Compiladores 24-2
Análisis Sintáctico: parser bottom-up

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

Facultad de Ciencias, UNAM

11 marzo 2024

Análisis sintáctico

Parser bottom-up

- Analizadores que construyen un árbol de sintaxis concreta (parse tree) desde las hojas y hacia la raíz.

Análisis sintáctico

Parser bottom-up

- Analizadores que construyen un árbol de sintaxis concreta (parse tree) desde las hojas y hacia la raíz.
- La técnica para generar los árboles se llama *shift-reduce*, que son acciones para obtener una derivación por la derecha y paso a paso mediante un proceso inverso.

Análisis sintáctico

Parser bottom-up

- Analizadores que construyen un árbol de sintaxis concreta (parse tree) desde las hojas y hacia la raíz.
- La técnica para generar los árboles se llama *shift-reduce*, que son acciones para obtener una derivación por la derecha y paso a paso mediante un proceso inverso.
- La clase de gramáticas de estos analizadores son las **LR** (lectura left-to-right del input y derivaciones más a la derecha).

Análisis sintáctico

Parser bottom-up

- Analizadores que construyen un árbol de sintaxis concreta (parse tree) desde las hojas y hacia la raíz.
- La técnica para generar los árboles se llama *shift-reduce*, que son acciones para obtener una derivación por la derecha y paso a paso mediante un proceso inverso.
- La clase de gramáticas de estos analizadores son las **LR** (lectura left-to-right del input y derivaciones más a la derecha).
- Ventajas del análisis **LR**:
  - gramáticas **LR** son más expresivas que las **LL** ya que permiten recursión izquierda en las producciones;

Análisis sintáctico

Parser bottom-up

- Analizadores que construyen un árbol de sintaxis concreta (parse tree) desde las hojas y hacia la raíz.
- La técnica para generar los árboles se llama *shift-reduce*, que son acciones para obtener una derivación por la derecha y paso a paso mediante un proceso inverso.
- La clase de gramáticas de estos analizadores son las **LR** (lectura left-to-right del input y derivaciones más a la derecha).
- Ventajas del análisis **LR**:
  - gramáticas **LR** son más expresivas que las **LL** ya que permiten recursión izquierda en las producciones;
  - método más usual para reconocer gramáticas libres de contexto de lenguajes de programación;

Análisis sintáctico

Parser bottom-up

- Analizadores que construyen un árbol de sintaxis concreta (parse tree) desde las hojas y hacia la raíz.
- La técnica para generar los árboles se llama *shift-reduce*, que son acciones para obtener una derivación por la derecha y paso a paso mediante un proceso inverso.
- La clase de gramáticas de estos analizadores son las **LR** (lectura left-to-right del input y derivaciones más a la derecha).
- Ventajas del análisis **LR**:
  - gramáticas **LR** son más expresivas que las **LL** ya que permiten recursión izquierda en las producciones;
  - método más usual para reconocer gramáticas libres de contexto de lenguajes de programación;
  - método más eficiente sin backtracking que utiliza shift-reduce;

Análisis sintáctico

Parser bottom-up

- Analizadores que construyen un árbol de sintaxis concreta (parse tree) desde las hojas y hacia la raíz.
- La técnica para generar los árboles se llama *shift-reduce*, que son acciones para obtener una derivación por la derecha y paso a paso mediante un proceso inverso.
- La clase de gramáticas de estos analizadores son las **LR** (lectura left-to-right del input y derivaciones más a la derecha).
- Ventajas del análisis **LR**:
  - gramáticas **LR** son más expresivas que las **LL** ya que permiten recursión izquierda en las producciones;
  - método más usual para reconocer gramáticas libres de contexto de lenguajes de programación;
  - método más eficiente sin backtracking que utiliza shift-reduce;
  - detecta errores más rápido, dado que se construye desde abajo.

