Análisis Sintáctico Descendente

CI4721 – Lenguajes de Programación II

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

UNIVERSIDAD SIMÓN BOLÍVAR

Reconocedor descendente determinístico

Sea $G = (N, \Sigma, P, S)$ una CFG cualquiera. Entonces el PDA extendido

$$LL_k(G) = (\{q_0, q_1\}, \Sigma, N \cup \Sigma, \delta, q_0, \{q_1\})$$

con δ definida según

$$\delta(q_0, \lambda, \lambda, \lambda) = \{(q_1, S)\}$$
$$\delta(q_1, \mathbf{a}, \mathbf{a}, \mathbf{a}) = \{(q_1, \lambda)\}, \forall \mathbf{a} \in \Sigma$$
$$\delta(q_1, \lambda, \mathbf{u}, A) = \{A \to \alpha \in P \wedge u \in LA_k(A \to \alpha) : (q_1, \alpha)\}, \forall A \in N$$

es un reconocedor descendente y determinístico para G.

- $\mathbf{u} \in \Sigma^+$ corresponde al *lookahead*.
- Si G es $LL(k)$ la conjunción asegura que habrá exactamente una transición por cada combinación de no-terminal y *lookahead*.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo convertirlo en un programa?

Convirtiendo transiciones en datos

- En nuestro PDA determinístico con *lookahead* . . .
  - El estado inicial solamente se usa para empilar S.
  - El estado final solamente tiene transiciones a sí mismo.
  - En cada transición solamente varían el *lookahead* y el símbolo a remover del tope de la pila.
  - Las expansiones están implícitas en las transiciones.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo convertirlo en un programa?

Convirtiendo transiciones en datos

- En nuestro PDA determinístico con *lookahead* . . .
  - El estado inicial solamente se usa para empilar S.
  - El estado final solamente tiene transiciones a sí mismo.
  - En cada transición solamente varían el *lookahead* y el símbolo a remover del tope de la pila.
  - Las expansiones están implícitas en las transiciones.
- Podemos diseñar un algoritmo muy compacto alrededor de
  - Una entrada con su marcador final $\#^k$.
  - Un *buffer* de tamaño k para el *lookahead*.
  - Una pila que comienza conteniendo S sobre un centinela $\#$.
  - Una tabla cuyas entradas indiquen la producción a expandir para cada combinación de *lookahead* y tope de pila.

El algoritmo es independiente de la gramática
solamente necesita la tabla.

UNIVERSIDAD SIMÓN BOLÍVAR

Reconocedor $LL(k)$ por Tabla

Inicialización

input: $w \in \Sigma$ con $\#^k$ marcadores y la Tabla M asociada a la gramática

{Empilar el centinela y el símbolo inicial}
push(#)
push(S)
{Preparar el $k-lookahead$}
$a \leftarrow$ los primeros k símbolos de w
{Determinar el tope de la pila sin consumirlo}
$X \leftarrow$ **top**()

UNIVERSIDAD SIMÓN BOLÍVAR

Reconocedor $LL(k)$ por Tabla

Procesamiento

while $X \neq \#$ **do**
  if $X = trunc_1(a)$ **then**
    pop() y consumir siguiente de w al final de a
  else if $X \in \Sigma$ **then**
    error por terminal inesperado en la entrada
  else if $M[X, a]$ vacío **then**
    error por no poder expandir el no terminal
  else if $M[X, a] = X \to Y_1 Y_2 \ldots Y_n$ **then**
    print $X \to Y_1 Y_2 \ldots Y_n$
    pop()
    push$(Y_n Y_{n-1} \ldots Y_2 Y_1)$
  end if
  $X \leftarrow$ **top**()
end while

output: Si $w \in L(G)$, la derivación más izquierda, sino error

UNIVERSIDAD SIMÓN BOLÍVAR

El secreto está en la tabla

Características

- Una fila por cada $A \in N$.
- Una columna por cada *lookahead* posible.

