Nueva estrategia para la recuperación de errores sintácticos*

Henry F. Báez**
Juan G. Vargas***

Strategy for syntax error recovering

Resumen

Este artículo describe una nueva estrategia de recuperación de errores sintácticos para un compilador de lenguaje que no utiliza separadores de instrucciones como el punto y coma ';' o los corchetes de apertura '{' y de cierre '}'. Esta estrategia se desarrolla en 4 pasos: 1. encontrar un conjunto de tokens (llamado conjunto ACEPTA) para cada símbolo no terminal de la gramática; 2. en el análisis sintáctico de cada símbolo no terminal, se eliminan tokens que no se encuentren en el conjunto ACEPTA; 3. eliminar tokens repetidos que no son aceptados por la gramática y 4. completar símbolos en el análisis sintáctico con la esperanza que el token que no se ha borrado coincidirá más adelante con un símbolo terminal esperado por el análisis sintáctico. En caso contrario, el token se eliminará en algunas producciones determinadas.

La estrategia de recuperación de errores sintácticos es una metodología que puede usarse en cualquier gramática libre de contexto y no ambigua, incluso en las que utilizan separadores de instrucciones como el ';'. Se implementa de forma algorítmica y es mucho más fácil de implementar que otras estrategias clásicas como las basadas en pilas.

PALABRAS CLAVE

Compilador, analizador sintáctico, gramática, recuperación de errores sintácticos

Abstract

This paper describes a new strategy for syntax error recovering for a complier that does not have instruction separators like ';' or opening and closing brackets like '{' and '}'. This strategy is based on 4 steps. 1. Find a set of tokens (called ACEPTA set) for each non terminal symbol of the grammar 2. During the syntax analysis of each non terminal symbol, eliminate the tokens that are not in the ACEPTA set. 3. Eliminate repeated tokens that are not accepted by the grammar, and 4. Complete symbols in the syntax analysis with the hope that the token that has not been erased later will match with a terminal symbol expected by the syntax analyser; otherwise the symbol will be eliminated in some particular productions.

The strategy for syntax error recovering is a method that can be used in whatever not ambiguous context free grammar includying those that use instruction separators like ';'. It is implemented with an algorithm and it is much more easy to implement than other strategies for syntax error recovering like those based on stacks.

KEY WORDS

Compiler, syntax analizer, grammar, syntax error recovering

* Tesis meritoria.

** Ingeniero de Sistemas, Universidad Nacional de Colombia. henrybaez@softhome.net.

*** Ingeniero de Sistemas, Universidad Nacional de Colombia. gabrielvargas@terra.com.co.

INTRODUCCIÓN

A grandes rasgos un compilador es un programa que lee código de un algoritmo escrito en un lenguaje fuente y lo traduce a código equivalente en un lenguaje objeto. Para realizar esta traducción, el compilador debe verificar que el código del lenguaje fuente esté correctamente escrito, de lo contrario informará al usuario sobre la presencia de errores.

Los compiladores se pueden diseñar de forma que detengan la compilación al encontrar el primer error. Para un programador no es muy práctico tener que compilar el código fuente por cada error que haya cometido. La recuperación de errores consiste en no detener el proceso de compilación cuando se encuentre el primer error, y su objetivo es encontrar los errores restantes. El analizador sintáctico detecta un error de sintaxis cuando el analizador léxico proporciona un token [1] y éste es incompatible con el estado actual del analizador sintáctico. No hay estrategias definidas para la recuperación de errores sintácticos, y se usan heurísticas basadas en predecir lo que quiso decir el programador al escribir determinada construcción.

Este artículo describe una nueva estrategia de recuperación de errores sintácticos para un compilador. Esta estrategia fue diseñada e implementada en un compilador de SeudoCódigo realizado como parte del trabajo de grado para optar al título de ingeniero de sistemas titulado Ayudas Didácticas para el Curso Virtual de Programación de Computadores. El SeudoCódigo es el lenguaje utilizado para la enseñanza de la lógica de programación de computadores en la Universidad Nacional de Colombia, el cual no utiliza separadores de instrucciones como el punto y coma '**;**' ni los corchetes de apertura '**{**' y de cierre '**}**'.