Parser bottom-up

Preliminares

- ✓ Las gramáticas que consideramos no son ambiguas, para asegurar que existe una única derivación, en este caso reescribiendo el símbolo no-terminal más a la derecha en el proceso.

- ✓ Las gramáticas que consideramos no son ambiguas, para asegurar que existe una única derivación, en este caso reescribiendo el símbolo no-terminal más a la derecha en el proceso.

El análisis **LR** funciona en general de la siguiente forma:

- la gramática se usa como un autómata no-determinista donde un estado indica una producción parcialmente reconocida (usando una marca o bandera •) y la pila contiene a los estados precedentes;
- transformar este autómata, de ser posible, en uno determinista

Parser bottom-up

Preliminares

✓ Las gramáticas que consideramos no son ambiguas, para asegurar que existe una única derivación, en este caso reescribiendo el símbolo no-terminal más a la derecha en el proceso.

El análisis **LR** funciona en general de la siguiente forma:

- la gramática se usa como un autómata no-determinista donde un estado indica una producción parcialmente reconocida (usando una marca o bandera •) y la pila contiene a los estados precedentes;
- transformar este autómata, de ser posible, en uno determinista

★ La pila de estados representa la historia de visita de éstos, el tope de la pila es justo el estado actual y la entrada es una secuencia de lexemas.

Parser bottom-up

Preliminares

Se pueden realizar dos acciones desde un estado q:
un desplazamiento (*shift*) o una reducción(*reduce*).

Se pueden realizar dos acciones desde un estado q:
un desplazamiento (*shift*) o una reducción(*reduce*).

- desplazamiento (*shift*)
  si el lexema inicial es a, eliminar a de la cadena entrada y guardar en la pila
  el nuevo estado q' obtenido por la función de transición;
  es decir que si se lee un símbolo terminal se guarda en la pila.

Se pueden realizar dos acciones desde un estado q:
un desplazamiento (*shift*) o una reducción(*reduce*).

- desplazamiento (*shift*)
  si el lexema inicial es a, eliminar a de la cadena entrada y guardar en la pila el nuevo estado q' obtenido por la función de transición;
  es decir que si se lee un símbolo terminal se guarda en la pila.
- reducción (*reduce*)
  si el estado q está etiquetado por $A \to \beta\bullet$, sacar de la pila el mismo número de símbolos que la longitud de β para regresar el autómata a un estado anterior p y después guardar el nuevo estado p' que se obtenga desde p al leer A; es decir que se reconoce el tope de la pila con la parte derecha de alguna producción y se reemplaza.

Parser bottom-up

Preliminares

Dada una cadena $\alpha\beta z$ (con z cadena de símbolos terminales), el **handle** o mango de ella es :

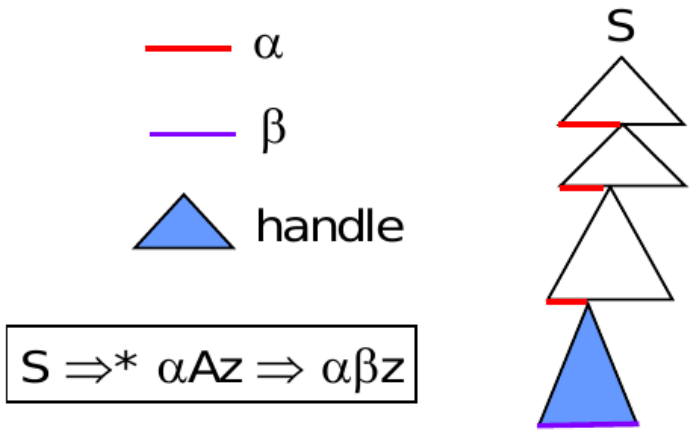
> una subcadena que tiene el mismo patrón que la parte derecha de una producción $A \rightarrow \beta$ y cuya reducción es el símbolo no-terminal en la izquierda de la producción:

Parser bottom-up

Preliminares

Dada una cadena $\alpha\beta z$ (con z cadena de símbolos terminales), el **handle** o mango de ella es :

> una subcadena que tiene el mismo patrón que la parte derecha de una producción $A \to \beta$ y cuya reducción es el símbolo no-terminal en la izquierda de la producción:
>
> $\alpha \in (\Sigma \cup \Gamma)^\star$
>
> $A \to \beta\bullet$ (β ha sido reconocida)



Parser bottom-up

Preliminares

- Los algoritmos que estudiaremos utilizan las operaciones shift-reduce para encontrar handles y construir el árbol.