UNIVERSIDAD SIMÓN BOLÍVAR

El secreto está en la tabla

Características

- Una fila por cada $A \in N$.
- Una columna por cada *lookahead* posible.

$$\sum_{i=0}^{k} |\Sigma|^i = \frac{1 - |\Sigma|^{k+1}}{1 - |\Sigma|}$$

(no se les olvide la columna para $\#^k$)

UNIVERSIDAD SIMÓN BOLÍVAR

El secreto está en la tabla

Características

- Una fila por cada $A \in N$.
- Una columna por cada *lookahead* posible.

$$\sum_{i=0}^{k} |\Sigma|^i = \frac{1 - |\Sigma|^{k+1}}{1 - |\Sigma|}$$

(no se les olvide la columna para $\#^k$)

- En cada posición se almacena
  - La producción a expandir – para ahorrar espacio, típicamente es un identificador numérico o apuntador a la representación de la regla.
  - Un indicador de error sintáctico.

Los requerimientos de espacio son prohibitivos para $k > 1$

UNIVERSIDAD SIMÓN BOLÍVAR

El secreto está en la tabla

Construcción

input: $G = (N, \Sigma, P, S)$ y $\forall A \in N, FIRST_k(A)$ y $FOLLOW_k(A)$

for all $A \to \alpha \in P$ **do**
  for all $a \in FIRST_k(\alpha)$ **do**
    $M[A, a] \leftarrow A \to \alpha$
  end for
  if $\lambda \in FIRST_k(\alpha)$ **then**
    for all $b \in FOLLOW_k(A)$ **do**
      $M[A, b] \leftarrow A \to \alpha$
    end for
  end if
end for

Las posiciones que queden vacías corresponden a errores de sintaxis.

Reconocer (mini) JSON

La gramática

Gramática inicial para una versión reducida de JSON

$$
\begin{aligned}
J &\to \{L\} \\
L &\to \mathbf{s} : V \\
L &\to L, \mathbf{s} : V \\
A &\to [X] \\
X &\to V \\
X &\to V, L \\
V &\to \mathbf{s} \\
V &\to \mathbf{n} \\
V &\to J \\
V &\to A
\end{aligned}
$$

La gramática no es LL:

- Símbolo inicial J recursivo indirectamente a través de L por más de una vía.
- Recursión izquierda en L.
- Prefijos comunes en X.

UNIVERSIDAD SIMÓN BOLÍVAR

Reconocer (mini) JSON

La gramática transformada – 30 % más reglas

$S \to J\#$

$J \to \{L\}$

$L \to \mathbf{s} : VR$

$R \to ,L$

$R \to \lambda$

$A \to [X]$

$X \to VY$

$Y \to ,X$

$Y \to \lambda$

$V \to \mathbf{s}$

$V \to \mathbf{n}$

$V \to J$

$V \to A$

	FIRST	FOLLOW
S	{	#
J	{	# , }]
L	s	}
R	λ ,	}
A	[	, }]
X	s n { [	]
Y	λ ,	]
V	s n { [	, }]

Verifiquen que la gramática es fuertemente $LL(1)$

UNIVERSIDAD SIMÓN BOLÍVAR

Reconocer (mini) JSON

La tabla

	`s`	`n`	`,`	`:`	`[`	`]`	`{`	`}`	`#`
S							$S \to J\#$		
J							$J \to \{L\}$		
L	$L \to \mathbf{s}:VR$								
R			$R \to ,L$					$R \to \lambda$	
A					$A \to [X]$				
X	$X \to VY$	$X \to VY$			$X \to VY$		$X \to VY$		
Y			$Y \to ,X$			$Y \to \lambda$			
V	$V \to \mathbf{s}$	$V \to \mathbf{n}$			$V \to J$		$V \to A$		

- 72 celdas – 77 % *vacío*
- ¿Por qué **:** es irrelevante?
- ¿Por qué # es irrelevante?

UNIVERSIDAD SIMÓN BOLÍVAR

Recuperación de Errores

. . . porque el mundo no es perfecto

- El reconocedor predictivo es capaz de detectar dos errores:
  - Terminal al tope de la pila diferente del siguiente terminal de entrada.
  - La combinación de no terminal al tope de la pila y el *lookahead* no tiene regla de expansión asociada.
- Abortar el reconocimiento es inaceptable
  - El programa "está mal" – el proceso de síntesis no ocurrirá, pero el análisis debería continuar tanto como se pueda.
  - Cada error debe ser amplio y detallado
    - Ubicación – línea y columna, contexto de ser posible.
    - Condición – "esperaba X pero recibí Y"
  - Encontrar otros defectos ayudará al programador.
- Existen dos técnicas de recuperación aplicables

