Compiladores 24-2
Análisis Sintáctico

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

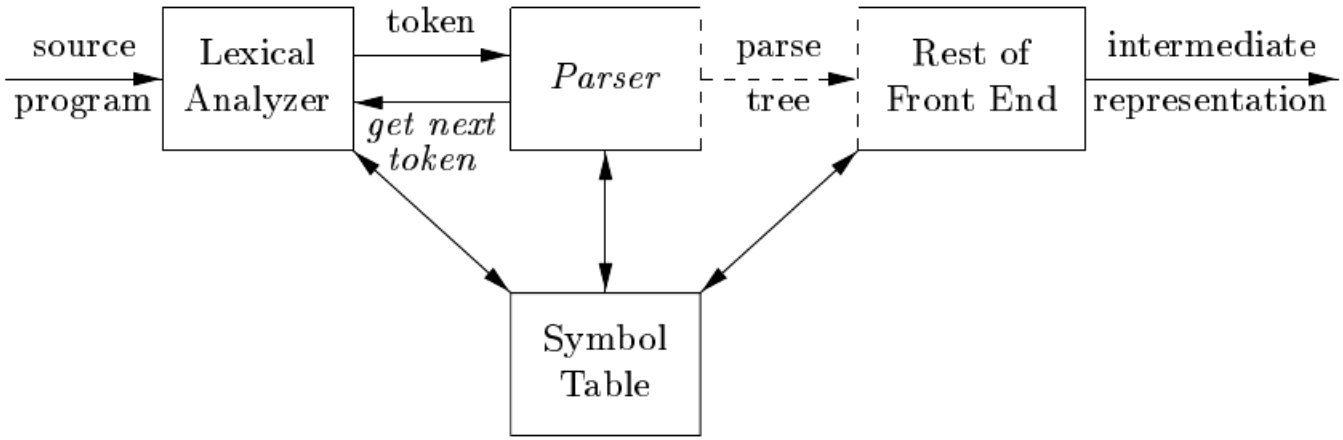
Facultad de Ciencias, UNAM

19 febrero 2024

Análisis sintáctico

panorama

- ✓ La segunda fase de un compilador, también llamado *parsing*, es considerada la parte más importante de la compilación.
- ✓ Esta fase utiliza el resultado del analizador léxico, es decir la secuencia de tokens obtenidos del archivo fuente (sólo el nombre de ellos), para crear una estructura concreta que represente la forma gramatical del programa.



Análisis sintáctico

panorama

- ✓ El objetivo de esta fase es reconocer palabras y verificar que la secuencia de tokens pueda ser generada/aceptada por la gramática del lenguaje.

Análisis sintáctico

panorama

- ✓ El objetivo de esta fase es reconocer palabras y verificar que la secuencia de tokens pueda ser generada/aceptada por la gramática del lenguaje.

el *parser* es un analizador gramatical

Análisis sintáctico

panorama

- ✓ El objetivo de esta fase es reconocer palabras y verificar que la secuencia de tokens pueda ser generada/aceptada por la gramática del lenguaje.

  el *parser* es un analizador gramatical

- ✓ Las gramáticas libres de contexto (no recursivas, sin transiciones épsilon y no ambiguas) definen lenguajes al establecer las reglas para estructurar palabras y así poder obtener una representación única del archivo fuente.

Análisis sintáctico

idea general

Un parser es un aceptador de una gramática libre de contexto que genera una derivación de una cadena de tokens aceptada.

Análisis sintáctico

idea general

Un parser es un aceptador de una gramática libre de contexto que genera una derivación de una cadena de tokens aceptada.

- Describir a un lenguaje de programación mediante una gramática libre de contexto no-recursiva, sin transiciones épsilon y no-ambigua (en la medida de lo posible).
  usar precedencia de operadores, transformaciones a formas normales

Análisis sintáctico

idea general

Un parser es un aceptador de una gramática libre de contexto que genera una derivación de una cadena de tokens aceptada.