Parser bottom-up

Preliminares

- Los algoritmos que estudiaremos utilizan las operaciones shift-reduce para encontrar handles y construir el árbol.
- El handle representa un paso en el proceso reverso de una derivación más a la derecha de una cadena y serán justo los símbolos en el tope de la pila para ser reducidos.

Parser bottom-up

Preliminares

- Los algoritmos que estudiaremos utilizan las operaciones shift-reduce para encontrar handles y construir el árbol.
- El handle representa un paso en el proceso reverso de una derivación más a la derecha de una cadena y serán justo los símbolos en el tope de la pila para ser reducidos.
- Extensiones de los lenguajes:
  incluir el fin de cadena (#) en la cadena de entrada
  incluir un nuevo símbolo inicial (S) en la gramática

Parser bottom-up

Preliminares

- Los algoritmos que estudiaremos utilizan las operaciones shift-reduce para encontrar handles y construir el árbol.
- El handle representa un paso en el proceso reverso de una derivación más a la derecha de una cadena y serán justo los símbolos en el tope de la pila para ser reducidos.
- Extensiones de los lenguajes:
  incluir el fin de cadena (#) en la cadena de entrada
  incluir un nuevo símbolo inicial (S) en la gramática
- Las configuraciones del autómata cambian respecto a cada método pero la configuración inicial debe tener por un lado el símbolo de fondo de la pila y por otro la cadena de entrada seguida de #.

Parser bottom-up

Preliminares

- Los algoritmos que estudiaremos utilizan las operaciones shift-reduce para encontrar handles y construir el árbol.
- El handle representa un paso en el proceso reverso de una derivación más a la derecha de una cadena y serán justo los símbolos en el tope de la pila para ser reducidos.
- Extensiones de los lenguajes:
  incluir el fin de cadena ($\#$) en la cadena de entrada
  incluir un nuevo símbolo inicial (S) en la gramática
- Las configuraciones del autómata cambian respecto a cada método pero la configuración inicial debe tener por un lado el símbolo de fondo de la pila y por otro la cadena de entrada seguida de $\#$.
- La configuración final debe de tener por un lado el símbolo inicial de la gramática extendida S y por otro lado se consume la cadena de entrada y sólo resta el símbolo de fin de cadena.

Parser bottom-up

intuición (shift-reduce)

- Iniciar con las hojas del árbol (tokens) y terminar con la raíz (símbolo inicial).
- Se busca reconocer partes derechas de las producciones para sustituirlas por símbolos no-terminales hasta obtener el símbolo inicial de la gramática.

Parser bottom-up

intuición (shift-reduce)

- Iniciar con las hojas del árbol (tokens) y terminar con la raíz (símbolo inicial).
- Se busca reconocer partes derechas de las producciones para sustituirlas por símbolos no-terminales hasta obtener el símbolo inicial de la gramática.
- El análisis es una secuencia de desplazamientos y reducciones:

  shift mover el look-ahead al tope de la pila

stack	*entrada*	*accion*
$($	$1+2+(3+4))+5$	shift 1
$(1$	$+2+(3+4))+5$	

  reduce reemplazar los símbolos en β del tope de la pila por la variable correspondiente ($\texttt{pop}\ \beta$, $\texttt{push}$A)

stack	*entrada*	*accion*
$(S+E$	$+(3+4))+5$	reduce $S \to S+E$
$(S$	$+(3+4))+5$	

Parser bottom-up

intuición

- En cada paso del análisis decidir si se debe hacer un shift o un reduce

Parser bottom-up

intuición

- En cada paso del análisis decidir si se debe hacer un shift o un reduce
  ¿Cómo decidir esto?

