Compiladores 24-2
Análisis Sintáctico: parsers top-down y manejo de errores

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

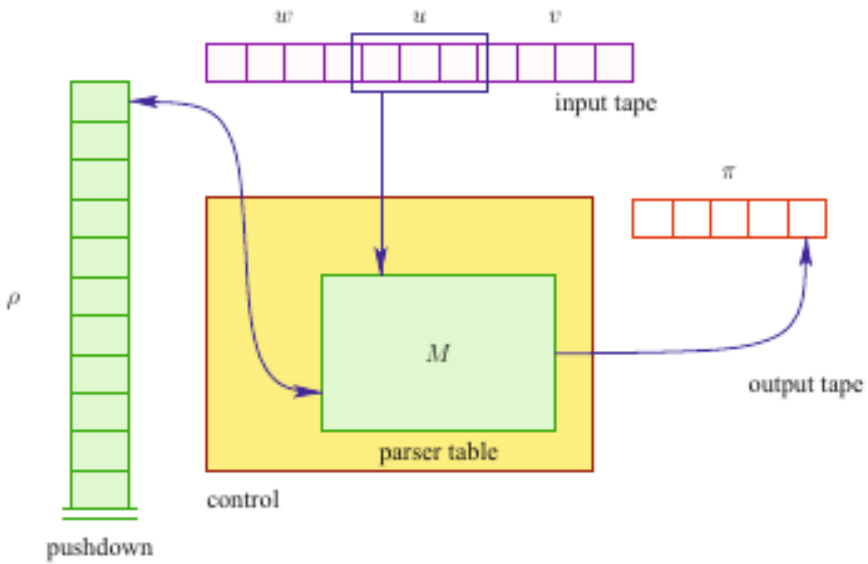
Facultad de Ciencias, UNAM

4 marzo 2024

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, los símbolos leídos por adelantado determinan de forma única la producción a aplicar
  - usar una tabla de predicción para indicar la producción correcta



Un **estado del parser** está conformado por la información de la cadena de entrada y la pila.

Manejo de errores

- Las entradas vacías de la tabla denotan errores que deben reportarse de forma clara o deben ser informativos respecto al error.
- Una vez hecho esto, el analizador debe *recuperarse* y resumir o regreasar a un estado donde el análisis pueda continuar. Esto debe ser tan rápido como sea posible sin agregar complejidad al proceso utilizando la información en la pila y la cadena de entrada.
- Se pueden diseñar compiladores que reporten o identifiquen la mayor cantidad de errores posibles o que, dado un error, se detenga el proceso de compilación.

Manejo de errores

- Las entradas vacías de la tabla denotan errores que deben reportarse de forma clara o deben ser informativos respecto al error.
- Una vez hecho esto, el analizador debe *recuperarse* y resumir o regreasar a un estado donde el análisis pueda continuar. Esto debe ser tan rápido como sea posible sin agregar complejidad al proceso utilizando la información en la pila y la cadena de entrada.
- Se pueden diseñar compiladores que reporten o identifiquen la mayor cantidad de errores posibles o que, dado un error, se detenga el proceso de compilación.
- Cuando no se tiene una predicción por la tabla de parsing se puede descartar el símbolo que se está analizando y por lo tanto detener el proceso de construcción del parse tree.
- Se puede afrontar esta falla de diferentes formas:
  - construir un parse tree incompleto al ignorar el no-terminal en la pila o ignorar los siguientes tokens de la cadena de entrada
  - continuar con el análisis al *arreglar* el error agregando un símbolo

Manejo de errores

Si dado el símbolo no-terminal X, la entrada $M[X, a]$ está vacía, aplicar el método de conjunto-aceptable para tener un método de recuperación al utilizar un conjunto de tokens aceptable.

1. usar alguna estrategia o algoritmo para construir un conjunto-aceptable T tomando el estado del parser (incluir $\# \in T$)
2. ignorar tokens de la cadena de entrada hasta encontrar un token en T
3. reestablecer el parser en un estado donde se pueda analizar el token encontrado mediante otra estrategia o algoritmo que esté relacionado con el del paso 1.

El conjunto-aceptable es llamado *follow set* y no tiene relación con la función FOLLOW, sólo determina el siguiente estado para continuar con el análisis sintáctico.
Las siguientes estrategias corresponden a los casos de los algoritmos usados en estos pasos.

Manejo de errores

Estrategia: Modo pánico

Caso 1:

En el tope de la pila aparece un símbolo no-terminal X que no coincide con el siguiente token, entonces descartar tokens hasta que uno pertenezca al conjunto $SYNCH(X)$ (sincronizar el token leido con el tope de la pila).

