Lenguajes de programación

Pedro O. Pérez M., PhD.

Implementación de métodos computacionales
Tecnológico de Monterrey

pperezm@tec.mx

02-2021

Contenido I

Contenido II

¿Qué es la teoría de la computación?

La teoría de la computación es el campo de las ciencias computacionales que trata con los problemas que pueden ser resueltos en un modelo de computación, usando un algoritmo, con qupe eficiencia se pueden resolver o en qué grado (soluciones aproximadas o precisas). Este campo se divide en tres ramas:

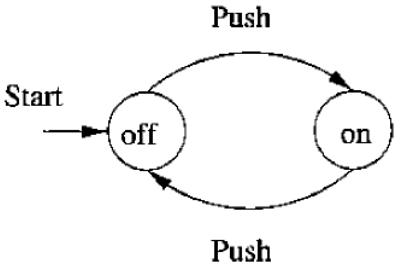
- Teoría de autómatas y lenguajes formales.
- Teoría de la computabilidad.
- Teoría de la complejidad computacional.

Los autómatas finitos son modelos computacionales muy útiles para varios tipos de hardware y software.

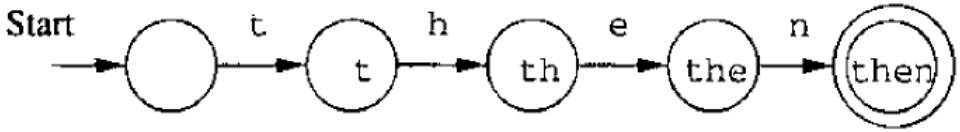
- Software para diseñar y verificar el comportamiento de circuitos digitales.
- El .ªnalizador léxico"de un compilador, el cual es el componente encargado de romper la entrada de texto en unidades lógicas, tales como identificadores, palabras reservadas, y símbolos de puntuación.
- Software para analizar grandes cantidad de texto, tales como colecciones de páginas Web, para encontrar ocurrencias de palabras, frases, u otros patrones.
- Software para verificar sistemas de todo tipo que tiene un número finito de distintos estados, tales como protocolos de comunicación o de intercambio seguro de información.

¿Qué es un autómata?

De manera informal, existe muchos sistemas o componentes que pueden ser representados por un número finito de estados. El proceso de un estado es recordar la porción relevante de la historia del sistema. Dado que puede llegar a existir una gran cantidad de estados, el sistema debe ser diseñado para recordar a que es importante. Y, pues que es una cantidad infinita de estados, es posible implementar un sistema con una cantidad fija de recursos. Un ejemplo, sería el autómata de un apagador.



En algunos casos, lo que debe recordar cada estado es más complejo que la opción de encendido/apagado. Por ejemplo, el autómata que reconoce la palabra reservada "then":



Alfabeto

Un alfabeto es un conjunto finito, no vacío de símbolos. Por convención, usaremos el símbolo Σ. Entre los ejemplos más comunes de alfabetos podemos encontrar:

1. $\Sigma = 0, 1$, alfabeto binario.
2. $\Sigma = a, b, c, ..., z$, el conjunto de todas las letras minúsculas.
3. El conjunto de todos los caracteres ASCII, o el conjunto de todos los caracteres imprimibles ASCII.

Cadenas (strings)

- Una cadena (o algunas veces palabras) es una secuencia finita de símbolos elegidos de algún alfabeto. Por ejemplo, 01101 es una cadena del alfabeto binarios $\Sigma = 0, 1$.
- La cadena vacía, ε, es un cadena con cero ocurrencia de símbolos y, por lo mismo, puede ser elegido de cualquier alfabeto.
- La longitud de una cadena es el número de posiciones para símbolos dentro de la misma. Por ejemplo, 01101 tiene una longitud de 5. La notación estándar para la longitud de una cadena w es $|w|$. Por ejemplo, $|011| = 3$ y $|\varepsilon| = 0|$.

Potencias de un alfabeto

- Si Σ es un alfabeto, podemos expresar el conjunto de todas las cadenas de una cierta longitud usando la notación exponencial. Con esto, podemos definir Σ^k como el conjunto de cadenas de longitud k, seleccionados de los símbolos de Σ.
- Es importante hacer notar que $\Sigma^0 = \{\varepsilon\}$, sin importar a que alfabeto nos refiramos.
- De tal forma que si $\Sigma = \{0, 1\}$, entonces:
  - $\Sigma^1 = \{0, 1\}$,
  - $\Sigma^2 = \{00, 01, 10, 11\}$,
  - $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

- El conjunto de todos las cadenas (de cualquier longitud) que existen sobre un alfabeto Σ, convencionalmente, se denomina Σ^*. Por ejemplo, si $\Sigma = \{0, 1\}$, $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, ...\}$. Este conjunto, de hecho, resulta ser: $\Sigma^* = \Sigma^0 \cap \Sigma^1 \cap \Sigma^2 ...$
- Algunas veces, quizás queremos excluir la cadena vacía del conjunto de cadenas. Para ello, usaremos Σ^+. Lo cual es igual a $\Sigma^+ = \Sigma^1 \cap \Sigma^2 \cap \Sigma^3 ...$ o $\Sigma^* = \Sigma^+ \cup \varepsilon$.