- Describir a un lenguaje de programación mediante una gramática libre de contexto no-recursiva, sin transiciones épsilon y no-ambigua (en la medida de lo posible).
  usar precedencia de operadores, transformaciones a formas normales
- Análisis de la cadena/secuencia de tokens para reconocer al programa como una cadena válida del lenguaje de programación mediante una máquina abstracta correspondiente.
  autómatas de pila

Análisis sintáctico

idea general

Un parser es un aceptador de una gramática libre de contexto que genera una derivación de una cadena de tokens aceptada.

- Describir a un lenguaje de programación mediante una gramática libre de contexto no-recursiva, sin transiciones épsilon y no-ambigua (en la medida de lo posible).
  usar precedencia de operadores, transformaciones a formas normales
- Análisis de la cadena/secuencia de tokens para reconocer al programa como una cadena válida del lenguaje de programación mediante una máquina abstracta correspondiente.
  autómatas de pila

Observación. Un parse tree o un árbol de sintaxis concreta es único en una gramática no-ambigua aunque las derivaciones pueden ser diferentes (por la izquierda o por la derecha).

Análisis sintáctico

idea general

- La cadena de tokens se analiza de izquierda a derecha y revisando símbolos por adelantado para construir el árbol de sintaxis concreta.
- El análisis del programa se lleva a cabo seleccionando una regla de producción de la gramática que coincida con el símbolo de lectura en turno y haciendo coincidir la producción con los símbolos subsecuentes.

Análisis sintáctico

idea general

- La cadena de tokens se analiza de izquierda a derecha y revisando símbolos por adelantado para construir el árbol de sintaxis concreta.
- El análisis del programa se lleva a cabo seleccionando una regla de producción de la gramática que coincida con el símbolo de lectura en turno y haciendo coincidir la producción con los símbolos subsecuentes.

La representación de alto nivel de un programa, generada por el *parser*, es un árbol de sintaxis concreta o ***parse tree*** cuyos nodos internos son símbolos no-terminales de la gramática y las hojas son símbolos terminales.

Análisis sintáctico

idea general

- La cadena de tokens se analiza de izquierda a derecha y revisando símbolos por adelantado para construir el árbol de sintaxis concreta.
- El análisis del programa se lleva a cabo seleccionando una regla de producción de la gramática que coincida con el símbolo de lectura en turno y haciendo coincidir la producción con los símbolos subsecuentes.

La representación de alto nivel de un programa, generada por el *parser*, es un árbol de sintaxis concreta o ***parse tree*** cuyos nodos internos son símbolos no-terminales de la gramática y las hojas son símbolos terminales.

Si el programa no es reconocido como una secuencia correcta de palabras, según la gramática que define al lenguaje, entonces se deben reportar los errores sintácticos.

Análisis sintáctico

tipos de analizadores

El árbol de parsing muestra por completo y en concreto que una secuencia de tokens se puede derivar de la gramática.

¿Cómo construir el *parse tree* de forma eficiente?

Análisis sintáctico

tipos de analizadores

El árbol de parsing muestra por completo y en concreto que una secuencia de tokens se puede derivar de la gramática.

¿Cómo construir el *parse tree* de forma eficiente?

R: el *parser* es un analizador gramatical

Existen varios tipos de analizadores que ofrecen diferentes ventajas, principalmente se busca reconocer programas bien estructurados de forma eficiente.

Análisis sintáctico

tipos de analizadores

El árbol de parsing muestra por completo y en concreto que una secuencia de tokens se puede derivar de la gramática.

¿Cómo construir el *parse tree* de forma eficiente?

R: el *parser* es un analizador gramatical

Existen varios tipos de analizadores que ofrecen diferentes ventajas, principalmente se busca reconocer programas bien estructurados de forma eficiente.

Dada una gramática libre de contexto en alguna forma normal:

- Parser Universal: gramática en forma normal de Chomsky y usar el algoritmo CYK (Cocke–Younger–Kasami) (bottom-up y dinámico).

Análisis sintáctico

tipos de analizadores

El árbol de parsing muestra por completo y en concreto que una secuencia de tokens se puede derivar de la gramática.

¿Cómo construir el *parse tree* de forma eficiente?