UNIVERSIDAD SIMÓN BOLÍVAR

Recuperación de Errores

. . . porque el mundo no es perfecto

- El reconocedor predictivo es capaz de detectar dos errores:
  - Terminal al tope de la pila diferente del siguiente terminal de entrada.
  - La combinación de no terminal al tope de la pila y el *lookahead* no tiene regla de expansión asociada.
- Abortar el reconocimiento es inaceptable
  - El programa "está mal" – el proceso de síntesis no ocurrirá, pero el análisis debería continuar tanto como se pueda.
  - Cada error debe ser amplio y detallado
    - Ubicación – línea y columna, contexto de ser posible.
    - Condición – "esperaba X pero recibí Y"
  - Encontrar otros defectos ayudará al programador.
- Existen dos técnicas de recuperación aplicables
  - Técnica del Pánico (*Panic Mode*).
  - Técnica del Engaño (*Phrase Level Recovery*).

UNIVERSIDAD SIMÓN BOLÍVAR

Técnica del Pánico

Panic Mode, a.k.a. Discard all the tokens!

- Descartar *tokens* hasta encontrar alguno que permita **resincronizar**.
- **Conjuntos de sincronización** construidos con heurísticas:
  - Resincronizar A usando $FOLLOW(A)$ – "se terminó el A incompleto".
  - $FIRST$ de construcciones de "alto nivel" para resincronizar construcciones de "bajo nivel" – depende de la gramática.
  - En algunos casos, agregar $FIRST(A)$ para resincronizar A hace posible continuar reconociendo cuando aparezca – depende de la gramática.
  - Ante un error expandiendo A, si existe $A \rightarrow \lambda$ entonces usarlo – esto difiere el error para más adelante.
  - Si el terminal al tope de la pila no coincide con el de la entrada, eliminarlo como si lo hubieras visto – "faltaba un ; y lo agregué".
- La posición de la tabla apunta al conjunto de resincronización.

UNIVERSIDAD SIMÓN BOLÍVAR

Técnica del Engaño

(Phrase Level Recovery, a.k.a. Let me type for you)

- Aplicada para recuperarse de los errores de expansión.
- La posición de la tabla apunta a una *subrutina* de manejo del error.
- Cada rutina es específica para la recuperación particular – altamente dependiente del lenguaje.
- Las rutinas "completan" lo que falta para continuar
  - Agregan, quitan o cambian símbolos en la entrada.
  - Sacan cosas de la pila – Muy peligroso.
  - Anuncian lo que hicieron para "corregir" el problema.
- Solamente aplicable a lenguajes (o sub-lenguajes)
  - Se conocen los errores más frecuentes.
  - Es fácil alcanzar formas sentenciales válidas con edición mínima.

UNIVERSIDAD SIMÓN BOLÍVAR

Técnica del Engaño

(Phrase Level Recovery, a.k.a. Let me type for you)

- Aplicada para recuperarse de los errores de expansión.
- La posición de la tabla apunta a una *subrutina* de manejo del error.
- Cada rutina es específica para la recuperación particular – altamente dependiente del lenguaje.
- Las rutinas "completan" lo que falta para continuar
  - Agregan, quitan o cambian símbolos en la entrada.
  - Sacan cosas de la pila – Muy peligroso.
  - Anuncian lo que hicieron para "corregir" el problema.
- Solamente aplicable a lenguajes (o sub-lenguajes)
  - Se conocen los errores más frecuentes.
  - Es fácil alcanzar formas sentenciales válidas con edición mínima.

No la llame *ad hoc*, llámela *ad hack*.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo convertirlo en un programa?

Convirtiendo transiciones en llamadas a funciones

- En nuestro PDA determinístico con *lookahead* . . .
  - La pila se usa para almacenar formas sentenciales.
  - Tan pronto se expande una forma sentencial, se intenta consumir tantos terminales como se pueda antes de volver a expandir.
  - Las expansiones están implícitas en las transiciones.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo convertirlo en un programa?