Parser bottom-up

intuición

- En cada paso del análisis decidir si se debe hacer un shift o un reduce
  ¿Cómo decidir esto? A veces se puede reducir pero no es la mejor acción y otras veces se podrá reducir de diferentes formas.

Parser bottom-up

intuición

- En cada paso del análisis decidir si se debe hacer un shift o un reduce
  ¿Cómo decidir esto? A veces se puede reducir pero no es la mejor acción y otras veces se podrá reducir de diferentes formas.
- Respuesta: usar una tabla de acciones que determine lo que debe hacer el parser.

Parser bottom-up

intuición

- En cada paso del análisis decidir si se debe hacer un shift o un reduce
  ¿Cómo decidir esto? A veces se puede reducir pero no es la mejor acción y otras veces se podrá reducir de diferentes formas.
- Respuesta: usar una tabla de acciones que determine lo que debe hacer el parser.

Algoritmo:
buscar en la tabla el estado actual del parser y el símbolo del input:

Si $M[q, c] = \mathit{shift}(q')$ entonces shift & push(q')

Si $M[q, c] = A \to \alpha$ entonces reduce & pop$_{|\alpha|}$; q = top; push($M[q', A]$)

Análisis sintáctico

Parser bottom-up

El algoritmo se sirve de la cadena de entrada y una tabla de parsing que contiene las acciones (shift, reduce o accept) y las transiciones entre estados.

Algoritmo *grosso modo*:
buscar en la tabla el estado actual del parser y el símbolo del input:

Si $M[q, c] = \mathit{shift}(q')$ entonces shift & push(q')

Si $M[q, c] = A \to \alpha$ entonces reduce & $\text{pop}_{|\alpha|}$; q = top; push($M[q', A]$)

Análisis sintáctico

Parser bottom-up

El algoritmo se sirve de la cadena de entrada y una tabla de parsing que contiene las acciones (shift, reduce o accept) y las transiciones entre estados.

Algoritmo *grosso modo*:
buscar en la tabla el estado actual del parser y el símbolo del input:

Si $M[q, c] = \mathit{shift}(q')$ entonces shift & push(q')

Si $M[q, c] = A \to \alpha$ entonces reduce & $\text{pop}_{|\alpha|}$; q = top; push($M[q', A]$)

El comportamiento del algoritmo LR es el mismo, lo que estudiaremos son variantes en la construcción de la tabla y del autómata que guarda el avance en el análisis de la cadena de entrada.

Análisis sintáctico

Parser bottom-up

Input: Cadena de entrada *w* y la tabla LR con las funciones ACTION y GOTO.

Output: Si la cadena *w* está en el lenguaje de la gramática, entonces se devuelven las reducciones del parsing bottom-up; sino se devuelve un error.

El parser comienza con el estado inicial s_0 en la pila y $w\#$ como la cadena de entrada

```
let a be the first symbol of w#;
while(1)  /* repeat forever */
{    let s be the state on top of the stack;
     if ( ACTION[s; a] = shift t )
     {   push t onto the stack;
         let a be the next input symbol;
     } else if ( ACTION[s; a] = reduce A->v )
            {
              pop |v| symbols off the stack;
              let state t now be on top of the stack;
              push GOTO[t; A] onto the stack;
              output the production A->v;
            } else if ( ACTION[s; a] = accept ) break;
                   /* parsing is done */
              else call error-recovery routine;
}
```

Referencias

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.
Compilers, Principles, Techniques and Tools.
Pearson Education Inc., Second edition, 2007.

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

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

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

[5] Michael Lee Scott.
Programming Language Pragmatics.
Morgan-Kauffman Publishers, Third edition, 2009.

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

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