Universidad de La Laguna

E. T. S. DE INGENIERÍA INFORMÁTICA
Departamento de E.I.O. y Computación
Compiladores

PRACTICA 3: Analizador sintáctico descendente recursivo predictivo para *P*ascal- con recuperación de errores en modo pánico

Factor de ponderación [0-10]: 8

3.1. Objetivos

Las acciones que lleva a cabo el compilador inmediatamente después de descubrir un error sintáctico se conocen como ”recuperación de errores”. El propósito de la recuperación de errores es hacer al compilador capaz de continuar el análisis del programa fuente para encontrar el mayor número de errores posible en un sólo paso y asegurarse de que se emite sólo un mensaje por cada error.

La práctica consiste en la ampliación del analizador sintáctico descendente recursivo predictivo para *P*ascal- con recuperación de errores en modo pánico.

3.2. Descripción

La recuperación de errores que utilizaremos consistirá en ignorar tokens hasta alcanzar uno (el mejor) que permita al analizador sintáctico realizar un buen reconocimiento de la entrada restante. A estos tokens mejores, los llamaremos tokens de parada o sincronización. Se tendrá un conjunto diferente de tokens de parada para las distintas sentencias que reconoce el lenguaje.

A continuación se revisan las reglas para la construcción del parser de forma que contemplen el uso de conjuntos de tokens de parada:

Regla 3.2.1 *Para cada regla de producción de la gramática:* $N \rightarrow \alpha$ *el analizador sintáctico define un procedimiento con el mismo nombre:*
procedure *N(Stop: token_set);*
begin
A(α*, Stop)*
end*;*
El procedimiento define un algoritmo de análisis $A(\alpha$*, Stop) que puede reconocer cualquier*

frase de la forma α y comprueba que la frase va seguida de un símbolo de parada correcto. Los símbolos de parada correctos son precisamente aquellos tokens que el analizador está preparado para reconocer después de una frase de la forma α.

Cuando el algoritmo reconoce la frase α seguida por uno de los símbolos de parada correctos, elimina de la entrada toda la cadena α y el símbolo de parada. Si es incorrecta la sentencia o el símbolo que la sigue no es uno de los de parada, el analizador genera un mensaje de error e ignora símbolos de entrada hasta que alcance uno de los símbolos de parada correctos. En otras palabras, el analizador sintáctico se recupera de un error buscando el primer símbolo que tenga sentido en el contexto en que se encuentra.

Regla 3.2.2 *Una lista de sentencias de la forma $S_1 S_2 \ldots S_n$ seguida por un símbolo de parada se reconoce mediante el análisis individual de cada una de ellas en el mismo orden en el que están escritas:*
$A(S_1 S_2 \ldots S_n, Stop)$:

$A(S_1, First(S_2) \cup First(S_3) \cup \ldots \cup First(S_n) \cup Stop)$;
$A(S_2, First(S_3) \cup First(S_4) \cup \ldots \cup First(S_n) \cup Stop)$;
...
$A(S_n, Stop))$;

Si hay un error sintáctico en S_1, el analizador inmediatamente trata de encontrar el principio de la sentencia S_2. Si no puede encontrar el primer símbolo de S_2, trata de encontrar el primer símbolo de S_3, y así sucesivamente. Si no consigue reconocer nada después del error, ignora los tokens de entrada hasta alcanzar uno de los símbolos de parada de toda la secuencia de sentencias.

Regla 3.2.3 *Cuando el analizador sintáctico espera un único token, t, seguido de un símbolo de parada invoca al siguiente procedimiento:*

```
procedure match(t: token; Stop: token_set);
  begin
  if lookahead = t then
    begin
    lookahead := Lexan;
    syntax_check(Stop)
    end
  else
    Syntax_error(Stop);
end;
```

Después de examinar el token actual el analizador siempre se asegura que el siguiente símbolo sea uno de los símbolos de parada esperados. Esta comprobación la efectúa el procedimiento `syntax_check()`*:*

```
procedure syntax_check(Stop: token_set);
begin
```

```
if not (lookahead in Stop) then
  Syntax_error(Stop)
end;
```

