INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Nombre: Cruz López Adrián

Grupo: 3CM15

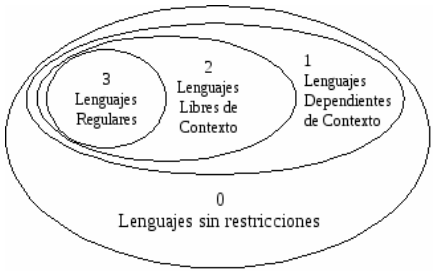
Asignatura: Compiladores

Profesor: Roberto Tecla Parra

Actividad: Tarea 1, Jerarquía de Chomsky, YACC y LEX

Fecha: 27/08/2021

Jerarquía de Chomsky



Jerarquía de Chomsky. En Lingüística, Matemáticas e Informática, se menciona el sistema de definición jerárquica diseñado por Noam Chomsky en el MIT en 1956, que divide matemáticamente los lenguajes formales en cuatro categorías enumeradas de 0 a 3 y mecanismos formales, como gramática formal, expresiones y autómatas, utilizados para identificar cada tipo. También se le llama clasificación de Chomsky o jerarquía matemática del lenguaje.

Gramática tipo 0

Llamaremos gramática de tipo 0 a toda G = (V, Σ, Q_0, P) gramática tal que todas las producciones de P son del tipo:

$$\gamma \rightarrow \omega, \text{ donde } \gamma, \omega \in (\Sigma \cup V)^*$$

También llamada gramática no restringida o recursivamente enumerable, es un lenguaje formal para el cual existe una máquina de Turing que acepta y se detiene con cualquier cadena del lenguaje, pero que puede parar y rechazar, o bien iterar indefinidamente, con una cadena que no pertenece al lenguaje, en contraposición a los lenguajes recursivos en cuyo caso se requiere que la máquina de Turing pare en todos los casos. Las reglas de derivación son de la forma $\alpha \rightarrow \beta$.

Siendo $\alpha \in (NT \cup T)^+ y \ \beta \ (NT \cup T)^*$, es decir la única restricción es que no puede haber reglas de la forma $\lambda \rightarrow \beta$ donde λ es la cadena vacía.

Los lenguajes recursivamente enumerables con las siguientes operaciones. Esto es, si L y P son dos lenguajes recursivamente e numerables también:

Cierre estrella L* de L, concatenación LP de L y P, la unión de L ∪ P, la intersección L ∩ P de L y P.

Ejemplo:

Considere la gramática G = ({a, b}, {A, B, C}, A, P), donde:

A:: = aABC | abC
CB:: = BC
bB:: = bb
bC:: =

Gramática tipo 1

Llamaremos gramática sensible al contexto a toda G = (V, Σ, Q_0, P) gramática tal que todas las producciones de P son del tipo siguiente:

γAδ , $A \mapsto$γωδ, donde A ∈ V y ω, γ, δ ∈ ,∈ $(\Sigma \cup V)^*$, ω ≠ λ

Es equivalente a una máquina de Turing no determinista linealmente acotada, también llamado autómata linealmente acotado. Se trata de una máquina de Turing no determinista con una cinta de solo kn posiciones, donde n es el tamaño de la entrada k es la constante asociada a la máquina. Esto significa que cada lenguaje formal que puede ser decidido por una máquina es el lenguaje sensible al contexto en ellas las reglas de producción son de la forma:

$$\boldsymbol{aA\beta \rightarrow a\gamma\beta}$$

Estas gramáticas tienen las reglas de la forma $aA\beta \rightarrow a\gamma\beta$ con A a un no terminal y a, β y γ cadenas terminales y no terminales las cadenas a, β pueden ser vacías, pero $\boldsymbol{\gamma}$ no puede serlo. La regala $\boldsymbol{S} \rightarrow \in$ está permitida si S no aparece en la parte derecha ninguna regla. Los lenguajes descritos por estadísticas son exactamente todos aquellos lenguajes conocidos por una máquina de Turing determinista cuya cinta de memoria estaba acotada por un cierto número entero de veces sobre la longitud de entrada, también conocidas como autónoma linealmente acotados.

