Compiladores 24-2
Análisis Sintáctico: parsers top-down LL(k)

Lourdes del Carmen González Huesca
luglzhuesca@ciencias.unam.mx

Facultad de Ciencias, UNAM

6 marzo 2024

Análisis sintáctico

Top-down parsing

Para obtener un parser top-down hay que considerar que la gramática del lenguaje pertenezca a la clase LL.

Análisis sintáctico

Top-down parsing

Para obtener un parser top-down hay que considerar que la gramática del lenguaje pertenezca a la clase LL.

Para decidir si una gramática está en la clase **LL(1)**:

1. revisar si la gramática tiene recursión izquierda
   si la tiene hay que eliminarla, y si no se puede eliminar entonces no existe un parser top-down para esa gramática

Análisis sintáctico

Top-down parsing

Para obtener un parser top-down hay que considerar que la gramática del lenguaje pertenezca a la clase LL.

Para decidir si una gramática está en la clase **LL(1)**:

1. revisar si la gramática tiene recursión izquierda
   si la tiene hay que eliminarla, y si no se puede eliminar entonces no existe un parser top-down para esa gramática
2. revisar si los conjuntos de símbolos de derivación (DS) de las producciones con la misma parte izquierda (variable) son ajenos

$$DS(A \to \alpha) \quad = \quad \{a \in \Sigma \mid \quad a \in \textsc{First}(A) \text{ o si } \alpha \to^\star a \text{ entonces } a \in \textsc{Follow}(A)\}$$

   si lo son entonces crear la tabla de parsing; si no, entonces calcular los factores comunes (intersección de los DS) y aplicar una transformación para eliminarlos, repetir el proceso hasta que los conjuntos sean ajenos: si la transformación falla entonces no se puede definir un parser top-down

Análisis sintáctico

Top-down parsing

- Top-down parsing para gramáticas tipo **LL** (*scan input from left to right & left-most derivation*).
- Un parser **LL(k)** es uno también llamado predictivo, en donde se revisan *k* tokens por adelantado
  para una variable o un símbolo no-terminal, el símbolo leído por adelantado determina de forma única la producción a aplicar

Análisis sintáctico

Top-down parsing

- Top-down parsing para gramáticas tipo **LL** (*scan input from left to right & left-most derivation*).
- Un parser **LL(k)** es uno también llamado predictivo, en donde se revisan *k* tokens por adelantado
  para una variable o un símbolo no-terminal, el símbolo leído por adelantado determina de forma única la producción a aplicar
- Estudiaremos la clase de parsers fuertes **LL**(k) (Strong LL **SLL**(k)) que son aquellos analizadores donde los símbolos leídos por adelantado son suficientes para seleccionar la producción correcta en la gramática para construir el parse tree.

Funciones auxiliares

k-prefijos y k-concatenación

Dado un alfabeto Σ y una palabra $\omega = a_1 a_2 \ldots a_n$ con $a_i \in \Sigma$ se define

- k-prefijo de ω (truncar w al tamaño k)

$$\omega|_k = \begin{cases} a_1 a_2 \ldots a_n & \text{si} \quad n \leqslant k \\ a_1 a_2 \ldots a_k & \text{en otro caso} \end{cases}$$

Funciones auxiliares

k-prefijos y k-concatenación

Dado un alfabeto Σ y una palabra $\omega = a_1 a_2 \dots a_n$ con $a_i \in \Sigma$ se define

- k-prefijo de ω (truncar w al tamaño k)

$$\omega|_k = \begin{cases} a_1 a_2 \dots a_n & \text{si} \quad n \leqslant k \\ a_1 a_2 \dots a_k & \text{en otro caso} \end{cases}$$

- k-concatenación $\odot_k : \Sigma^\star \times \Sigma^\star \to \Sigma^{\leqslant k}$, con $\Sigma^{\leqslant k} = \bigcup_{i=0}^{k} \Sigma^i$, como

$$a \odot_k b = (ab)|_k$$

Funciones auxiliares

first y follow para k

Definición (Función $\textsc{First}_k$)