Debido a que en la teoría clásica de compiladores no se encontró ninguna estrategia de recuperación de errores sintácticos que no involucrara separadores de instrucciones, entonces se recurrió a diseñar una estrategia propia que recuperara este tipo de errores, la cual es generalizada y aplicable a cualquier gramática libre de contexto y no ambigua, incluidas las que utilizan separadores de instrucciones.

1. DEFINICIONES

Para la estrategia de recuperación de errores sintácticos se define:

- *Producción:* corresponde a cada uno de los grafos sintácticos de Wirth [2].
- *Token léxico:* elemento léxico retornado por el analizador léxico.
- *Token sintáctico:* elemento léxico esperado por el analizador sintáctico.
- *Producción INICIAL:* se define en diagramas de Wirth como lo indica la figura 1:

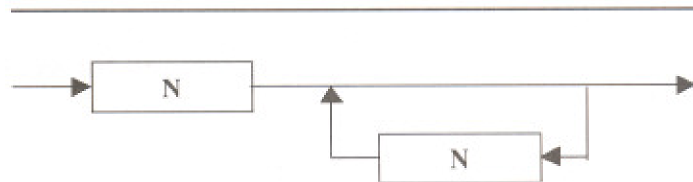

Figura 1. Producción INICIAL para la recuperación de errores sintácticos.

donde **N** era originalmente el símbolo inicial de la gramática.

- *Conjunto ACEPTA:* conjunto de tokens sintácticos definidos para cada símbolo no terminal. Los tokens que pertenecen a este conjunto no son eliminados en la producción.
- *Salto:* sucede cuando el analizador léxico retorna un token léxico que pertenece al conjunto ACEPTA de la producción. Si este token léxico no corresponde con el token sintáctico, entonces se conserva el token léxico y se salta el token sintáctico, y se informa el error. Se denota en los diagramas de Wirth con una flecha punteada que salta el token sintáctico. No se pueden saltar los tokens sintácticos con los que inicia una producción. Cuando se llega a un símbolo no terminal obligatorio dentro del análisis sintáctico y el token léxico que se tiene no corresponde a ninguno de los símbolos iniciales de tal símbolo no terminal, no se entra a la producción y se informa el error. Para esta última situación no se coloca notación en los diagramas de Wirth, ya que a todos los símbolos no terminales obligatorios se les aplica *salto*.

- *Desbordamiento:* se presenta cuando un token léxico que pertenece al conjunto ACEPTA de una producción no se puede empatar con ningún token sintáctico. Por la definición de *Salto,* este token léxico empieza a ascender hacia el símbolo inicial de la gramática, informando errores que no corresponden con los verdaderos errores que contiene la cadena de entrada.
- *Diagrama de producciones:* es un árbol resultante de la expansión de los símbolos no terminales a partir de símbolo inicial, el cual formará la raíz. Los símbolos no terminales sólo se expanden la primera vez que aparecen en el árbol.
- *Conjunto CONTENIDOTOTAL:* conjunto de tokens sintácticos que se pueden encontrar al expandir completamente una producción. Se utiliza para evitar el desbordamiento dentro del proceso de recuperación de errores.
- *Bucle:* se utiliza para eliminar los tokens léxicos que no pueden repetirse de forma consecutiva en la gramática. Se denota en los diagramas de Wirth como una flecha circular que sale y llega a un mismo token sintáctico.

2. Conjuntos ACEPTA y CONTENIDOTOTAL

Antes de empezar con los pasos para la recuperación de errores sintácticos, se procede a describir cómo encontrar los conjuntos ACEPTA y CONTENIDOTOTAL, así como los símbolos no terminales a los que se le define el conjunto CONTENIDOTOTAL. Para ejemplificar la obtención de estos conjuntos se define la siguiente gramática:

$$\alpha \rightarrow m\,\alpha\,\beta\,n \mid \alpha\,n$$
$$\alpha \rightarrow a\,\beta \mid \beta$$
$$\beta \rightarrow b$$