Siendo $a \in NT;\ a, \beta \in (NT \cup T)^*$ y $\gamma\ (VN \cup VT)^+$

Estas gramáticas se llaman sensibles al contexto pues se puede volver a reemplazar A por … siempre que estén en el contexto a … β

La unión, intersección y concatenación de dos lenguajes sensibles al contexto es un lenguaje sensible al contexto. El complemento de un lenguaje sensible al contexto es en sí mismo sensible al contexto. Cada gramática libre de contexto es un lenguaje sensible al contexto.

Ejemplo:

La gramática $G = (\{a.b\}, \{A, S\}, S, P)$ donde P son las producciones siguientes:

$S \rightarrow aS$
$S \rightarrow aA$
$A \rightarrow bA$
$A \rightarrow b$

Gramática tipo 2

Llamaremos gramática libre de contexto a toda G = (V, Σ, Q_0, P) gramática tal que todas las producciones de P son del tipo siguiente: A 7→ ω, donde A ∈ V y ω ∈ $(\Sigma \cup V)^*$

Sus reglas tan solo admiten tener un símbolo no terminal en su parte izquierda es decir son de la forma $A \rightarrow a$ Siendo $A \in NT\ y\ a \in (NT \cup T)^+$.

Si cada regla se representa con un par ordenado (A, a) el conjunto P es un subconjunto del conjunto cartesiano $NT\ x\ (\{T \cup NT\})^+$. La denominación contexto libre se debe a que puede cambiar A por a independientemente del contexto que parezca A.

Las reglas son de la forma $A \rightarrow a$ con A un no terminal y a una cadena de terminales y no terminales. Estos lenguajes son aquellos que pueden ser reconocidos por un autómata con pila.

El término libre de contexto se refiere al hecho de que el no terminal A puede ser sustituido por a sin tener en cuenta el contexto en el que ocurra. Un lenguaje formal es libre de contexto si hay una gramática libre de contexto que lo genera.

Una de las definiciones alternativas y equivalentes del lenguaje libre de contexto emplea autómatas no deterministas, un lenguaje puede ser también modelado con un conjunto de todas las secuencias de terminales aceptados por la gramática, la unión y concatenación de dos lenguajes y libres de contexto, el inverso de un lenguaje libre de contexto, los lenguajes regulares, la intersección de un lenguaje libre de contexto y un lenguaje regular es libre de contexto, la intersección no tiene por que serlo, para demostrar que un lenguaje dado no es libre de contexto, se puede emplear el Lema del bombeo para lenguajes libres de contexto.

Ejemplo:

La gramática $G = (\{S, A, B\}, \{a, b\}, S, P)$ cuyas producciones P son las producciones siguientes:

$S \rightarrow aB$ $\quad$ $A \rightarrow bAA$
$S \rightarrow bA$ $\quad$ $B \rightarrow b$
$A \rightarrow a$ $\quad$ $B \rightarrow bS$
$A \rightarrow aS$ $\quad$ $B \rightarrow aBB$

Gramática tipo 3

Llamaremos gramática lineal por la izquierda a toda G := (V, Σ, Q_0, P) gramática tal que todas las producciones de P son de uno de los dos tipos siguientes:

A ⟼ a, donde A ∈ V y a ∈ Σ ∪ {λ}.

