Analizadores sintácticos
Construcción con Yacc II

M. Luisa González Díaz
Departamento de Informática
Universidad de Valladolid

1. Conflictos

Como se ha visto, el análisis ascendente por desplazamiento reducción se basa en la localización del pivote de cada forma sentencial derecha, que se traduce en la decisión entre desplazar o reducir, y en el último caso, la regla por la que reducir.

Yacc determina, para cada situación de la pila y cada símbolo de entrada, la acción correspondiente, realizando determinado algoritmo[1] a partir de las reglas de la gramática. Si ésta permite determinar de forma única la acción en cada caso, se dice que es *LALR*(1). Si no, es decir, cuando haya más de una posibilidad en algún caso, en alguna casilla aparecerá más de una acción, y Yacc lo indicará como "*conflicto*". Hay dos tipos de conflicto:

desplazamiento-reducción cuando se obtenga en la casilla una acción de desplazamiento y una (o más) de reducción

reducción-reducción cuando no se obtenga en la casilla acción de desplazamiento, pero sí más de una acción de reducción

1.1. Ejemplo de conflicto desplazamiento-reducción

Consideremos la gramática:

$$\begin{array}{lcl} S & \rightarrow & cAba \mid cbbb \mid bA \\ A & \rightarrow & b \end{array}$$

El lenguaje generado es $\{cbba, cbbb, bb\}$ mediante las derivaciones más a la derecha (únicas):

$$S \Rightarrow \underline{cAba} \Rightarrow c\underline{b}ba$$

$$S \Rightarrow \underline{cbbb}$$

$$S \Rightarrow \underline{bA} \Rightarrow b\underline{b}$$

Yacc detecta un conflicto desplazamiento-reducción, en la casilla $[4, b]$, que aparece en `y.output` de la siguiente manera:

```
4: shift/reduce conflict (shift 8, reduce 4) on 'b'
state 4
        S : 'c' 'b' . 'b' 'b'  (2)
        A : 'b' .  (4)
```

El problema surge cuando se realiza el análisis de las sentencias *cbba* y *cbbb*:

[1] Sección 4.7 de "Compiladores. Principio, Técnicas y Herramientas", Aho, Sethi y Ullman

Pila	E ntrada	Acción	Pila	Entrada	Acción
$	c bba$	desplazar	$	c bbb$	desplazar
$c	b ba$	desplazar	$c	b bb$	desplazar
$cb	b a$	reducir 4 ($A \to b$)	$cb	b b$	desplazar
$cA	b a$	desplazar	$cbb	b $	desplazar
$cAb	a $	desplazar	$cbbb	$	reducir 2($S \to cbbb$)
$cAba	$	reducir 1($S \to cAba$)	$S	$	aceptar
$S	$	aceptar			

Los análisis empiezan a diferenciarse en la tercera línea, en la que Yacc debería tomar la decisión adecuada teniendo a la vista *solamente un símbolo* de la entrada (en este caso, *b*). Como la decisión depende realmente de un símbolo posterior (en este caso, el siguiente, *a* ó *b*), Yacc no puede decidir entre desplazamiento y reducción. La situación exacta de aparición del conflicto es *cb*.*b* . . .

Un problema similar se encuentra en la siguiente gramática:

$$\begin{array}{lcl} S & \to & cAba \mid cBbb \mid bB \\ A & \to & b \\ B & \to & \lambda \end{array}$$

Ahora el conflicto es

```
1: shift/reduce conflict (shift 4, reduce 5) on 'b'
state 1
        S : 'c' . A 'b' 'a'  (1)
        S : 'c' . B 'b' 'b'  (2)
        B : .  (5)
```

y aparece en la situación *c*.*b* . . . entre desplazar (para buscar el pivote *b* para *cbba*), o reducir (por la regla $B \to \lambda$, porque ya se ha encontrado el consecuente del pivote λ para $cba = c\lambda ba$)