El conjunto ACEPTA está dado por: INICIO $\cup$ CONTENIDO $\cup$ SIGUIENTE $\cup$ POSTERIOR RESERVADASINICIALES. Cada uno de estos conjuntos se define a continuación:

- El conjunto INICIO está formado por los tokens sintácticos que puedan iniciar una producción. En caso que la producción inicie con uno o más símbolos no terminales, los tokens sintácticos que conforman el conjunto INICIO de tales símbolos no terminales harán parte del conjunto INICIO de la producción. Ejemplo:

$$\begin{aligned} \text{INICIO}(\sigma) &= \{ m \cup \text{INICIO}(\alpha) \} \\ &= \{ m \cup a, b \} \\ &= \{ m, a, b \} \\ \text{INICIO}(\alpha) &= \{ a \cup \text{INICIO}(\beta) \} \\ &= \{ a \cup b \} \\ &= \{ a, b \} \\ \text{INICIO}(\beta) &= \{ b \} \end{aligned}$$

Así pues, el conjunto INICIO consta de los tokens sintácticos con los que se puede entrar en determinada producción.

- El conjunto CONTENIDO posee los tokens sintácticos que se encuentran en la producción; si la producción contiene únicamente símbolos no terminales, el conjunto CONTENIDO será igual al conjunto vacío ($\varnothing$). Ejemplo:

$$\begin{aligned} \text{CONTENIDO}(\sigma) &= \{ m, n \} \\ \text{CONTENIDO}(\alpha) &= \{ a \} \\ \text{CONTENIDO}(\beta) &= \{ b \} \end{aligned}$$

- El conjunto SIGUIENTE corresponde a la unión de los conjuntos INICIO de cada uno de los símbolos no terminales que contiene la producción a evaluar. Si la producción no contiene símbolos no terminales, entonces el conjunto SIGUIENTE será igual al conjunto vacío ($\varnothing$). Ejemplo:

$$\begin{aligned} \text{SIGUIENTE}(\sigma) &= \{ \text{INICIO}(\alpha) \cup \text{INICIO}(\beta) \} \\ &= \{ a, b \cup b \} \\ &= \{ a, b \} \\ \text{SIGUIENTE}(\alpha) &= \{ \text{INICIO}(\beta) \} \\ &= \{ b \} \\ \text{SIGUIENTE}(\beta) &= \{ \varnothing \} \end{aligned}$$

- El conjunto POSTERIOR para una producción (denotada como δ), corresponde al conjunto de tokens sintácticos que pueden seguir inmediatamente al terminar dicha producción. Para poder determinar tales tokens es necesario ubicar en toda la gramática

los símbolos no terminales (denotados como τ) que representan a δ. Si después de τ, sigue otro símbolo no terminal que representa una producción (denotada como γ), entonces los tokens sintácticos que conforman al conjunto INICIO(γ) pertenecen al conjunto POSTERIOR(δ); si tenemos una producción (denotada como φ), donde el último símbolo no terminal es τ, entonces el conjunto POSTERIOR (δ) contendrá los elementos del conjunto POSTERIOR (φ). Si se encuentra un token sintáctico después de τ, este token ingresa al conjunto POSTERIOR(δ). Ejemplo:

$$\begin{aligned} \text{POSTERIOR}(\sigma) &= \{\varnothing\} \\ \text{POSTERIOR}(\alpha) &= \{\text{INICIO}(\beta) \cup n\} \\ &= \{b, n\} \\ \text{POSTERIOR}(\beta) &= \{n \cup \text{POSTERIOR}(\alpha)\} \\ &= \{n \cup b, n\} \\ &= \{n, b\} \end{aligned}$$

- El conjunto RESERVADASINICIO es único para toda la gramática. Está compuesto por las palabras reservadas que inician una producción. En la gramática ejemplo no existen palabras reservadas, por lo tanto este conjunto es igual al conjunto vacío ($\varnothing$).
- El conjunto CONTENIDOTOTAL contiene todos los tokens sintácticos que pueden ser obtenidos al expandir totalmente una producción. Ejemplo:

$$\begin{aligned} \text{CONTENIDOTOTAL}(\sigma) &= \{m \cup n \cup \text{CONTENIDOTOTAL}(\alpha) \cup \text{CONTENIDOTOTAL}(\beta)\} \\ &= \{m, n, b, a\} \\ \text{CONTENIDOTOTAL}(\alpha) &= \{\text{CONTENIDOTOTAL}(\beta) \cup a\} \\ &= \{b, a\} \\ \text{CONTENIDOTOTAL}(\beta) &= \{b\} \end{aligned}$$

