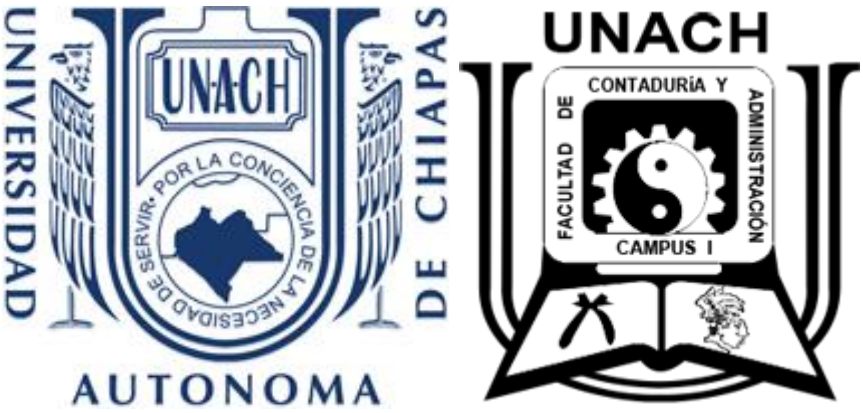
Universidad Autónoma de Chiapas



Facultad de Contaduría y Administración, Campus I

Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software

Compiladores

Actividad 1.1 – Investigar Analizador Léxico y Leguajes Regulares

Elaborado por:

Diego Arturo Anzá Díaz

6°M

Catedrático:

Dr. Luis Gutiérrez Alfaro

Tuxtla Gutiérrez, Chiapas A día domingo 18 de agosto de 2024

Introducción

El analizador léxico, también conocido como scanner o tokenizador, es la primera fase en el proceso de compilación o interpretación de un lenguaje de programación. Su función principal es leer el código fuente como una secuencia de caracteres y dividirlo en unidades significativas conocidas como tokens. Estos tokens representan las entidades básicas del lenguaje, como palabras clave, operadores, identificadores, literales, y símbolos de puntuación.

Durante el siguiente trabajo, en donde nos encargaremos de investigar acerca del analizador léxico y de los lenguajes regulares, daremos algunos conceptos básicos, así como también daremos ejemplos.

Funciones del analizador léxico

El analizador léxico es la primera etapa de un compilador, encargado de leer los caracteres de entrada y producir una secuencia de componentes léxicos que el analizador sintáctico utilizará para el análisis posterior. Este proceso se realiza mediante una interacción en la que el analizador léxico actúa como una subrutina del analizador sintáctico, respondiendo a las solicitudes para identificar y devolver el siguiente componente léxico.

Además de su función principal, el analizador léxico también lleva a cabo tareas adicionales, como eliminar comentarios y espacios en blanco del código fuente, y vincular los mensajes de error con las líneas específicas del código. En algunos compiladores, también se encarga de generar una copia del código fuente donde se señalan los errores y puede gestionar funciones de preprocesamiento, como la expansión de macros.

Componentes léxicos, patrones y lexema.

- Un token es un conjunto que incluye un nombre de token y, opcionalmente, un valor de atributo. El nombre del token representa un tipo específico de unidad léxica, como una palabra clave o un identificador, y es el símbolo que el analizador sintáctico procesa. Por lo general, el nombre del token se destaca en negrita.

- Un patrón es una descripción que define cómo pueden aparecer los lexemas asociados con un token. En el caso de una palabra clave, el patrón es simplemente la secuencia exacta de caracteres que la componen, mientras que para identificadores y otros tokens, el patrón puede ser más complejo.
- Un lexema es una secuencia de caracteres en el código fuente que coincide con el patrón de un token y que el analizador léxico reconoce como una instancia de ese token.

Lenguajes Regulares

Un lenguaje generado por una gramática regular consiste en cadenas formadas mediante la concatenación de símbolos, sin que exista dependencia entre diferentes partes de la cadena.