A ⟼ aB, donde A, B ∈ V y a ∈ Σ ∪ {λ}

Estas gramáticas se restringen a aquellas reglas que tienen en la parte izquierda un no terminal, y en la parte derecha un solo terminal, posiblemente seguirá siendo de un no terminal. La regla $S \rightarrow \in$ también está permitida si S no aparece en la parte derecha de ninguna regla. Estos lenguajes son aquellos que pueden ser aceptados por un autómata finito. También esta familia de lenguajes puede ser obtenidas por medio de expresiones regulares es decir de la forma:

$A \rightarrow aB$
$A \rightarrow a$

Dónde $A, B \in NT\ y\ a\ \in T$. Satisface las siguientes propiedades los lenguajes más sencillos que se consideran son los lenguajes regulares, es decir, los que se pueden generar a partir del lenguaje los básicos con la aplicación de las operaciones de unión concatenación y * de Kleene un número finito de veces.

Puede ser reconocido como un autómata finito determinista, un autómata finito no determinista, un autómata de pila, un autómata finito alterno, una máquina de Turing de solo lectura. Es generado por, una gramática regular, una gramática de prefijos. Es descrito por una expresión regular.

Ejemplo:

La gramática $G = (\{a, b\}, \{A, S\}, S, P)$ donde P son las producciones siguientes:

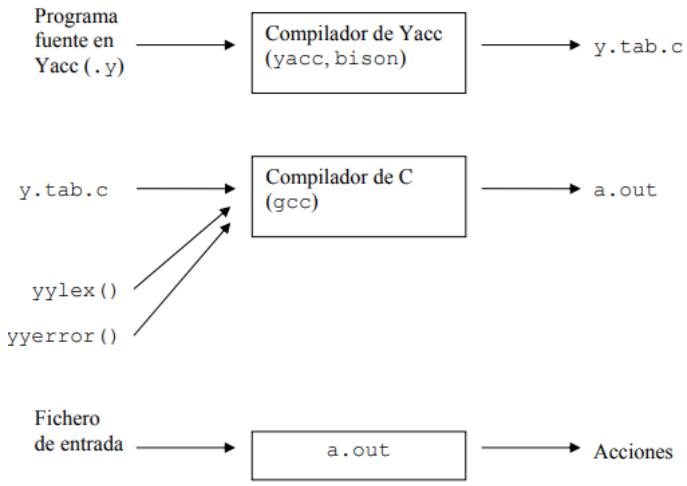
$S \rightarrow aS$
$S \rightarrow aA$
$S \rightarrow bA$
$S \rightarrow b$

YACC

De las siglas en inglés Yet Another Compiler Compiler, YACC un programa que parte del archivo escrito en el lenguaje específico Lenguaje de Descripción de Gramática y genera el código en C, C++ o Pascal. Permite un lenguaje propio para automatizar tareas repetitivas que, de otra manera, tendrían que hacerse manualmente.

Funcionamiento de YACC

A partir de un fichero fuente en YACC, se genera un fichero fuente en C que contiene el analizador sintáctico. Sin embargo, un analizador sintáctico de YACC no puede funcionar por sí solo, sino que necesita un analizador léxico externo para funcionar. Dicho de otra manera, el archivo fuente en C que genera YACC contiene llamadas a una función yyLEX( ) que debe estar definida y debe devolver el tipo de LEXema encontrado. Además, es necesario incorporar también una función yyerror( ), que será invocada cuando el analizador sintáctico encuentre un símbolo que no encaja en la gramática.



El lenguaje YACC

Esquema general

Un programa fuente de YACC se parece bastante a uno de LEX. La diferencia principal está en la sección de reglas, que en vez de expresiones regulares contiene las reglas de la gramática:

<sección de definiciones>
%%
<sección de reglas>
%%
<sección de rutinas>

- Sólo la segunda sección es obligatoria, y no debe estar vacía. Esto quiere decir que el mínimo programa en YACC es:

  %%
  regla gramatical acción en C

- La sección de declaraciones puede incluir varias cosas, su función principal es declarar los símbolos terminales de la gramática mediante la directriz %token. Todo lo que no sea un terminal, será considerado un símbolo no terminal, y por tanto debe haber una regla para él:

    %token IF, ELSE, LLAVE_AB, LLAVE_CE, IDENT

- La sección de reglas contiene la gramática en sí. "Componentes" es una combinación de terminales y no terminales que describe al no terminal de la izquierda de la regla:

    no_terminal: componentes

- La sección de rutinas no define por defecto las funciones main( ), yyLEX( ) e yyerror( ), así que se deben incluir aquí, o bien en otro fichero que se enlazará en la fase final de la compilación.