Para determinar las producciones que definen el conjunto CONTENIDOTOTAL, se toma el diagrama de producciones; luego, para cada hoja, se avanza hacia la raíz como si un token léxico de esta hoja empezara un *desbordamiento*. Entonces se marca la producción donde se quiere interrumpir el desbordamiento. Al final del proceso a las producciones marcadas, se les define el conjunto CONTENIDOTOTAL.

Las producciones a las que se les define el conjunto CONTENIDOTOTAL deben ser las menos posibles. Esto para evitar que la recuperación de errores se asemeje a una recuperación de errores en *modo pánico;* además deben estar a una altura media en el árbol, ya que al estar cerca de las hojas no se abarcan los suficientes símbolos terminales para evitar este modo pánico.

3. Procedimiento para implementar la recuperación de errores sintácticos

Los pasos para la recuperación de errores sintácticos son:

1. Expresar las producciones de la gramática en grafos sintácticos de Wirth.
2. Redefinir el símbolo inicial de la gramática como la producción INICIAL.
3. Aplicar el concepto de bucle a los grafos sintácticos de Wirth.
4. Aplicar el concepto de salto a los grafos sintácticos de Wirth.
5. Encontrar los conjuntos INICIO, CONTENIDO, SIGUIENTE y POSTERIOR de cada producción y el conjunto RESERVADASINICIO de la gramática.
6. Encontrar el conjunto ACEPTA para cada producción, a partir de los conjuntos encontrados en el paso anterior.
7. Elaborar un diagrama de producciones para determinar los símbolos no terminales a los cuales se les define el conjunto CONTENIDOTOTAL.
8. Hallar el conjunto CONTENIDOTOTAL para los símbolos no terminales encontrados en el paso anterior.
9. Implementar el conjunto ACEPTA. Se debe modificar cada producción del analizador sintáctico para que elimine los tokens léxicos que no pertenecen al conjunto ACEPTA de la producción, si se elimina un token léxico se debe informar el error.
10. Implementar el concepto de *salto* definido en los grafos sintácticos de Wirth. En los tokens sintácticos y símbolos no terminales obligatorios se implementa con una sentencia *if– else,* y en los tokens y símbolos opcionales se implementa con una estructura *if*, donde la sentencia *if* representa el camino correcto en el grafo, y la sentencia *else* informa del error.

11. Para los tokens sintácticos a los que se les definió el concepto de bucle, se utiliza una sentencia *while*, que elimina los tokens léxicos repetidos e informa el error. Esta sentencia se implementa dentro de la correspondiente sentencia *if* del numeral anterior para cada producción.
12. Se deben eliminar tokens léxicos para evitar el desbordamiento. Esto se hace al final de las producciones que representan los símbolos no terminales seleccionados en el paso 7. Un token léxico se elimina, si cumple con las siguientes condiciones:
    a. Pertenece al conjunto CONTENIDOTOTAL.
    b. No pertenece al conjunto INICIO.
    c. La gramática no acepta la escritura del token dos o más veces seguidas. (esta condición no aplica para la producción INICIAL).

4. Ejemplo de recuperación de errores sintácticos

Considerando la gramática de la Figura 2, se procede a realizar un ejemplo de la estrategia de recuperación de errores.

