Universidad Nacional Autónoma de México
Facultad de Ciencias

Compiladores

Proyecto 1

Cruz Pineda Fernando
Espinosa Roque Rebeca
Guzmán Bucio Luis Antonio
Morales Martínez Edgar Jesús
Vázquez Dávila José Adolfo

10 de octubre de 2025

Índice general

0.1. Introducción

0.1.1. Objetivos

- Construir un lenguaje IMP a partir de la teoría de lenguajes formales y autómatas como aplicación práctica en la materia de compiladores.
- Implementar las transformaciones ER $\rightarrow$ AFNε $\rightarrow$ AFN $\rightarrow$ AFD $\rightarrow$ AFD$_{\text{min}}$ $\rightarrow$ MDD $\rightarrow$ función `lexer`.
- Validar el analizador léxico en casos de prueba de IMP.

0.2. Preliminares Formales

0.2.1. Gramática de IMP

El lenguaje IMP es un lenguaje imperativo simple que constituye un fragmento mínimo de lenguajes convencionales como C y Java. Su sintaxis se define mediante las siguientes producciones:

```
Aexp ::= n | x | Aexp + Aexp | Aexp - Aexp | Aexp * Aexp
Bexp ::= true | false | Aexp = Aexp | Aexp ≤ Aexp | not Bexp | Bexp and Bexp
Com  ::= skip | x := Aexp | Com; Com | if Bexp then Com else Com | while Bexp do Com
```

donde `Aexp` denota expresiones aritméticas, `Bexp` expresiones booleanas, y `Com` comandos.

0.2.2. Formalización

Lenguajes Formales

1 Definición** **Alfabeto, Cadena, Cerradura de Kleene & Lenguaje Formal

1. Un **alfabeto** Σ es un conjunto finito no vacío de símbolos.
2. Una **cadena** sobre un alfabeto Σ es una secuencia finita de símbolos de Σ. La cadena vacía se denota por ε.
3. La **cerradura de Kleene** de un alfabeto Σ, denotada por Σ^*, es el conjunto de todas las cadenas finitas que pueden formarse con símbolos de Σ, incluyendo la cadena vacía ε.
4. Un **lenguaje formal** L sobre un alfabeto Σ es un subconjunto de Σ^*, es decir, $L \subseteq \Sigma^*$.
5. Un **lexema** es una secuencia de caracteres en el programa fuente que coincide con el patrón de un token. Representa la instancia concreta de un token en el código fuente.
6. Un **token** es un par ordenado que consta de un nombre de token y un valor opcional. El nombre del token es una categoría sintáctica que representa una unidad léxica, mientras que el valor típicamente corresponde al lexema asociado. Formalmente, un token puede representarse como $\langle$nombre, valor$\rangle$, donde el nombre identifica la clase léxica (e.g., identificador, palabra reservada, operador) y el valor preserva el lexema correspondiente.
7. Un **patrón** es una descripción formal de la estructura que pueden tener los lexemas de un token. Los patrones se especifican típicamente mediante expresiones regulares.

Expresiones Regulares

Las expresiones regulares constituyen un formalismo para especificar lenguajes regulares. Proporcionan una notación algebraica que describe patrones de cadenas mediante la aplicación recursiva de operadores sobre un alfabeto base.

2 Definición ***Expresión Regular***

Sea Σ un alfabeto. Las **expresiones regulares** sobre Σ y los lenguajes que denotan se definen recursivamente como sigue:
Casos base:

- $\emptyset$ es una expresión regular y $L(\emptyset) = \emptyset$.
- ε es una expresión regular y $L(\varepsilon) = \{\varepsilon\}$.
- Para cada $a \in \Sigma$, a es una expresión regular y $L(a) = \{a\}$.

Paso inductivo: Sean E y F expresiones regulares sobre Σ. Entonces:

- $(E + F)$ es una expresión regular y $L(E + F) = L(E) \cup L(F)$.
- $(E \cdot F)$ es una expresión regular y $L(E \cdot F) = L(E)L(F) = \{uv \mid u \in L(E) \wedge v \in L(F)\}$.
- (E^*) es una expresión regular y $L(E^*) = (L(E))^* = \bigcup_{i=0}^{\infty}(L(E))^i$.
- (E) es una expresión regular y $L((E)) = L(E)$.

donde $L(E)$ denota el lenguaje descrito por la expresión regular E.