R: el *parser* es un analizador gramatical

Existen varios tipos de analizadores que ofrecen diferentes ventajas, principalmente se busca reconocer programas bien estructurados de forma eficiente.

Dada una gramática libre de contexto en alguna forma normal:

- Parser Universal: gramática en forma normal de Chomsky y usar el algoritmo CYK (Cocke–Younger–Kasami) (bottom-up y dinámico).
- Parsing LL (Left-to-right & Left-most derivation, predictivo y top-down)

Análisis sintáctico

tipos de analizadores

El árbol de parsing muestra por completo y en concreto que una secuencia de tokens se puede derivar de la gramática.

¿Cómo construir el *parse tree* de forma eficiente?

R: el *parser* es un analizador gramatical

Existen varios tipos de analizadores que ofrecen diferentes ventajas, principalmente se busca reconocer programas bien estructurados de forma eficiente.

Dada una gramática libre de contexto en alguna forma normal:

- Parser Universal: gramática en forma normal de Chomsky y usar el algoritmo CYK (Cocke–Younger–Kasami) (bottom-up y dinámico).
- Parsing LL (Left-to-right & Left-most derivation, predictivo y top-down)
- Parsing LR (Left-to-right & Right-most derivation y bottom-up)

Análisis sintáctico

parsing

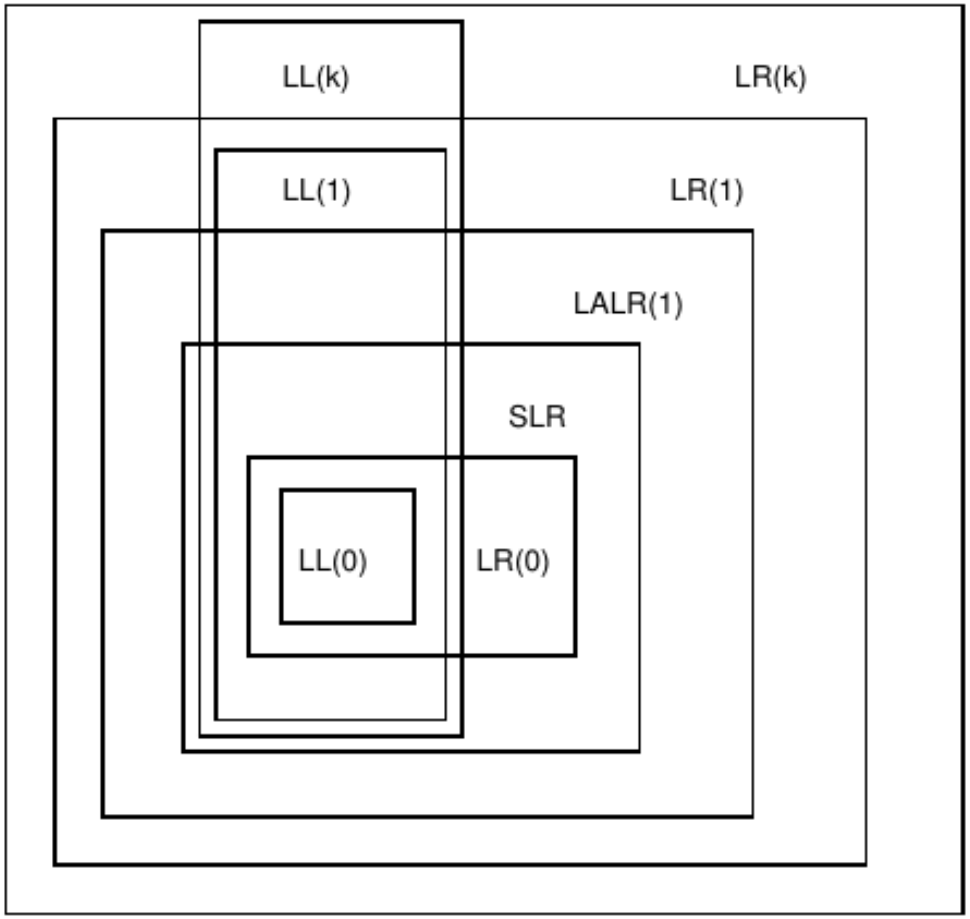
El tipo de una gramática y la forma en que se obtiene una derivación determinan los tipos de parsers