Concatenación de cadenas

- Sean x y y cadenas. Entonces, xy indica la concatenación de x y y. De forma más precisa, si x es una cadena compuesta por i símbolos $x = a_1 a_2 ... a_i$ y y es una cadena compuesta por j símbolos $y = b_1 b_2 ... b_j$, entonces xy es una cadena de longitud $i + j$, $xy = a_1 a_2 ... a_i b_1 b_2 ... b_j$.
- Para cualquier cadena w, la ecuación $\varepsilon w = w\varepsilon = w$ es verdad. Esto es porque ε es identidad para concatenación.

Lenguajes

- Un conjunto de cadenas, todas la cuales son seleccionadas de algún Σ^*, donde Σ es un alfabeto particular, es llamada lenguaje. Dicho de otra forma, si Σ es un alfabeto, y $L \subseteq \Sigma^*$, entonces L es un lenguaje sobre Σ. Por ejemplo,
  - El lenguaje Español, la colección legal de palabras españolas, es un conjunto de cadenas sobre el alfabeto que consiste de todas las letras.
  - El lenguaje C, o cualquier otro lenguaje de programación, es el conjunto de programas válidos que son un subconjunto de todas las posibles cadenas que pueden ser formadas sobre el conjunto de los caracteres ASCII.
- También podemos definir otros lenguajes:
  - El lenguaje de todas las cadenas consistentes de n 0's seguidos de n 1's para alguna $n \geq 0$: $\{\varepsilon, 01, 0011, 000111, ...\}$.

- También podemos definir otros lenguajes:
  - El lenguaje de todas las cadenas consistentes de n 0's seguidos de n 1's para alguna $n \geq 0$: $\{\varepsilon, 01, 0011, 000111, ...\}$.
  - El lenguaje de todas las cadenas que tiene un igual número de 0's y 1's: $\{\varepsilon, 01, 10, 0011, 1100, 0101, 1010, 1001, ...\}$.
  - El conjunto de números binarios cuyo valor es primo: $\{10, 11, 101, 111, 1011, ...\}$.
  - Σ^* es un lenguaje para cualquier alfabeto Σ.
  - $\emptyset$, el lenguaje vacío, es un lenguaje sobre cualquier alfabeto.
  - $\{\varepsilon\}$, lenguaje conformado por solo una cadena vacía, es, también, un lenguaje sobre cualquier alfabeto. Importante, $\emptyset \neq \{\varepsilon\}$.

Un poco de historia...

- De manera invidivual, revisa el documento: "A History of Computer Programming Languages" de Andrew Ferguson.
- En grupos pequeños, de tres personas, comenta los siguientes puntos:
  - ¿Porqué crees que debemos aprender varios lenguajes de programación?
  - ¿Podrían mencionar el aporte de algunos de los lenguajes mencionados?

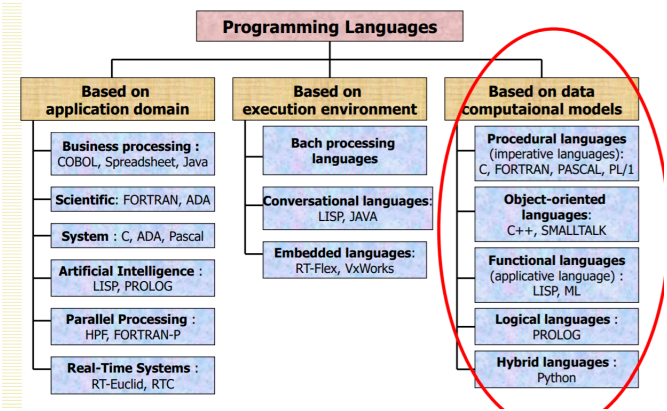
¿Porqué debemos aprender lenguajes de programación?

- Implementar algoritmos más efectivos mediante el aprendizaje de funciones, la comprensión de las implementaciones, el aumento de vocabulario como recursividad y los mecanismos de gestión de la memoria.
- Elegir mejores lenguajes de programación comparando varios lenguajes para ese propósito.
- Desarrollar mejores lenguajes de programación para mejores interfaces de usuario y funciones.

¿Cuáles son los atributos de buen lenguaje de programación?

- Ortogonalidad: Habilidad de combinar varias características de un idioma.
- Soporte para la abstracción: Construir para descartar patrones recurrentes.
- Natural para diversas aplicaciones: Para negocios, matemáticas, ciencia, juegos, etc.
- Fácil verificación y depuración del programa: Verificación formal y control de escritorio.
- Portabilidad y traducción: Grado mínimo de computación y complejidad espacial. Considerar varias arquitecturas.