YACC genera una función llamada yyparse( ) que contiene el analizador sintáctico. Esta función se comporta como una máquina de estados cuya misión es intentar reducir todo el fichero de entrada al símbolo inicial de la gramática.

Sección de reglas

Las reglas deben estar dadas de forma que la parte izquierda conste de un único símbolo no terminal, y la parte derecha indique la combinación de terminales y no terminales de que puede estar compuesto. Además, toda regla debe incluir una instrucción en C que se ejecutará en cuanto YACC consiga encontrar los componentes del símbolo resultado:

símbolo_result: componentes intrucción_en_C

- El símbolo resultado debe estar situado en la primera posición de la línea; es decir, que no puede haber espacios antes del símbolo resultado.
- Los componentes son una combinación de terminales y no terminales separados por espacios en blanco.
- La acción puede ser una sola sentencia de C, o una sentencia compuesta, encerrada entre llaves.

LEX

Es un generador de programas diseñado para el proceso léxico de cadenas de caracteres de input. El programa acepta una especificación, orientada a resolver un problema de alto nivel para comparar literales de caracteres, y produce un programa C que reconoce expresiones regulares. Estas expresiones las especifica

el usuario en las especificaciones fuente que se le dan al LEX. El código LEX reconoce estas expresiones en una cadena de input y divide este input en cadenas de caracteres que coinciden con las expresiones. En los bordes entre los literales, se ejecutan las secciones de programas proporcionados por el usuario. El fichero fuente LEX asocia las expresiones regulares y los fragmentos de programas. Puesto que cada expresión aparece en el input del programa escrito por el LEX, se ejecuta el fragmento correspondiente.

El LEX convierte las expresiones y acciones del usuario en un programa C llamado yyLEX. El programa yyLEX reconoce expresiones en un y lleva a cabo las acciones especificadas para cada expresión a medida que se va detectando.

Formato fuente del LEX

El formato general de la fuente LEX es:

{definiciones}
%%
{órdenes}
%%
{subrutinas del usuario}

donde las definiciones y las subrutinas del usuario se omiten a menudo. El segundo %% es opcional, pero el primero se requiere para marcar el principio de las órdenes. El programa LEX mínimo absoluto es, por lo tanto %% (sin definiciones, ni órdenes) lo cual se traduce en un programa que copia el input en el output sin variar.

Expresiones del LEX

Una expresión especifica un conjunto de literales que se van a comparar. Esta contiene caracteres de texto y caracteres operador. Las letras del alfabeto y los dígitos son siempre caracteres de texto.

Los caracteres operador son:

“ \ [ ] ^ - ? . * + | ( ) $ / { } % < >

Si cualquiera de estos caracteres se va a usar literalmente, es necesario incluirlos individualmente entre caracteres barra invertida (\ ) o como un grupo dentro de comillas (“).

El operador comillas (“) indica que siempre que esté incluido dentro de un par de comillas se va a tomar como un carácter de texto.

Llamar al LEX

El programa LEX fuente tiene que ser convertido en un programa regenerado en el lenguaje de propósito general. Entonces este programa tiene que ser compilado y cargado, normalmente con una librería de subrutinas LEX.

El programa resultante se pone en el fichero usual a.out para ser ejecutado posteriormente. Aunque las rutinas I/O por defecto del LEX usan la librería estándar C, el autómata del LEX no lo hace. Si se especifican las versiones privadas de input, output, y unput, la librería se puede evitar.