Proyecto Final Procesadores

Por: Daniel Giraldo Valencia, Santiago Jiménez Cotrini, Miguel Suárez Obando

Análisis Léxico y Sintáctico

- **Componentes léxicos principales de OCaml**
    - Palabras Reservadas
        - let
        - in
        - rec
        - if
        - then
        - else
        - match
        - with
        - fun
        - function
        - while
        - do
        - done
        - for
        - to
        - downto
        - true
        - false
        - type
        - of
        - and
    - Operadores
        - Aritméticos
            - +
            - -
            - *
            - /
            - +.
            - -.
            - *.
            - /.
        - Comparadores
            - =
            - <>

            - <
            - >
            - <=
            - >=
        - Booleanos
            - &&
            - ||
            - not
        - Listas y cons
            - ::
            - @
        - Otros
            - |
            - ->
            - :=
    - Delimitadores
        - Paréntesis - ( )
        - Corchetes - [ ], [| |]
        - Llaves - { }
        - Separadores - ; , ;; :
        - Comentarios - (* ... *)
    - Tipos de datos básicos
        - int
        - float
        - bool
        - char
        - string
        - unit
    - Estructuras de control
        - Condicional – if x then y else z
        - Bucle While – while x do y done
        - Bucle For – for i = 0 x to y do z done
        - Patrón – match e with | p1 -> e1 | p2 -> e2

Nuestra proposición de un nuevo lenguaje: Firme (FRM)

- **Idea General**
    - Tener una sintaxis más cercana al español manteniendo la lógica y semántica de OCaml.
        - Declaraciones se traducen a let
        - Condicionales se traducen a if … then … else

        - Ciclos se traducen a while o for
        - Llamadas a funciones se traducen a llamadas normales de OCaml
    - El plan es que el intérprete lea un .txt con código en FRM y lo ejecute traduciéndose internamente a OCaml.
- **Tabla Léxica de tokens y patrones para FRM**
    -

Token	Patrón	Significado
KW_DECLARAR	declarar\b	Introduce una declaración de variable
KW_FUNCTION	funcion\b	Declaración de función
KW_SI	si\b	Inicio de condicional
KW_ENTONCES	entonces\b	Parte “then”
KW_SINO	sino\b	Parte “else”
KW_MIENTRAS	mientras\b	Bucle while
KW_PARA	para\b	Bucle for
KW_DESDE	desde\b	Límite inferior del for
KW_HASTA	hasta\b	Límite superior del for
KW_HACER	hacer\b	Cuerpo de ciclo
KW_FIN	fin\b	Fin de bloque
KW_IMPRIMIR	imprimir\b	Imprimir en pantalla
KW_RETORNAR	retornar\b	Retorno de función
KW_ENTERO	entero\b	Anotación de tipo entero
KW_REAL	real\b	Anotación de tipo real
KW_LOGICO	logico\b	Anotación de tipo booleano
KW_VERDADERO	verdadero\b	Constante booleana true
KW_FALSO	falso\b	Constante booleana false
OP_ASIG	<- O ->	Asignación / Inicialización

Token	Patrón	Significado
OP_MAS	\+	Suma
OP_MENOS	-	Resta
OP_POR	*	Multiplicación
OP_DIV	/	División
OP_MAYOR	>	Mayor que
OP_MENOR	<	Menor que
OP_MAYORIG	>=	Mayor o igual
OP_MENORIG	<=	Menor o igual
OP_IGUAL	==	Igualdad
OP_DISTINTO	!=	Distinto
OP_Y	y\b	Conjunción lógica (&&)
OP_O	o\b	Disyunción lógica (||)
DOS_PUNTOS	:	Separador
PAREN_IZQ	\(	Paréntesis izquierdo
PAREN_DER	\)	Paréntesis derecho
LLAVE_IZQ	{	Llave izquierda
LLAVE_DER	}	Llave derecha
COMA	,	Coma
PUNTOYCOMA	;	Fin de sentencia opcional
STRING	"[^"\\]*"	Cadena de caracteres
INT	[0-9]+	Entero
FLOAT	[0-9]+\.[0-9]+	Real
IDENT	[a-zA-Z_][a-zA-Z0-9_]*	Identificador
ESPACIO	[\t\r]+	Espacios (ignorados)
NUEVA_LINEA	\n	Salto de línea