Paso1. El conjunto de producciones de la gramática está definido por los siguientes grafos sintácticos de Wirth:

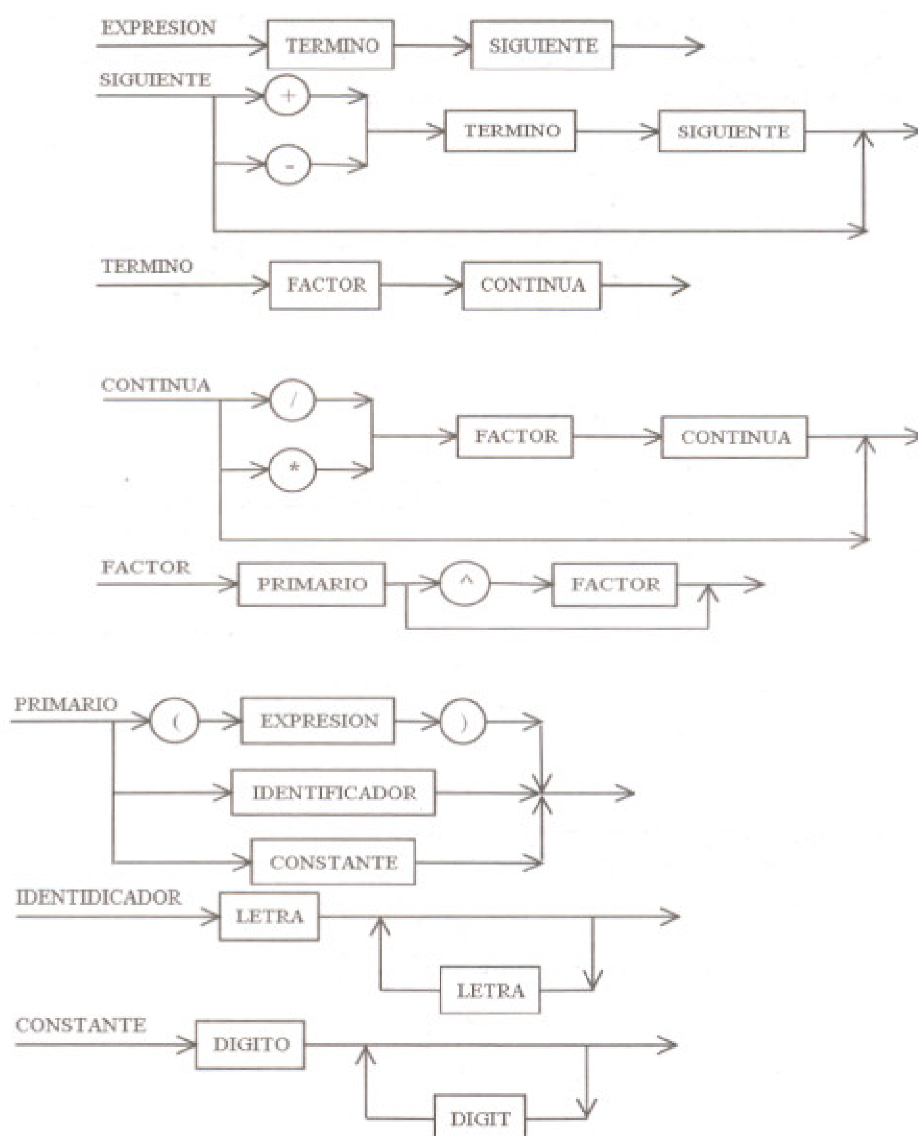

Figura 2. Grafos sintácticos de ejemplo para la recuperación de errores sintácticos.

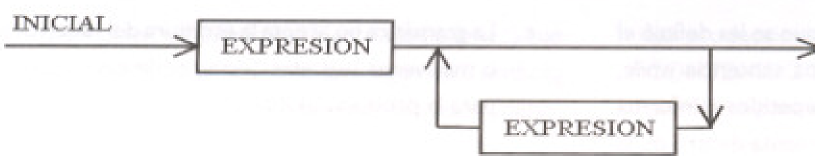

Figura 3. Producción INICIAL para el ejemplo de la recuperación de errores sintácticos.

Paso2. La producción INICIAL debe ser como la que se muestra en la figura 3, en donde se llama una o más veces la producción con que comienza la gramática.

Paso 3 y 4. Se identifican los *bucles* y los *saltos* en cada producción, como se muestra en la figura 4.