Modelos de lenguajes de programación

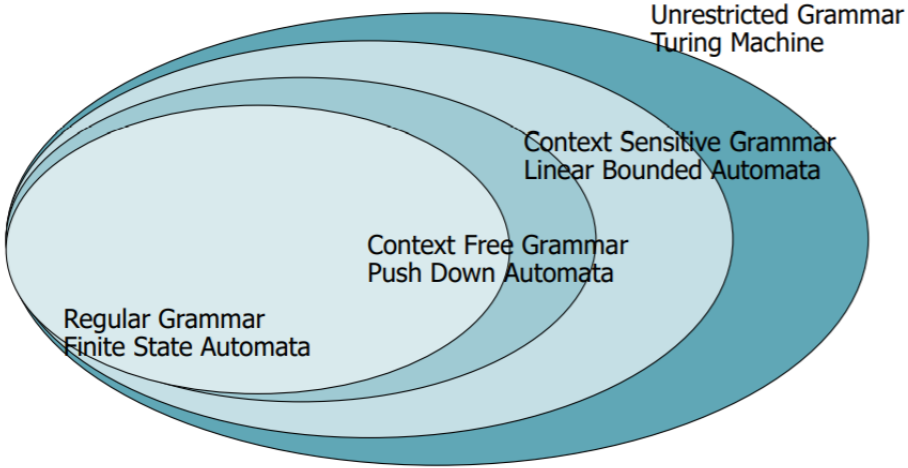


- Lenguajes imperativos
  - Un programa consiste de una secuencia de enunciados.
  - La ejecución de cada declaración hace que el intérprete cambie el estado del sistema.
  - Los sucesivos estados de la máquina conducen a la solución deseada.
  - Ejemplos: C, FORTRAN, Algol, PL/I, Pascal.
- Lenguajes basados en objetos
  - Los objetos complejos están diseñados como extensiones de objetivos simples.
  - Herencia de objetos simples.
  - Ejemplos: C++, Java, Smalltalk.

- Lenguajes aplicativos (lenguajes funcionales)
  - Realizar una función con sus argumentos para generar resultados.
  - Los resultados de una función pueden ser argumentos de otra función.
  - Ejemplos: LISP, ML, Scheme, Erlang y Clojure.
- Lenguajes basados en reglas (lenguajes lógicos)
  - Ejecutar una acción cuando se cumple una determinada condición de habilitación (predicado).
  - Construir una matriz o tabla de posibles condiciones y acciones apropiadas para pruebas o búsqueda en bases de datos.
  - El programa consta de hechos, reglas y consultas.
  - Ejemplos: Prolog
- Lenguajes híbridos: Python.

Jerarquía de Chomsky-Schützenberger

- La jerarquía de Chomsky-Schützenberger es la base formal para describir un lenguaje (natural o artificial). "How Complex is Natural Language? The Chomsky Hierarchy".


Unrestricted Grammar
Turing Machine
Context Sensitive Grammar
Linear Bounded Automata
Context Free Grammar
Push Down Automata
Regular Grammar
Finite State Automata

Definición de gramática

Una grámatica formal es un cuadrupla $G = (N, \Sigma, P, S)$ donde

- N es un conjunto finito de No Terminales.
- Σ es un conjunto finito de terminales y es disjunto de N.
- P es un conjunto finito de reglas de producción de la forma $w \epsilon (N \cup \Sigma)* \rightarrow w \epsilon (N \cup \Sigma)*$.
- $S \epsilon N$ es el símbolo inicial.

Tipo-0, Sin restricciones

- Los lenguajes definidos por gramáticas Tipo-0 son aceptadas por Maquinas de Turing.
- Las reglas son de la forma: $\alpha \rightarrow \beta$, donde α y β son cadenas arbitrarias sobre N y $\alpha \neq \varepsilon$.
- Ejemplos de producciones: Sab $\rightarrow$ ba, A $\rightarrow$ S.

Tipo-1, Sensibles al contexto

- Los lenguajes definidos por gramáticas Tipo-1 son aceptadas por Autómatas Delimitados Linealmente.
- Sintaxis de algunos lenguajes naturales (Alemán).
- Las reglas son de la forma: $\alpha A \beta \rightarrow \alpha B \beta$, $S \rightarrow \varepsilon$ donde A y $S \epsilon N$, α, β, $B \epsilon (N \cup \Sigma)*$ y $B \neq \varepsilon$.

Tipo-2, Libres de contexto

- Los lenguajes definidos por gramáticas Tipo-2 son aceptadas por Autómatas Push-Down.
- El lenguaje natural es casi enteramente definible por árboles sintácticos de Tipo-2.
- Las reglas son de la forma: $A \rightarrow \alpha$ donde $A \epsilon N$, $\alpha \epsilon (N \cup \Sigma)*$.

Tipo-3, Regulares

- Los lenguajes definidos por gramáticas Tipo-3 son aceptadas por Autómatas de Estados Finitos.
- La mayoría de la sintaxis de dialogo hablado informal.
- Las reglas son de la forma: $A \rightarrow \alpha$, $A \rightarrow \varepsilon$, $A \rightarrow \alpha B$ donde A, $B \epsilon N$, $\alpha \epsilon \Sigma$.

Si quieres conocer un poco más, revisa estos dos vídeos:

- "TYPES OF GRAMMAR- Type 0, Type 1, Type 2, Type 3 (CHOMSKY HIERARCHY)|| THEORY OF COMPUTATION".
- "Noam Chomsky, Fundamental Issues in Linguistics (April 2019 at MIT) - Lecture 1".