Manejo de errores

Estrategia: Modo pánico

Caso 1:

En el tope de la pila aparece un símbolo no-terminal X que no coincide con el siguiente token, entonces descartar tokens hasta que uno pertenezca al conjunto $SYNCH(X)$ (sincronizar el token leido con el tope de la pila).

La decisión de qué elementos pertenecen a $SYNCH(X)$ puede guiarse por las siguientes estrategias:

Manejo de errores

Estrategia: Modo pánico

Caso 1:

En el tope de la pila aparece un símbolo no-terminal X que no coincide con el siguiente token, entonces descartar tokens hasta que uno pertenezca al conjunto $SYNCH(X)$ (sincronizar el token leido con el tope de la pila).

La decisión de qué elementos pertenecen a $SYNCH(X)$ puede guiarse por las siguientes estrategias:

1. Incluir elementos de $\textsc{Follow}(X)$. Si encontramos un token que pertenzca a este conjunto, podemos saltarnos la derivación de X y resumir el análisis. El error a reportar involucra la derivación de X: “Se espera un ...”

Manejo de errores

Estrategia: Modo pánico

Caso 1:

En el tope de la pila aparece un símbolo no-terminal X que no coincide con el siguiente token, entonces descartar tokens hasta que uno pertenezca al conjunto $SYNCH(X)$ (sincronizar el token leido con el tope de la pila).

La decisión de qué elementos pertenecen a $SYNCH(X)$ puede guiarse por las siguientes estrategias:

1. Incluir elementos de $\textsc{Follow}(X)$. Si encontramos un token que pertenzca a este conjunto, podemos saltarnos la derivación de X y resumir el análisis. El error a reportar involucra la derivación de X: “Se espera un ...”
2. En el caso de lenguajes que definen una jerarquía, se pueden incluir símbolos de la jerarquía superior.

Manejo de errores

Estrategia: Modo pánico

Caso 1:

En el tope de la pila aparece un símbolo no-terminal X que no coincide con el siguiente token, entonces descartar tokens hasta que uno pertenezca al conjunto $SYNCH(X)$ (sincronizar el token leido con el tope de la pila).

La decisión de qué elementos pertenecen a $SYNCH(X)$ puede guiarse por las siguientes estrategias:

1. Incluir elementos de $\textsc{Follow}(X)$. Si encontramos un token que pertenzca a este conjunto, podemos saltarnos la derivación de X y resumir el análisis.
   El error a reportar involucra la derivación de X: “Se espera un ...”
2. En el caso de lenguajes que definen una jerarquía, se pueden incluir símbolos de la jerarquía superior.
3. Incluir elementos de $\textsc{First}(X)$. Si encontramos un token que pertenezca a este conjunto, podemos resumir el análisis en ese estado.
   Se reporta el error con los tokens que fueron descartados: “Tokens inesperados ...”

Manejo de errores

Estrategia: Modo pánico

Caso 1:

En el tope de la pila aparece un símbolo no-terminal X que no coincide con el siguiente token, entonces descartar tokens hasta que uno pertenezca al conjunto $SYNCH(X)$ (sincronizar el token leido con el tope de la pila).

La decisión de qué elementos pertenecen a $SYNCH(X)$ puede guiarse por las siguientes estrategias:

1. Incluir elementos de $\textsc{Follow}(X)$. Si encontramos un token que pertenzca a este conjunto, podemos saltarnos la derivación de X y resumir el análisis.
   El error a reportar involucra la derivación de X: “Se espera un ...”
2. En el caso de lenguajes que definen una jerarquía, se pueden incluir símbolos de la jerarquía superior.
3. Incluir elementos de $\textsc{First}(X)$. Si encontramos un token que pertenezca a este conjunto, podemos resumir el análisis en ese estado.
   Se reporta el error con los tokens que fueron descartados: “Tokens inesperados ...”
4. Si X puede derivar la cadena vacía, usar esa producción.

Manejo de errores

Estrategia: Modo pánico

Caso 2:
Cuando el tope de la pila es un símbolo terminal que no coincide con el siguiente token, la estrategia más simple es sacarlo de la pila y marcar un error de inserción usando el tope de la pila γ:
“Se esperaba token γ ...”