Una gramática regular, también conocida como de tipo 3, se define por tener reglas de producción que siguen la forma A→ uB o A→ u, donde u es una secuencia de símbolos terminales, y A y B son variables.

Lema de Bombeo para lenguajes regulares

El lema de bombeo se utiliza para demostrar que un lenguaje no es regular. Si un lenguaje no cumple con el lema, se puede concluir que no es regular. Sin embargo, si cumple con el lema, no se puede afirmar que sea regular, ya que el lema es una condición necesaria pero no suficiente. Para confirmar que un lenguaje es regular, es necesario encontrar un autómata finito determinista (AFD), una expresión regular o una gramática regular que lo demuestre.

Ejemplo

- Supongamos un n∈Nn \in \mathbb{N}n∈N arbitrario que cumple con las condiciones del lema de bombeo.
- Encuentra una palabra, que puede depender de nnn, con una longitud mayor o igual que nnn para la cual sea imposible dividirla de acuerdo con las condiciones del lema. Para demostrar que una palabra zzz no cumple con el lema de bombeo, se debe hacer lo siguiente:

- Supón una división arbitraria de zzz en z=uvwz = uvwz=uvw tal que ∣uv∣≤n|uv| \leq n∣uv∣≤n y ∣v∣=1|v| = 1∣v∣=1.
- Encuentra un i∈Ni \in \mathbb{N}i∈N tal que uviw∉Lu v^i w \notin Luviw∈/L.

Propiedades de cerradura de lenguajes regulares

La Proposición afirma que los lenguajes regulares son cerrados bajo los operadores de intersección, suma, producto, estrella y complemento. Es decir:

- La intersección de dos lenguajes regulares sigue siendo un lenguaje regular.
- La suma (unión) de dos lenguajes regulares también es regular.
- El producto de dos lenguajes regulares resulta en un lenguaje regular.
- La estrella de Kleene aplicada a un lenguaje regular produce un lenguaje regular.
- El complemento de un lenguaje regular es igualmente regular.
- En todos los casos, se demuestra que, al aplicar estos operadores a lenguajes regulares, el resultado sigue siendo un lenguaje regular.

Propiedades de decisión de lenguajes regulares

Este texto describe varias propiedades de los lenguajes regulares cuando se aplican diferentes operaciones entre dos lenguajes regulares L1 y L2:

1. **Unión**: La unión de L1 y L2 es regular porque se puede construir una expresión regular combinando las expresiones regulares de L1 y L2.

2. **Concatenación**: La concatenación de L1 y L2 es regular por la misma razón; se puede formar una expresión regular que los combine.

3. **Clausura de Kleene**: La clausura de Kleene de L1 es regular porque se puede construir una expresión regular aplicando la operación de estrella a la expresión regular de L1.

4. **Complemento**: El complemento de L1 es regular. Para construir un autómata que lo acepte, basta con invertir los estados finales y no finales del autómata que acepta L1.

5. **Intersección**: La intersección de L1 y L2L es regular. Usando las reglas de DeMorgan, la intersección se puede expresar como una combinación de complementos y uniones, que son operaciones que preservan la regularidad.

6. **Diferencia**: La diferencia entre L1 y L2 es regular, ya que se puede expresar como la intersección de L1 con el complemento de L2.

Además, se explica cómo construir un autómata finito determinista (AFD) que acepte la intersección de L1 y L2 mediante el producto cartesiano de los estados de los autómatas que aceptan cada uno de estos lenguajes. La construcción funciona de manera similar para autómatas no deterministas (AFND).

Ejemplo

[afdprod]

Obviamente la construcción funciona igual con autómatas finitos no-deterministas (AFND)

Proceso de determinación de equivalencias entre estados y lenguajes regulares

El proceso de determinación de equivalencias entre estados en lenguajes regulares es clave para simplificar autómatas finitos. Consiste en:

- Dividir estados: Se separan los estados en finales y no finales.
- Determinar equivalencia: Se comparan los estados para ver si, frente a cualquier cadena de entrada, llevan a resultados equivalentes. Si lo hacen, se consideran equivalentes.
- Combinar estados equivalentes: Los estados equivalentes se combinan en uno solo, reduciendo así el autómata.