1.2. Ejemplo de conflicto reducción-reducción

La gramática

$$\begin{array}{lcl} S & \to & cAba \mid cBbb \mid bB \\ A & \to & b \mid a \\ B & \to & b \end{array}$$

produce un conflicto reducción-reducción:

```
4: reduce/reduce conflict (reduce 4, reduce 6) on 'b'
state 4
        A : 'b' .  (4)
        B : 'b' .  (6)
```

que se produce, en la situación *cb*.*b* . . . como se ve en las siguientes simulaciones:

Pila	Entrada	Acción	Pila	Entrada	Acción
$	c bba$	desplazar	$	c bbb$	desplazar
$c	b ba$	desplazar	$c	b bb$	desplazar
$cb	b a$	reducir 4 ($A \to b$)	$cb	b b$	reducir 6 ($B \to b$)
$cA	b a$	desplazar	$cB	b b$	desplazar
$cAb	a $	desplazar	$cBb	b $	desplazar
$cAba	$	reducir 1	$cBbb	$	reducir 2

Nuevamente la decisión depende de un símbolo que aparecerá más tarde.

1.3. Gramáticas ambiguas

En los ejemplos anteriores, cada forma sentencial derecha tenía un único pivote, y el problema surgía al intentar localizarlo, viendo sólo una parte de los datos.

Si la gramática es ambigua, para alguna forma sentencial existirá más de un árbol de derivación, luego más de una derivación más a la derecha, y ello se traducirá en que tal forma sentencial tendrá más de un pivote. Por lo tanto se detectará, por fuerza, algún conflicto.

Recordemos que un pivote consta de dos partes: una subcadena (con su posición) y una regla cuyo consecuente es dicha subcadena: $< (\beta, p), A \rightarrow \beta >$. El análisis por desplazamiento-reducción debe desplazar símbolos de entrada hasta colocar el pivote en la cima de la pila, momento en el que deberá realizar una reducción.

Por ejemplo, la gramática para expresiones

$$G_5 \left\{ \; E \quad \rightarrow \quad E - E \mid \textbf{núm} \right.$$

presenta un conflicto desplazamiento-reducción por su ambigüedad. Para la cadena **núm** - **núm** - **núm**, hay dos derivaciones más a la derecha:

$$E \Rightarrow \underline{E - E} \Rightarrow E - \underline{\textbf{núm}} \Rightarrow \underline{E - E} - \textbf{núm} \Rightarrow E - \underline{\textbf{núm}} - \textbf{núm} \Rightarrow \underline{\textbf{núm}} - \textbf{núm} - \textbf{núm}$$

$$E \Rightarrow \underline{E - E} \Rightarrow E - \underline{E - E} \Rightarrow E - E - \underline{\textbf{núm}} \Rightarrow E - \underline{\textbf{núm}} - \textbf{núm} \Rightarrow \underline{\textbf{núm}} - \textbf{núm} - \textbf{núm}$$

La forma sentencial $E - E -$ **núm** tiene dos pivotes, para las subcadenas $E - E$ y **núm**, correspondientes a las intepretaciones de cálculo (**núm**-**núm**)-**núm** y **núm**-(**núm**-**núm**) respectivamente. Cuando el análisis llegue a la situación

Pila	Entrada	Acción
$	**núm** - **núm** - **núm** $	desplazar
$ **núm**	-**núm**-**núm**$	reducir
$ E	- **núm**-**núm**$	desplazar
$ E -	**núm**-**núm**$	desplazar
$ E-**núm**	- **núm**$	reducir
$E-E	- **núm**	?

la elección entre reducir o desplazar determinará la elección entre la la primera y segunda de las interpretaciones respectivamente, es decir, entre la asociación por la izquierda o derecha del operador -. Tal elección no podría ser resuelta leyendo más símbolos de la entrada.