Teorema. 1 ***Equivalencia con Lenguajes Regulares***

Un lenguaje L es regular si y solo si existe una expresión regular E tal que $L = L(E)$.

Demostración. $\rightarrow$: Construimos inductivamente un autómata finito no determinista con transiciones ε (AFNε) a partir de la estructura de la expresión regular, aplicando las construcciones de Thompson para cada operador.
$\leftarrow$: Dado un autómata finito determinista que reconoce L, se aplica el algoritmo de eliminación de estados para obtener una expresión regular equivalente. Los detalles constructivos de ambas direcciones establecen la equivalencia expresiva entre expresiones regulares y autómatas finitos. □

Expresiones Regulares en IMP

Habiendo establecido la definición formal de expresiones regulares, procedemos a especificar la sintaxis léxica del lenguaje IMP. El alfabeto Σ sobre el cual se construyen los tokens de IMP se define como:

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, +, *, a, \ldots, z, A, \ldots, Z\}$$

donde $a, \ldots, z$ denota el alfabeto castellano en minúsculas y $A, \ldots, Z$ el alfabeto castellano en mayúsculas.

A continuación especificamos los tokens mediante expresiones regulares. Cada definición consiste en un identificador seguido de su expresión regular correspondiente.

Palabras reservadas:

$$\begin{aligned}
\texttt{not} &\rightarrow \texttt{not} \\
\texttt{and} &\rightarrow \texttt{and} \\
\texttt{if} &\rightarrow \texttt{if} \\
\texttt{then} &\rightarrow \texttt{then} \\
\texttt{else} &\rightarrow \texttt{else} \\
\texttt{skip} &\rightarrow \texttt{skip} \\
\texttt{while} &\rightarrow \texttt{while} \\
\texttt{do} &\rightarrow \texttt{do} \\
\texttt{true} &\rightarrow \texttt{true} \\
\texttt{false} &\rightarrow \texttt{false}
\end{aligned}$$

Enteros:

$$\begin{aligned} \texttt{dígito} &\to 0 \mid 1 \mid \cdots \mid 9 \\ \texttt{Z} &\to 1 \mid 2 \mid \cdots \mid 9 \\ \texttt{entero} &\to \texttt{Z} \cdot \texttt{dígito}^* + (- \cdot \texttt{Z} \cdot \texttt{dígito}^*) \end{aligned}$$

Operadores:
Para distinguir los operadores del metalenguaje de expresiones regulares de aquellos pertenecientes a IMP, subrayamos estos últimos.

$$\begin{aligned} \texttt{PLUS} &\to \underline{+} \\ \texttt{MINUS} &\to \underline{-} \\ \texttt{TIMES} &\to \underline{*} \\ \texttt{EQ} &\to \underline{=} \\ \texttt{LEQ} &\to \underline{\leq} \\ \texttt{SEMICOLON} &\to \underline{;} \\ \texttt{ASSIGN} &\to \underline{:=} \end{aligned}$$

Delimitadores:

$$\begin{aligned} \texttt{LPAREN} &\to \underline{(} \\ \texttt{RPAREN} &\to \underline{)} \end{aligned}$$

Espacios en blanco:

$$\begin{aligned} \texttt{espacio} &\to \underline{\sqcup} \mid \underline{\backslash\texttt{t}} \mid \underline{\backslash\texttt{n}} \\ \texttt{espacios} &\to \texttt{espacio}^+ \end{aligned}$$

Comentarios:

$$\texttt{comentario} \to \underline{//} \cdot \Sigma^* \cdot \underline{\backslash\texttt{n}}$$

Identificadores:

$$\begin{aligned} \texttt{letra} &\to A \mid B \mid \cdots \mid Z \mid a \mid b \mid \cdots \mid z \\ \texttt{id} &\to \texttt{letra} \cdot (\texttt{letra} \mid \texttt{dígito})^* \end{aligned}$$

Bibliografía

[1] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Pearson, 2006.

[2] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools* (2nd ed.). Pearson Education, 2006.

[3] M. S. Romero. *Compiladores: Unidad 2: Análisis Léxico*. Facultad de Ciencias, UNAM, 2026-1. https://lambdasspace.github.io/CMP/notas/cmp_n08.pdf

[4] Michael Sipser. *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning, 2012.

[5] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

[6] Tobias Nipkow, Gerwin Klein. *Concrete Semantics with Isabelle/HOL*. Springer, 2014.