Manejo de errores

Modo pánico, ejemplo

De la gramática de expresiones aritméticas se incluyen conjuntos de sincronización en algunas entradas

$$
\begin{array}{lcllcllcl}
E & \to & TE' & E' & \to & +TE' & E' & \to & \varepsilon \\
T & \to & FT' & T & \to & F & T' & \to & *FT' \\
T' & \to & \varepsilon & F & \to & (E) & F & \to & id
\end{array}
$$

Var/input	n	$+$	$*$	$($	$)$	$\#$
E	$E \to TE'$			$E \to TE'$	$SYNCH(E)$	$SYNCH(E)$
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$	$SYNCH(T)$		$T \to FT'$	$SYNCH(T)$	$SYNCH(T)$
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to n$	$SYNCH(F)$	$SYNCH(F)$	$F \to (E)$	$SYNCH(F)$	$SYNCH(F)$

$\textsc{First}(E) = \{(, n\}$ $\quad$ $\textsc{Follow}(E) = \{\#,)\}$

$\textsc{First}(F) = \{(, n\}$ $\quad$ $\textsc{Follow}(F) = \{*, +, \#,)\}$

Manejo de errores

Modo pánico, ejemplo

De la gramática de expresiones aritméticas, la cadena $n + *n$ es errónea:

reconocido	*stack*	*entrada*	*accion*
ε	$E\#$	$n + *n\#$	
ε	$TE'\#$	$n + *n\#$	
ε	$FT'E'\#$	$n + *n\#$	
ε	$nT'E'\#$	$n + *n\#$	
n	$T'E'\#$	$+ * n\#$	
n	$\varepsilon E'\#$	$\varepsilon + *n\#$	
n	$E'\#$	$+ * n\#$	
n	$+TE'\#$	$+ * n\#$	
$n+$	$TE'\#$	$*n\#$	error
			$*$ no pertenece ni a FOLLOW ni a FIRST
			descartar el token con mensaje
			error : “Simbolo * inesperado”
$n+$	$TE'\#$	$n\#$	n está en el FIRST(T), resumir/restaurar parser
$n+$	$FT'E'\#$	$n\#$	
$\vdots$	$\vdots$	$\vdots$	
$n+$	$n\#$	$n\#$	
$n + n$	$\#$	$\#$	pila vacía, termina el análisis

Manejo de errores

Estrategia: Recuperación a nivel de frase

En esta estrategia las entradas vacías de la tabla de análisis son llenadas con funciones que manipulen la pila y/o la entrada para corregir el estado actual y resumir el análisis.

Manejo de errores

Estrategia: Recuperación a nivel de frase

En esta estrategia las entradas vacías de la tabla de análisis son llenadas con funciones que manipulen la pila y/o la entrada para corregir el estado actual y resumir el análisis.

Esta estrategia deriva una cadena distinta al corregirla, sin embargo, las funciones a usar deben ser diseñadas con cuidado para evitar caer en un loop infinito cuando no se descarta un elemento de la pila o entrada.
La implementación de las funciones es decisión del diseñador del compilador y está fuertemente relacionada a la gramática.

Manejo de errores

Recuperación a nivel de frase,ejemplo

En el ejemplo anterior, $n + *n$, la entrada de la tabla $M[T, *]$ puede tener una función que inserte un token de identificador nuevo con el objetivo de reconocer la expresión completa:

reconocido	*stack*	*entrada*	*accion*
$\vdots$	$\vdots$	$\vdots$	
$n+$	$TE'\#$	$*n\#$	*error* : T esperaba un identificador $M[T, *]$ crea un identificador nuevo **n** se reporta el identificador
$n+$	$TE'\#$	$\mathbf{n} * n\#$	resume análisis
$n+$	$FT'E'\#$	$\mathbf{n} * n\#$	
$n+$	$nT'E'\#$	$\mathbf{n} * n\#$	
$n + \mathbf{n}$	$T'E'\#$	$*n\#$	
$n + \mathbf{n}$	$*FT'E'\#$	$*n\#$	
$n + \mathbf{n}*$	$FT'E'\#$	$n\#$	
$n + \mathbf{n}*$	$nT'E'\#$	$n\#$	
$n + \mathbf{n} * n$	$T'E'\#$	$\#$	
$n + \mathbf{n} * n$	$E'\#$	$\#$	
$n + \mathbf{n} * n$	$\#$	$\#$	

Referencias

Imagen tomada de libro “ Compiler Design”, capítulo 3.
Ejemplo tomado del libro “Compilers, Principles, Techniques and Tools”, capítulo 4.

[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] D. Grune and C. J. H. Jacobs.
Parsing Techniques: A Practical Guide.
Ellis Horwood, USA, 1990.

[3] D. Grune, K. v. Reeuwijk, H. E. Bal, C. J. Jacobs, and K. Langendoen.
Modern Compiler Design.
Springer Publishing Company, Incorporated, 2nd edition, 2012.

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

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

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