Después de un error sintáctico, el analizador ignora la entrada hasta que alcanza uno de los símbolos de parada, como vemos en el procedimiento `Syntax_error()`*:*

```
procedure Syntax_error(Stop: token_set);
  begin
  writeln("Error sintactico en linea ", lineno);
  while (not(lookahead in Stopi)) do
    lookahead := yylex();
  end;
```

Regla 3.2.4 *Para reconocer un lado derecho en el que aparece un no terminal N, el analizador sintáctico invoca al correspondiente procedimiento llamado N utilizando como símbolos de parada los correspondientes al no terminal:*

A(N, Stop) = N(Stop);

Regla 3.2.5 *Para reconocer una producción descrita con una expresión de la forma $[E]$ seguida por un símbolo de parada se utiliza el siguiente algoritmo:*
A([E], Stop) =
syntax_check(First(E) $\cup$ Stop);
***i**f (lookahead $\in$ FIRST(E)) **t**hen*
A(E, Stop)

En este caso se ha de tener en cuenta que si el algoritmo de análisis anterior va precedido de otro que haga que $lookahead \in (First(E) \cup Stop)$ entonces la llamada inicial a `syntax_check()` *puede omitirse.*

Regla 3.2.6 *Para reconocer una producción descrita con una expresión de la forma $\{E\}$ seguida por un símbolo de parada, usar el siguiente algoritmo:*
A($\{E\}$, Stop) =
syntax_check(First(E) $\cup$ Stop);
***w**hile (lookahead $\in$ FIRST(E)) **d**o*
A(E, FIRST(E) $\cup$ Stop)

Igual que antes, si se tiene constancia de que $lookahead \in (First(E) \cup Stop)$ es cierto al comienzo del algoritmo entonces la llamada inicial a `syntax_check()` *puede omitirse.*

Regla 3.2.7 *Si todas las cadenas de la forma $S_1, S_2, \ldots, S_n$ son distintas de ϵ el siguiente algoritmo reconocerá una producción de la forma $S_1|S_2|\ldots|S_n$ seguida por un símbolo de parada:*
$A(S_1|S_2|\ldots|S_n, Stop)$ =
***i**f lookahead $\in$ FIRST(S_1) **t**hen A(S_1, Stop)*

***e**lse if lookahead ∈ FIRST(S_2) **t**hen A(S_2, Stop)*
...
***e**lse if lookahead ∈FIRST(S_n) **t**hen A(S_n, Stop)*
***e**lse Syntax_error();*

Si cualquiera de las alternativas puede derivar en ϵ el mensaje de error del final se debe reemplazar por una llamada a `syntax_check(Stop)`*.*

Regla 3.2.8 *El siguiente algoritmo reconoce una o más sentencias de la forma S seguidas por un símbolo de parada:*
A(S{S}, Stop) =
A(S, FIRST(S) ∪ Stop);
***w**hile (lookahead ∈ FIRST(S)) **d**o*
A(S, FIRST(S) ∪ Stop)

Regla 3.2.9 *El siguiente algoritmo reconoce una o más sentencias de la forma S separadas por el símbolo t y seguidas por un símbolo de parada:*
A(S{tS}, Stop) =
A(S, [t] ∪ Stop);
***w**hile (lookahead = t) **d**o*
***b**egin*
match(t, FIRST(S) ∪ [t] ∪ Stop);
A(S, [t] ∪ Stop)
***e**nd;*

Se ha de modificar el procedimiento `error()` del analizador léxico de modo que sólo se notifique un único error por línea del código fuente.

El programa principal después de inicializar las variables, lee el primer token e invoca a la función `program()` (asociada al símbolo de arranque de la gramática) con un conjunto de parada compuesto únicamente por el token `ENDTEXT`.

Tanto si el programa analizado tiene errores sintácticos como si no los tiene, el analizador sintáctico deberá indicar que ha finalizado de forma normal.

El alumno debe diseñar también un programa `test_err.p` de prueba que contenga todas las posibles situaciones de error que se le ocurran de acuerdo con la gramática de *P*ascal-, y comprobar la correcta recuperación de los errores por parte del analizador sintáctico. Contrastar el comportamiento de su analizador con el del compilador de *P*ascal- que se entregó con la primera práctica.

3.3. Notas

Se valorará positivamente que el alumno haya diseñado su práctica de forma estructurada utilizando un fichero makefile para la compilación de los diferentes módulos.