Análisis sintáctico

parsing

El tipo de una gramática y la forma en que se obtiene una derivación determinan los tipos de parsers

1. Top-down parsing
2. Bottom-up parsing

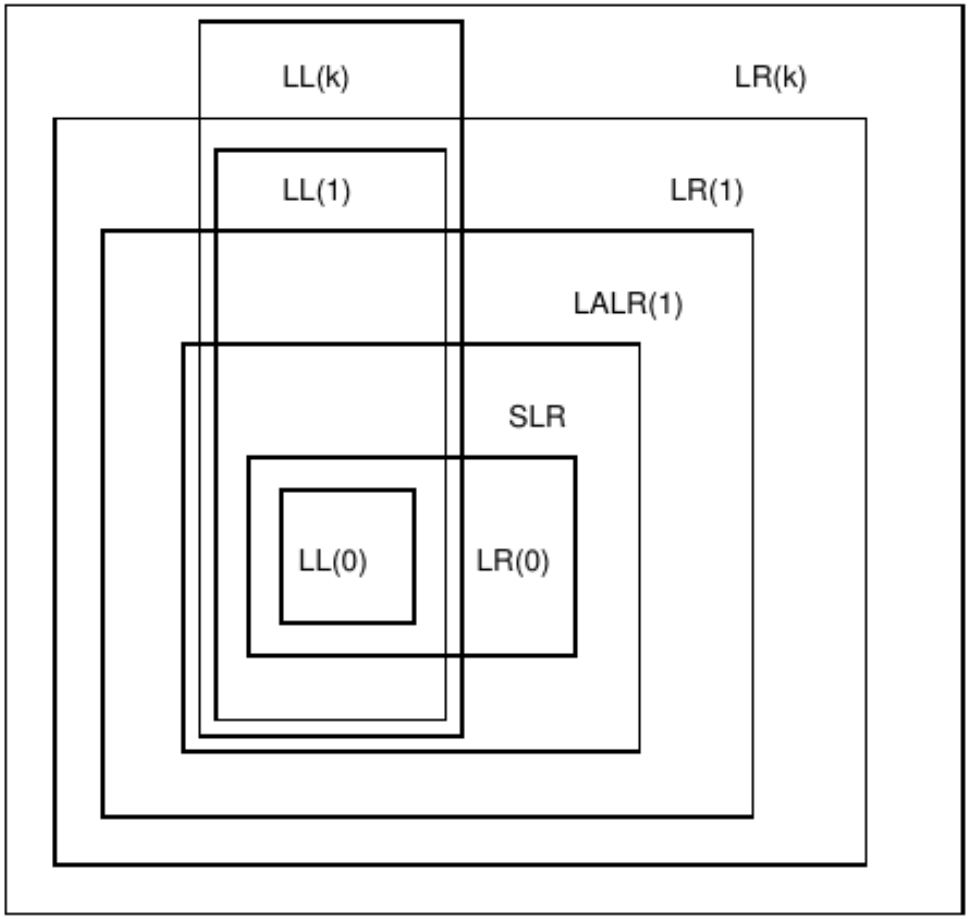


Análisis sintáctico

parsing

El tipo de una gramática y la forma en que se obtiene una derivación determinan los tipos de parsers

1. Top-down parsing
2. Bottom-up parsing



Anticipar la regla de producción adecuada para la derivación.

Análisis sintáctico

gramáticas

Una gramática tipo **LL** tiene la característica que para cada símbolo no terminal A, si la derivación

$$S \rightarrow^{\star} w_1 A \alpha \rightarrow w_1 v \alpha \rightarrow^{\star} w_1 w_2 w_3$$

coincide con otra derivación con el mismo prefijo hasta la reescritura de A, entonces de las producciones de A sólo existe una regla que lo permite.