Paso 5 y 6. La tabla I contiene los tokens sintácticos pertenecientes a los conjuntos INICIAL, CONTENIDO, SIGUIENTE, POSTERIOR, los cuales conforman el conjunto ACEPTA. En esta gramática no existen palabras reservadas, por lo tanto el conjunto RESERVADAS-INICIO será igual al conjunto vacío ( ).

En la columna (producción) de la tabla I se escribe el nombre de la producción y se señalan con viñetas los nombres de los símbolos no terminales que contiene dicha producción. Las producciones IDENTIFICADOR y CONSTANTE son parte del analizador léxico y hacen referencia a los identificadores y constantes de la gramática, por tanto no son incluidas como producciones, sino como símbolos terminales escritos en minúscula.

Un Identificador se define como una secuencia de letras y dígitos que debe comenzar con una letra. Sirven para identificar nombres de variables, constantes, funciones en un lenguaje de programación. Una constante hace referencia a un número entero de uno o varios dígitos.

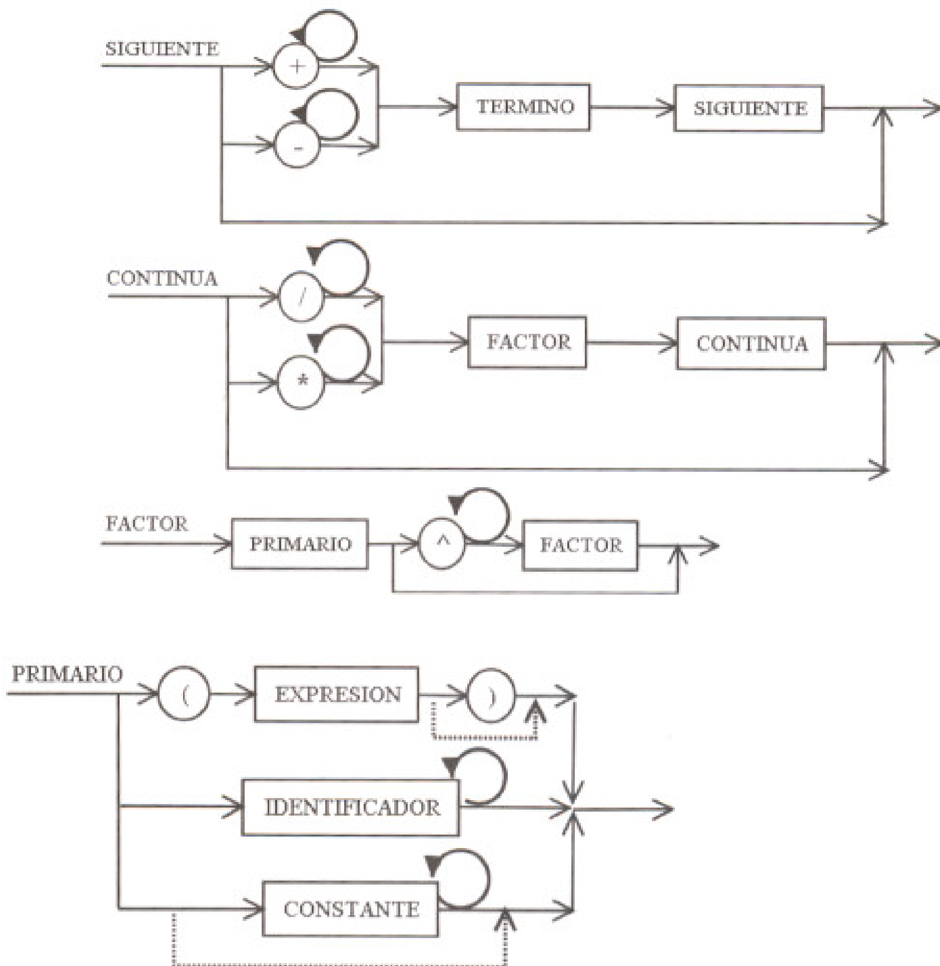

Figura 4. Grafos sintácticos extendidos para la recuperación de errores sintácticos.

Tabla 1. Conjunto ACEPTA para la recuperación de errores sintácticos.