Por ejemplo, en un autómata con cinco estados, tras identificar y combinar estados equivalentes, se reducen los estados a cuatro, manteniendo el mismo lenguaje aceptado.

Proceso de minimización de DFA

La minimización de un autómata finito determinista (DFA) es el proceso de transformar un DFA en otro con el menor número posible de estados, manteniendo el mismo lenguaje aceptado. Los pasos clave incluyen:

1. **Identificación de Estados Inalcanzables**: Se eliminan los estados que no pueden ser alcanzados desde el estado inicial, ya que no contribuyen al reconocimiento del lenguaje.

2. **Determinación de Estados Distinguibles**: Se comparan los estados para ver si, a partir de cada estado, todas las posibles cadenas de entrada llevan a estados equivalentes. Los estados que no son distinguibles se fusionan en uno solo.

3. **Construcción del DFA Mínimo**: Se crea un nuevo DFA con los estados minimizados, asegurando que produzca las mismas salidas para las mismas entradas que el DFA original.

Un DFA mínimo es aquel que no se puede reducir más, es decir, tiene el menor número de estados posibles para reconocer el mismo lenguaje.

Conclusión

El analizador léxico es fundamental para la primera fase de un compilador, transformando el código fuente en una secuencia de componentes léxicos. Los lenguajes regulares, generados por gramáticas regulares y definidos por operaciones de cerradura, presentan propiedades que se pueden explotar para simplificar y analizar autómatas. La minimización de DFA es crucial para optimizar los autómatas finitos deterministas, garantizando que se reconozca el mismo lenguaje con el menor número de estados posible. La comprensión y aplicación de estos conceptos son esenciales para el diseño y la optimización de compiladores y otros sistemas que procesan lenguajes formales.

GitHub

https://github.com/ArturitoAnDi/Compiladores

Bibliografía

Diagrama de estados. (s. f.). MATLAB & Simulink. https://la.mathworks.com/discovery/state-diagram.html#:~:text=Los%20diagramas%20de%20estados%20se,o%20distintos%20modos%20de%20funcionamiento

2.2. Componentes Léxicos, Patrones y Lexemas. (s. f.). http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/22_componentes_lxicos_patrones_y_lexemas.html

LENGUAJES REGULARES. (s. f.). https://ccia.ugr.es/~rosa/tutormc/teoria/lenguajesregulares2.html#:~:text=LENGUAJES%20REGULARES&text=DEFINICI%C3%93N%20LENGUAJE%20REGULAR%3A,otra%20parte%20de%20la%20cadena.

LEMA DE BOMBEO PARA LOS LENGUAJES REGULARES. (s. f.). https://ccia.ugr.es/~rosa/tutormc/teoria/LEMA%20DE%20BOMBEO.htm

Propiedades de cerradura. (s. f.). https://delta.cs.cinvestav.mx/~gmorales/ta/node70.html

Propiedades de lenguajes regulares. (s. f.). https://formella.webs.uvigo.es/doc/talf05/talf/node38.html

Academy, E. (2023, 2 agosto). *Explicar la equivalencia entre lenguajes regulares y expresiones regulares. - Academia EITCA*. EITCA Academy. https://es.eitca.org/cybersecurity/eitc-is-cctf-computational-complexity-theory-fundamentals/regular-languages/equivalence-of-regular-expressions-and-regular-languages/examination-review-equivalence-of-regular-expressions-and-regular-languages/explain-the-equivalence-between-regular-languages-and-regular-expressions/

Minimización de DFA. (s. f.). https://es.wikibrief.org/wiki/DFA_minimization