Las siguientes gramáticas, ambiguas, producen conflictos por ello de tipo desplazamiento-reducción

$$\left\{ \begin{array}{lcl} S & \rightarrow & Bb \mid bB \\ B & \rightarrow & bb \end{array} \right. \qquad \left\{ \begin{array}{lcl} S & \rightarrow & cAb \mid cA \\ A & \rightarrow & bb \mid b \end{array} \right.$$

y en las siguientes, la ambigüedad produce conflitcos reducción-reducción:

$$\left\{ \begin{array}{lcl} S & \rightarrow & cAbb \mid cBbb \\ A & \rightarrow & b \\ B & \rightarrow & b \end{array} \right. \qquad \left\{ \begin{array}{lcl} S & \rightarrow & cA \mid cbb \\ A & \rightarrow & bb \end{array} \right. \qquad \left\{ \begin{array}{lcl} S & \rightarrow & abBc \mid Ac \\ A & \rightarrow & ab \\ B & \rightarrow & \lambda \end{array} \right.$$

1.4. Resolución de conflictos

Cuando Yacc detecta un conflicto (bien por ambigüedad de la gramática o por otros motivos), aún genera un analizador sintáctico resolviéndolos sistemáticamente en favor de los desplazamientos, o eligiendo la primera regla que aparezaca en la gramática para conflictos reducción-reducción. En `y.output` pueden comprobarse estas decisiones.

Como consecuencia, es posible que haya cadenas del lenguaje que el analizador no reconozca como correctas. En primer ejemplo del apartado 1.1 no se reconocerá la cadena *cbba*, ni en el primero del apartado 1.2 la cadena *cbbb*.

Algunas veces, es fácil darse cuenta de esta reducción del lenguaje, porque puede ocurrir que una de las reglas nunca llegue a usarse, y Yacc lo indica mediante el mensaje "`1 rule never reduced`" (por ejemplo, si se suprime la tercera regla de las gramáticas de los apartados 1.1 y 1.2).

En general, lo más recomendable es modificar la gramática para que no aparezcan conflictos. Nuevamente para los ejemplos de los apartados 1.1 y 1.2, se podrían utilizar las gramáticas equivalentes *LALR*(1) siguientes:

$$\begin{cases} S \to cABA \mid cABb \mid bA \\ A \to a \\ B \to b \end{cases} \qquad \begin{cases} S \to cbbA \mid bB \\ A \to a \mid \lambda \\ B \to b \end{cases} \qquad \begin{cases} S \to cAa \mid cBb \mid B \\ A \to bb \mid ab \\ B \to bb \end{cases}$$

Para gramáticas ambiguas, muy frecuentemente la técnica de resuloción de conflictos de Yacc supone sencillamente la elección de una de las posibles derivaciones más a la derecha para la cadena de entrada, con lo que el lenguaje reconocido sigue siendo el mismo, aunque se elige una de las interpretaciones o estructuras, entre varias posibles.

Este es el caso de la gramática de expresiones aritméticas de 1.3: la elección del desplazamiento en lugar de la reducción supone la elección del segundo de los árboles que siguen para **núm** - **núm** - **núm** (con lo que 7 – 5 – 2 tomaría el valor 4), pero no reduciría el conjunto de cadenas reconocidas.

La elección de resolución de conflictos en favor de los desplazamientos favorece que los árboloes tengán más desarrollo en su parte derecha, y que lo operadores sean asociativos a la derecha, como en este ejemplo. Como muchos operadores son asociativos por la izquierda, la política de resolución de conflictos en Yacc no será suficiente. Por lo tanto, o se formula una gramática no ambigua que contemple las asociatividades deseadas, o se indica a Yacc especificamente de qué forma se desea la resolución de conflictos.

Además de un problema de asociatividad de operadores puede aparecer también un problema de precedencias. Si contemplamos en las expresiones aritméticas el operador producto, la mera especificación

$$E \to E + E \mid E - E \mid E * E \mid \textbf{núm}$$

hará también ambigua la expresión **núm**+ **núm*** **núm** y Yacc interpretaría (**núm**+ **núm**)* **núm** en contra de lo que normalmente significa. Se desea el operador * también asociativo por la izquierda, pero con precedencia sobre el + .