Función que calcula el conjunto de prefijos de tamaño k que se pueden derivadar de α:

$$\textsc{First}_k(\alpha) = \{\omega|_k \mid \alpha \rightarrow^\star \omega\}$$

$$\textsc{First}_k(A) = \bigcup\{\textsc{First}_k(A_1) \odot_k \cdots \odot_k \textsc{First}_k(A_n) \mid A \rightarrow A_1 A_2 \ldots A_n,\ A_i \ var\}$$

Definición (Función $\textsc{Follow}$)

La función $\textsc{Follow}_k$ para un símbolo no-terminal X calcula el conjunto de símbolos terminales de a lo más tamaño k que pueden seguir directamente a la variable X:

$$\textsc{Follow}_k(X) = \{\omega \in \Sigma^\star \mid S \rightarrow^\star vX\gamma \text{ y } \omega \in \textsc{First}_k(\gamma\#)\}$$

Gramática **SLL**(k)

Decimos que una gramática pertenece a la clase **SLL(k)** si para cualquier símbolo no-terminal con más de dos reglas de producción $A \to \beta \mid \gamma$ con $\beta \neq \gamma$ siempre sucede que

$$\textsc{First}_k(\beta) \odot_k \textsc{Follow}_k(A) \cap \textsc{First}_k(\gamma) \odot_k \textsc{Follow}_k(A) = \varnothing$$

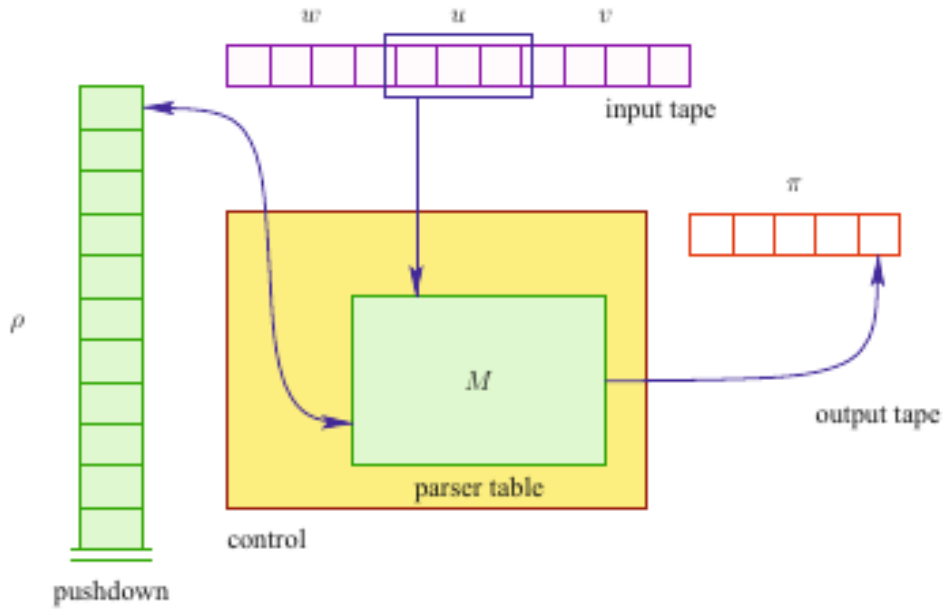
Gramática **SLL**(k)

Decimos que una gramática pertenece a la clase **SLL(k)** si para cualquier símbolo no-terminal con más de dos reglas de producción $A \to \beta \mid \gamma$ con $\beta \neq \gamma$ siempre sucede que

$$\textsc{First}_k(\beta) \odot_k \textsc{Follow}_k(A) \cap \textsc{First}_k(\gamma) \odot_k \textsc{Follow}_k(A) = \varnothing$$

Obs. Toda gramática **LL(1)** es una gramática **SLL(1)**. Pero para $k > 1$, una gramática **LL(k)** no necesariamente es también una **SLL(k)**, dado que el conjunto $\textsc{Follow}_k(A)$ contiene todas las palabras que se pueden generar desde A hacia la izquierda.