Análisis sintáctico

gramáticas

Una gramática tipo **LL** tiene la característica que para cada símbolo no terminal A, si la derivación

$$S \rightarrow^{\star} w_1 A \alpha \rightarrow w_1 v \alpha \rightarrow^{\star} w_1 w_2 w_3$$

coincide con otra derivación con el mismo prefijo hasta la reescritura de A, entonces de las producciones de A sólo existe una regla que lo permite. Una gramática **LL** no tiene recursión por la izquierda (Forma Normal de Greibach).

Análisis sintáctico

gramáticas

Una gramática tipo **LL** tiene la característica que para cada símbolo no terminal A, si la derivación

$$S \rightarrow^{\star} w_1 A \alpha \rightarrow w_1 v \alpha \rightarrow^{\star} w_1 w_2 w_3$$

coincide con otra derivación con el mismo prefijo hasta la reescritura de A, entonces de las producciones de A sólo existe una regla que lo permite. Una gramática **LL** no tiene recursión por la izquierda (Forma Normal de Greibach).

- Un parser **LL(k)** es uno para una gramática de este estilo y donde se revisan k tokens por adelantado. Se creará un parse tree top-down y en preorden con los tokens adelantados para crear un prefijo útil.
- Una gramática **LL** sin transiciones épsilon es una gramática **SLR**.

Análisis sintáctico

gramáticas

Una gramática tipo **LL** tiene la característica que para cada símbolo no terminal A, si la derivación

$$S \rightarrow^{\star} w_1 A \alpha \rightarrow w_1 v \alpha \rightarrow^{\star} w_1 w_2 w_3$$

coincide con otra derivación con el mismo prefijo hasta la reescritura de A, entonces de las producciones de A sólo existe una regla que lo permite. Una gramática **LL** no tiene recursión por la izquierda (Forma Normal de Greibach).

- Un parser **LL(k)** es uno para una gramática de este estilo y donde se revisan k tokens por adelantado. Se creará un parse tree top-down y en preorden con los tokens adelantados para crear un prefijo útil.
- Una gramática **LL** sin transiciones épsilon es una gramática **SLR**.

Las gramáticas **LR** son también lineales y son más expresivas, permiten recursión por la izquierda.

Análisis sintáctico

ejemplo LL vs LR

Gramática que describe una secuencia de identificadores

$$
\begin{array}{rcl}
\mathit{id\_list} & \rightarrow & \texttt{id}\ \ \mathit{id\_list\_tail} \\
\mathit{id\_list\_tail} & \rightarrow & \texttt{,}\ \ \texttt{id}\ \ \mathit{id\_list\_tail} \\
\mathit{id\_list\_tail} & \rightarrow & \texttt{;}
\end{array}
$$

Análisis sintáctico

ejemplo LL vs LR

Gramática que describe una secuencia de identificadores

$$
\begin{array}{rcl}
id\_list & \rightarrow & \texttt{id}\ \ id\_list\_tail \\
id\_list\_tail & \rightarrow & \texttt{,}\ \texttt{id}\ \ id\_list\_tail \\
id\_list\_tail & \rightarrow & \texttt{;}
\end{array}
$$

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] J.-C. Filliâtre.
Curso Compilation (inf564) école Polytechnique, Palaiseau, Francia.
http://www.enseignement.polytechnique.fr/informatique/INF564/, 2018.
Material en francés.

[3] A. Milani Fard, A. Deldari, and H. Deldari.
Quick grammar type recognition: Concepts and techniques.
https://www.researchgate.net/publication/228609916_Quick_Grammar_Type_Recognition_Concepts_and_Techniques, 2007.

[4] F. Pfenning.
Notas del curso (15-411) Compiler Design.
https://www.cs.cmu.edu/~fp/courses/15411-f14/, 2014.

[5] F. Pottier.
Presentaciones del curso Compilation (inf564) École Polytechnique, Palaiseau, Francia.
http://gallium.inria.fr/~fpottier/X/INF564/, 2016.
Material en francés.

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

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

[8] S. Zdancewic.
Notas del curso (CIS 341) - Compilers, Universidad de Pennsylvania, Estados Unidos.
https://www.cis.upenn.edu/~cis341/current/, 2018.