PRODUCCIÓN	ACEPTA			
	INICIO	CONTENIDO	SIGUIENTE	POSTERIOR
INICIAL • EXPRESIÓN	(identificador constante	∅	(identificador constante	∅
EXPRESIÓN • TÉRMINO • SIGUIENTE	(identificador constante + -	∅	(identificador Constante	)
SIGUIENTE • TÉRMINO • SIGUIENTE	+ -	+ -	(identificador constante + -	)
TÉRMINO • FACTOR • CONTINUA	(identificador constante	∅	(identificador Constante * /	+ -
CONTINUA • FACTOR • CONTINUA	* /	* /	(identificador constante * /	+ -
FACTOR • PRIMARIO • FACTOR	(identificador constante	^	(identificador Constante * /	+ -
PRIMARIO • EXPRESIÓN	(identificador constante	( ) identificador constante	(identificador Constante	^ + - * /

Paso 7. Como resultado de este paso, se obtiene que la producción INICIAL es la única que define el conjunto CONTENIDOTOTAL.

Paso 8. La tabla 2 contiene los símbolos terminales pertenecientes al conjunto CONTENIDOTOTAL.

Tabla 2. Conjunto CONTENIDOTOTAL para la recuperación de errores sintácticos

Producción	CONTENIDOTOTAL	
INICIAL	identificador constante ( )	+ - * / ^

Paso 9 – 12. Estos pasos no se escriben a continuación ya que son para la implementación de la estrategia de recuperación de errores en el compilador.

5. Implementación

La estrategia de recuperación de errores sintácticos descrita en este artículo fue implementada en un compilador de SeudoCódigo traductor a C++, el cual está ubicado página web del curso virtual de programación de computadores de la Universidad Nacional de Colombia:

http://www.virtual.unal.edu.co/cursos/ingenieria/25111/index.html (presionar el botón Aplicaciones). o se puede encontrar directamente en: www.virtual.unal.edu.co/cursos/ingenieria/25111/java/compilador_pse_C/Applet1.html

Conclusiones

Esta estrategia de recuperación de errores sintácticos está diseñada para aplicarse en cualquier gramática li-

bre de contexto y no ambigua, incluso a las que utilizan separadores de instrucciones como el ';'.

La estrategia de recuperación de errores sintácticos no es una heurística, sino un proceso algorítmico, y por tanto se puede elaborar una herramienta de software que, dada la gramática, encuentre los conjuntos necesarios para la recuperación de errores sintácticos y, además detecte los símbolos no terminales a los que se les aplica el concepto de 'bucle' y de 'salto'.

Agradecimientos

Al ingeniero Luis Roberto Ojeda por sus enseñanzas sobre la teoría de compiladores y por su colaboración desinteresada en la elaboración de una gramática para el SeudoCódigo y una estrategia de recuperación de errores sintácticos, que fueron un gran aporte para la realización del trabajo de grado para optar al título de ingeniero de sistemas de la Universidad Nacional de Colombia.

A nuestro director de trabajo de grado, el ingeniero Luis Fernando Niño, que a partir de la idea de crear ayudas didácticas, nos condujo hasta el final de este trabajo de grado.

Referencias

[1] Aho, A.; Sethi, R., and Ullman J. (1990). *Compiladores, principios técnicas y herramientas.* Wilmington, Delaware, USA: Addison-Wesley.

[2] Lemone, K. (1996). *Fundamentos de compiladores, cómo traducir al lenguaje de computadora.* Primera edición. Mexico D.F.: Continental.

[3] Pittman, T. and Peters J. (1992). *The Art of Compiler Design, Theory and Practice.* New Jersey: Prentice Hall.

[4] Pressman, R. (1994). *Ingeniería del software, un enfoque práctico.* Tercera edición. Madrid: McGraw Hill.

[5] Sethi, R. (1991). *Lenguajes de programación conceptos y constructores.* México: Addison Wesley.

[6] Schmidt, S.; Teufel, B., and Teufel, T. (1995). *Compiladores, conceptos fundamentales.* Wilmington, Delaware, USA: Addison-Wesley.

[7] Trembay, J., and Sorenson, P. (1996). Tehg theory and practice of compiler writing. 11 edition. Saskatoon, Canada: McGraw Hill.