Token	Patrón	Significado
COMENTARIO	#.*	Comentario hasta fin de línea

- **Palabras Reservadas y sus homólogos en OCaml**
  -

En FRM	En OCaml	Comentario
declarar	let	Nueva variable / binding
funcion	let / let rec	Definicion de funcion
si	if	Condicional
entonces	then	Parte then
sino	else	Parte else
mientras	while	Bucle While
para	for	Bucle for
desde	= dentro del for	Límite inferior del for
hasta	to / downto	Límite superior del for
hacer	do	Inicio del cuerpo de un ciclo
fin	done o cierre de bloque	Fin de ciclo / función
imprimir	print_endline / print_int	Salida por consola
retornar	return simulado: última expresión	En OCaml se usa la última expresión
verdadero	true	Booleano true
falso	false	Booleano false
entero	int	Tipo entero
real	float	Tipo real
logico	bool	Tipo booleano

- **Operadores y símbolos (semántica) en FRM**
  - Aritméticos
    - +

        - -
        - *
        - /
        - +.
        - -.
        - *.
        - /.
    - Comparación
        - == → =
        - != → <>
        - <
        - >
        - <=
        - >=
    - Booleanos
        - y → &&
        - o → ||
    - Asignación
        - ← (OP_ASIG) → let x = expr
    - Delimitadores / bloques
        - : se usa tras condiciones
        - fin marca cierre de bloques (if, while, for, funcion)
- **Estructuras básicas del lenguaje**
    - Especificación de una especie de EBNF simplificada
        - ***Programa (lista de sentencias una debajo de la otra)***
            - $programa ::= \{sentencia\}$
        - ***Sentencias***
            -
```
sentencia ::=
    decl_var
  | decl_funcion
  | asignacion
  | sentencia_if
  | sentencia_mientras
  | sentencia_para
  | llamada_imprimir
  | expresion  (opcionalmente terminada en ';')
```
        - ***Declaraciones de variables***
            - $declarar\ IDENT$ [": " $tipo$]" ← $expresion$
                - Traducción a OCaml seria:
                    - $let\ IDENT\ = expr$

        - Con tipo – $let\ (IDENT\colon tipoocaml\ =\ expr$
    - Por ejemplo – $declarar\ x\colon\ entero \leftarrow 20$
        - En OCaml – $let\ (x\colon int)\ =\ 20$
- ***Declaración de función***
    - ```
      funcion IDENT "(" [ lista_param ] ")" [ ":" tipo ] "→"
          bloque
      fin
      ```
    - ```
      lista_param ::= param { "," param }
      param       ::= IDENT [ ":" tipo ]
      bloque      ::= { sentencia }
      ```
    - En OCaml sería:
        - ```
          let IDENT param1 param2 ... =
            (* bloque traducido, última expresión es el retorno *)
          ```
    - Ejemplo:
        - ```
          funcion suma(a : entero, b : entero) : entero →
              retornar a + b
          fin
          ```
        - En OCaml:
            - ```
              let suma (a : int) (b : int) : int =
                a + b
              ```
- ***Condicionales***
    - ```
      si expresion_logica ":"
          bloque_then
      [sino ":"
          bloque_else]
      fin
      ```
    - En OCaml sería:
        - ```
          if expr_logica then
            (* bloque_then *)
          else
            (* bloque_else *)
          ```
- ***Bucle While (Mientras)***
          ```

```
mientras expresion_logica ":"
    bloque
fin
```

- En OCaml sería:

```
while expr_logica do
  (* bloque *)
done
```

- ***Bucle For (Para)***

```
para IDENT desde expresion hasta expresion ":"
    bloque
fin
```

- En OCaml sería:

```
for IDENT = expr1 to expr2 do
  (* bloque *)
done
```

- ***Impresión***

```
imprimir("(" expresion ")")
```

- En OCaml sería:

```
print_endline (string_of_expr)
```

- **Programa de ejemplo completo en FRM**

```
declarar x : entero ← 20

si x > 5:
    imprimir("Mayor que 5")
sino:
    imprimir("Menor o igual")
fin
```

- En OCaml sería:

- ```
  let x = 20 in
  if x > 5 then
    print_endline "Mayor que 5"
  else
    print_endline "Menor o igual"
  ```
  ```