La gramática

$$\begin{aligned} E &\to E + T \mid E - T \mid T \\ T &\to T * F \mid T/F \mid F \\ F &\to \textbf{núm} \mid (E) \end{aligned}$$

es inambigua y mantiene las precedencias y asociatividades deseadas. Además, es también *LALR*(1), con lo que Yacc podrá generara un analizador sin problemas. En general, las reglas *recursivas por la izquierda* ($A \to A\alpha$) hacen asociativos a la izquierda a los operadores, mientras que las *recursivas por la derecha* ($A \to \alpha A$) los hacen asociativos a la derecha. El orden de precedencia se resuelve

mediante el uso de distintos niveles de sustitución: la "expresión" (E) aparece como suma o resta de "términos" (T), y éstos como producto o cociente de "factores" (F), que son finalmente expresiones elementales (**núm**) o bien expresiones entre paréntesis.

Otras posibilidad es especificar en Yacc el modo en que se quiere resolver los conflictos, proporcionando información sobre asociatividad y precedencia. En la sección de declaraciones se pueden asociar precedencias y asociatividades a los componentes léxicos con líneas de la forma `%left`, `%right` o `%nonassoc`. Todos los componentes léxicos de una misma línea tendrán la misma precedencia, y las líneas deben escribirse en orden creciente de precedencia. Por ejemplo:

```
%left '+' '-'
%left '*' '/'
```

`%nonassoc` se usa para especificar operadores no asociativos (por ejemplo, los relacionales <, <= ...) Las reglas también llevarán asociadas una precedencia y asociatividad, que será la que tenga el último componente léxico del consecuente. Los componentes léxicos pueden o no estar declarados también en una línea `%token`.

A veces se necesita que operadores unarios de alta precedencia utilicen la misma representación simbólica que otros binarios de precedencia menor, como el "-" de cambio de signo respecto al binario de resta. La palabra clave `%prec` cambia el nivel de precedencia asociado en una regla particular, cuando aparece inmediatamente a continuación del cuerpo de la regla, antes de la acción o el ";" de cierre. Provoca que la regla tenga la precedencia que se especifica en `%prec`.

Por ejemplo, las expresiones aritméticas podrían describirse en Yacc:

```
%token NUM

%left '+' '-'
%left '*' '/'
%%
expr : expr '+' expr
     | expr '-' expr
     | expr '*' expr
     | expr '/' expr
     | '-' expr %prec '*'
     | NUM
     ;
```

que está estableciendo 2 niveles de precedencia (para operadores aditivos y multiplicativos, respectivamente), ambos con asociatividad izquierda, y además el operador unario "-" que tendrá la misma precedencia que los multiplicativos.

Yacc ahora resolverá los conflictos con el siguiente algoritmo:

1. se asignará a cada componente léxico las asociatividades y precedencias que se hayan especificado (algunos pueden no tenerlas)

2. cada regla tendrá asociada la asociatividad y precedencia del último componente léxico del consecuente (si lo tiene y éste tiene asignadas asociatividad y precedencia), salvo que se especifique otra cosa mediante `%prec`

3. ante un conflicto reducción-reducción, o un conflicto desplazamiento-reducción cuando las posibles reglas a usar en reducciones o el símbolo de entrada no tienen asignadas precedencia y asociatividad, se actuará como se expuso al principio de esta sección

4. ante un conflicto desplazamiento-reducción, en el que tanto el símbolo de entrada como la regla a emplear en la reducción tienen asignadas asociatividad y precedencia, el conflicto se resolverá en favor de la acción de desplazamiento si el símbolo de entrada tiene más precedencia que la regla, y en favor de la reducción si ésta lo tiene mayor. A igualdad de precedencia,

se optará por la reducción si la asociatividad es izquierda, desplazamiento si es derecha o se marcará un error si está especificada no asociatividad.

Los conflictos resueltos por precedencia no se tomarán en cuenta en el resumen de número de conflictos encontrados (lo que aún podría disfrazar errores).