Parsers **SLL**(k)



- Usa una tabla de predicción o de parsing para seleccionar la generación de una subcadena.
- El autómata compara si el símbolo de entrada coincide con el tope de la pila o expande el símbolo no-terminal

Parsers **SLL**(k)

tabla de parsing

La tabla $M[X, \omega]$ tiene por renglones las variables de la gramática y como columnas cadenas de terminales de longitud k, almacena producciones que se determinan de la siguiente forma:

Sean $Y \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_r$ las reglas correspondientes a la variable Y.

Dado que los conjuntos $\textsc{First}_k(\alpha_i) \odot_k \textsc{Follow}_k(Y)$ son disjuntos entonces

- para cada $\omega \in \textsc{First}_k(\alpha_1) \odot_k \textsc{Follow}_k(Y) \cup \textsc{First}_k(\alpha_2) \odot_k \textsc{Follow}_k(Y) \cup \cdots \cup \textsc{First}_k(\alpha_r) \odot_k \textsc{Follow}_k(Y)$ se tiene que

$$M[Y, \omega] = \alpha_i \quad \text{si y sólo si} \quad \omega \in \textsc{First}_k(\alpha_i) \odot_k \textsc{Follow}_k(Y)$$

- en otro caso se deja la entrada de la tabla vacía o se incluye una rutina para manejar el error

Parsers **SLL**(k)

tabla de parsing

La tabla $M[X, \omega]$ tiene por renglones las variables de la gramática y como columnas cadenas de terminales de longitud k, almacena producciones que se determinan de la siguiente forma:

Sean $Y \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_r$ las reglas correspondientes a la variable Y.

Dado que los conjuntos $\textsc{First}_k(\alpha_i) \odot_k \textsc{Follow}_k(Y)$ son disjuntos entonces

- para cada $\omega \in \textsc{First}_k(\alpha_1) \odot_k \textsc{Follow}_k(Y) \cup \textsc{First}_k(\alpha_2) \odot_k \textsc{Follow}_k(Y) \cup \cdots \cup \textsc{First}_k(\alpha_r) \odot_k \textsc{Follow}_k(Y)$ se tiene que

$$M[Y, \omega] = \alpha_i \quad \text{si y sólo si} \quad \omega \in \textsc{First}_k(\alpha_i) \odot_k \textsc{Follow}_k(Y)$$

- en otro caso se deja la entrada de la tabla vacía o se incluye una rutina para manejar el error

Obs. La construcción de parsers **LL(k)** se puede restringir a gramáticas **SLL(k)**. La coincidencia de los símbolos leidos por adelantado y las columnas en la tabla de parsing hacen que la regla de producción sea única. Pero esto hace que la tabla pueda ser muy grande para $k > 1$, por lo que los parsers k-predictivos son evitados en la práctica y se usan más los de tipo **LL(1)** que son **SLL(1)**.

Referencias

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman.
Compilers, Principles, Techniques and Tools.
Pearson Education Inc., Second edition, 2007.

[2] M. L. Scott.
Programming Language Pragmatics.
Morgan-Kauffman Publishers, Third edition, 2009.

[3] Y. Su and S. Y. Yan.
Principles of Compilers, A New Approach to Compilers Including the Algebraic Method.
Springer-Verlag, Berlin Heidelberg, 2011.

[4] R. Wilhelm, H. Seidl, and S. Hack.
Compiler Design.
Springer-Verlag Berlin Heidelberg, 2013.

Ejemplo e imágenes tomadas de libro “Compiler Design”, capítulo 3.