Convirtiendo transiciones en llamadas a funciones

- En nuestro PDA determinístico con *lookahead* . . .
  - La pila se usa para almacenar formas sentenciales.
  - Tan pronto se expande una forma sentencial, se intenta consumir tantos terminales como se pueda antes de volver a expandir.
  - Las expansiones están implícitas en las transiciones.
- Podemos diseñar un algoritmo muy compacto alrededor de
  - Una subrutina por cada no-terminal – la ejecución comienza por `S()`.
  - Cada procedimiento modela la secuencia de expansiones y consumo de entrada para las reglas $A \to \alpha$.

El algoritmo es dependiente de la gramática
simulando la pila del PDA con la pila de ejecución.

UNIVERSIDAD SIMÓN BOLÍVAR

Un procedimiento típico

```
void A () {
```

Escoger alguna $A \to X_1 X_2 \ldots X_n$
for $i = 1$ **to** n **do**
  if X_i es un no-terminal **then**
    `call` X_i `()`
  else if X_i = el símbolo actual de entrada **then**
    avanzar al siguiente símbolo de entrada
  else
    reportar error de sintaxis
  end if
end for

```
}
```

- Si la selección no es determinística, requiere *backtracking*.
- Para incluir *backtracking* el ciclo debe cambiarse por un avance selectivo sin descartar la entrada hasta estar seguros.

UNIVERSIDAD SIMÓN BOLÍVAR

Reconocer (mini) JSON

Uno de los procedimientos triviales

Reconocer `L` es trivial

- Una sola regla – $L \rightarrow \mathbf{s} : VR$
- *Lookahead* según $FIRST_1(L \rightarrow \mathbf{s} : VR)$

```
void L()
{
  try_rule(2);
  match('L','s');
  match('L',':');
  V();
  R();
}
```

- `try_rule` permite recordar los pasos de derivación.
- `match` verifica el *lookahead* esperado o aborta con error.

UNIVERSIDAD SIMÓN BOLÍVAR

Reconocer (mini) JSON

Uno de los procedimientos interesantes

Reconocer R es sólo un poco más complicado

- Dos reglas $R \rightarrow ,L$ y $R \rightarrow \lambda$
- *Lookaheads* $FIRST_1(R \rightarrow ,L)$ y $FOLLOW_1(R)$, respectivamente.

```
void R()
{
  switch (lookahead) {
    case ',': try_rule(3); match('R',','); L();
    case '}': try_rule(4); return;
    default:  error('R',lookahead);
  }
}
```

- Necesario verificar el *lookahead* antes de decidir cuál regla usar.
- Alternativa de error es explícita en este caso.

UNIVERSIDAD SIMÓN BOLÍVAR

Consideraciones

- El reconocedor por tabla solamente tiene sentido práctico para gramáticas $LL(1)$ – para el resto es muy costoso.
- Las gramáticas generales transformadas a $LL(k)$ suelen ser más complejas tanto en número como en forma de las reglas.
- ¿Cómo optimizaría una tabla $LL(k)$ para reducir la ocupación de espacio al mínimo posible?
- Aplique la técnica de recuperación de errores Modo de Pánico a la tabla para (mini) JSON. ¿Son útiles todas las heurísticas?
- Si el lenguaje es suficientemente simple y $LL(1)$, un recursivo descendiente es más barato – incluso si una de las subgramáticas es $LL(k) > 1$, ¿por qué?
- ¿Cómo extender el reconocedor para construir el árbol de derivación?

UNIVERSIDAD SIMÓN BOLÍVAR

En la práctica

- Muchos lenguajes de programación proveen librerías que asisten en la construcción de Reconocedores Recursivos Descendentes con o sin *backtracking*.
  - Perl ofrece `Parse::RecDescent`.
  - Haskell ofrece `Parsec`.
  - Java dispone de JavaCC y ANTLR.
  - C++ dispone de Spirit Parser Generator.
  - . . . siempre se puede escribir a mano.
- Si el lenguaje a reconocer es simple, es más práctico usar esta técnica
  - Lenguaje de un archivo de configuración – INI, JSON, . . .
  - Lenguaje de comandos interactivos.

UNIVERSIDAD SIMÓN BOLÍVAR

Bibliografía

- [*Aho*]
  - Sección 4.4
- WikiPedia – Recursive Descent Parser
- JSON: The Fat-Free alternative to XML

UNIVERSIDAD SIMÓN BOLÍVAR