad de estos modelos, especialmente en términos de su habilidad de resolver problemas computacionales. Algunas preguntas que pretendemos resolver en este curso son:

- ¿Que problemas pueden ser resueltos?
- ¿Qué problemas pueden ser resueltos eficientemente?

2. Historia

1930 Alan Turing definió las máquinas más poderosas que cualquiera en existencia, o que incluso podamos imaginar – su meta fue establecer el límite entre lo que era y no era computable.

1940s/1950s En un intento de modelar *funciones cerebrales* los investigadores definieron las máquinas de estados finitos.

Finales de 1950s El lingüista Noam Chomsky comenzó el estudió de gramáticas formales.

Vemos una convergencia de todo esto a una teoría formal de las Ciencias de la Computación, con implicaciones filosóficas muy profundas así como aplicaciones prácticas (compiladores, búsquedas web, hardware, inteligencia artificial, diseño de algoritmos, ingeniería de software, etc.).

Culminación En la decada de 1970, Steve Cook extendió todo esto a la *Teoría de problemas NP-completos*, la cual separó una clase cuyos problemas:

- pueden ser resueltos, en principio;
- pero no pueden ser resueltos eficientemente, incluso dada la ley de Moore para el hardware.

En la práctica tales problemas son del mayor interés.

También se pretende en este curso enseñar al alumno como pensar de forma *precisa* y desarrollar habilidades de razonamiento de un modo preciso, formal y abstracto. Esto es lo que separa a programadores de científicos de la computación.

2.1. Autómata, computabilidad y complejidad

2.1.1. Teoría de la complexidad

Algunos problemas en computación son fáciles, y otros difíciles. Ordenar un arreglo es fácil. El problema de diseñar un calendarizador es más difícil,

¿Qué hace a algunos problemas computacionalmente difíciles y a otros fáciles? Notablemente, no conocemos la respuesta.

En uno de los más importantes logros de la teoría de la complexidad hasta el momento, los investigadores han descubierto un esquema elegante para clasificar problemas de acuerdo a su dificultad computacional. Usando este esquema, podemos mostrar un método para dar evidencia que ciertos problemas son computacionalmente difíciles.

2.1.2. Teoría de la computación

Durante la primera mitad del siglo veinte, matemáticos como Kurt, Gödel, Alan Turing y Alonzo Church descubrieron que ciertos problemas no pueden ser resueltos por computadoras.Un ejemplo es el problema de determinar si un enunciado matemático es verdadero o falso. No hay algoritmo para una computadora que pueda llevar a cabo esta tarea.

Entre las consecuencias de este resultado estuvieron el desarrollo de ideas relativas a modelos teóricos de computadoras que eventualmente guiaron la construcción de computadoras reales.

2.1.3. Teoría de autómatas

La teoría de autómatas lidia con las definiciones y propiedades de los modelos matemáticos de computación. Un modelo, llamado *autómata finito*, es usado en el procesamiento de texto, compiladores y diseño del hardware. Otro modelo, llamado *gramáticas libres de contexto*, es usado en lenguajes de programación e inteligencia artificial.

Las teorías de computación y complejidad requieren una definición precisa de una computadora. La teoría de autómatas permite practicar con definiciones formales de computación al introducir conceptos relevantes a otras áreas no teóricas de las ciencias de la computación.