PyTiger2C
Anotaciones sobre la comprobación sintáctica

Yasser González Fernández
yglez@uh.cu

Ariel Hernández Amador
gnuaha7@uh.cu

1. Análisis lexicográfico

Durante el análisis lexicográfico se separa el flujo de caracteres de entrada en un conjunto de *tokens*. A continuación se enumeran las expresiones regulares utilizadas para reconocer cada *token* del lenguaje Tiger.

1. `ARRAY : array`
2. `IF : if`
3. `THEN : then`
4. `ELSE : else`
5. `WHILE : while`
6. `FOR : for`
7. `TO : to`
8. `DO : do`
9. `LET : let`
10. `IN : in`
11. `END : end`
12. `OF : of`
13. `BREAK : break`
14. `NIL : nil`
15. `FUNCTION : function`
16. `VAR : var`
17. `TYPE : type`
18. `ID : [a-zA-Z][a-zA-Z0-9_]*`
19. `STRLIT :`

```
"(
    (\\[nt])
  | (\\")
  | (\\\\)
  | (\\\^[@A-Z[\]^_])
  | (\\[0-9][0-9][0-9])
  | (\\\s+\\)
  | ([^\\"])
)*"
```

20. `INTLIT : [0-9]+`
21. `PLUS : \+`
22. `MINUS : \-`

23. `TIMES : \*`
24. `DIVIDE : /`
25. `EQ : =`
26. `NE : <>`
27. `LT : <`
28. `LE : <=`
29. `GT : >`
30. `GE : >=`
31. `AND : &`
32. `OR : \|`
33. `ASSIGN : :=`
34. `PERIOD : \.`
35. `COMMA : ,`
36. `COLON : :`
37. `SEMICOLON : ;`
38. `LPAREN : \(`
39. `RPAREN : \)`
40. `LBRACKET : \[`
41. `RBRACKET : \]`
42. `LBRACE : \{`
43. `RBRACE : \}`

2. Análisis sintáctico

El análisis sintáctico reconoce estructuras del lenguaje a partir del flujo de *tokens* obtenidos del análisis lexicográfico. A continuación se enumeran las reglas de la gramática libre del contexto utilizada por PyTiger2C para reconocer las estructuras del lenguaje Tiger.

1. `program ::= expr`
2. `expr ::= NIL`
3. `expr ::= INTLIT`
4. `expr ::= STRLIT`
5. `expr ::= lvalue`
6. `expr ::= ID LBRACKET expr RBRACKET OF expr`

7. expr ::= ID LBRACE field_list RBRACE
8. expr ::= MINUS expr
9. expr ::= expr PLUS expr
10. expr ::= expr MINUS expr
11. expr ::= expr TIMES expr
12. expr ::= expr DIVIDE expr
13. expr ::= expr EQ expr
14. expr ::= expr NE expr
15. expr ::= expr LT expr
16. expr ::= expr LE expr
17. expr ::= expr GT expr
18. expr ::= expr GE expr
19. expr ::= expr AND expr
20. expr ::= expr OR expr
21. expr ::= LPAREN expr_seq RPAREN
22. expr ::= lvalue ASSIGN expr
23. expr ::= ID LPAREN expr_list RPAREN
24. expr ::= IF expr THEN expr
25. expr ::= IF expr THEN expr ELSE expr
26. expr ::= WHILE expr DO expr
27. expr ::= FOR ID ASSIGN expr TO expr DO expr
28. expr ::= BREAK
29. expr ::= LET dec_group IN expr_seq END
30. lvalue ::= ID
31. lvalue ::= lvalue PERIOD ID
32. lvalue ::= ID LBRACKET expr RBRACKET
33. lvalue ::= lvalue LBRACKET expr RBRACKET
34. expr_seq ::=
35. expr_seq ::= expr_seq SEMICOLON expr
36. expr_seq ::= expr
37. dec_group ::=
38. dec_group ::= dec_group dec

39. field_list ::=
40. field_list ::= field_assign
41. field_list ::= field_list COMMA field_assign
42. field_assign ::= ID EQ expr
43. expr_list ::=
44. expr_list ::= expr_list COMMA expr
45. expr_list ::= expr
46. dec ::= type_dec_group
47. dec ::= var_dec
48. dec ::= func_dec_group
49. func_dec_group ::= func_dec
50. func_dec_group ::= func_dec_group func_dec
51. type_dec_group ::= type_dec
52. type_dec_group ::= type_dec_group type_dec
53. type_dec ::= TYPE ID EQ type
54. type ::= ID
55. type ::= LBRACE field_types RBRACE
56. type ::= ARRAY OF ID
57. field_types ::=
58. field_types ::= field_type
59. field_types ::= field_types COMMA field_type
60. field_type ::= ID COLON ID
61. var_dec ::= VAR ID ASSIGN expr
62. var_dec ::= VAR ID COLON ID ASSIGN expr
63. func_dec ::= FUNCTION ID LPAREN field_types RPAREN EQ expr
64. func_dec ::= FUNCTION ID LPAREN field_types RPAREN COLON ID EQ expr

2.1. Reglas de precedencia

La gramática anterior presenta algunas ambigüedades que se reportan en forma de conflictos *SHIFT-REDUCE* al hacer el análisis LR. En algunos casos, es posible hacer modificaciones a la gramática para evitar estos conflictos pero decidimos no hacer dichas modificaciones y en su lugar utilizar reglas de precedencia para evitar perder claridad en las estructuras del lenguaje Tiger que reconocen cada una de las producciones.

Las reglas de precedencia indican que acción se debe tomar ante un conflicto *SHIFT-REDUCE*, indicando si se debe introducir en la pila el *token* de la cadena de entrada o reducir la producción del conflicto.

Utilizando PLY, las reglas de precedencia se especifican asignando valores de prioridad a los *tokens* de la gramática mediante una lista. Un *token* en la lista tendrá mayor prioridad que todos los *tokens* que aparezcan anteriormente. Se seleccionará hacer *REDUCE* a una regla o hacer *SHIFT* a un *token* en favor del que tenga mayor prioridad. La prioridad de una regla está dada por la prioridad del último *token* presente en la regla.

A continuación se muestran las reglas de precedencia utilizadas para la gramática anterior.

```
precedence = (
    # The following fixes the shift/reduce conflict caused by rules ending with
    # the non-terminal expr and the rules for binary operators.
    ('nonassoc', 'OF', 'THEN', 'DO'),
    # The token ELSE has higher priority to fix the dangling-else shift/reduce
    # conflict. If an ELSE if found it should be shifted instead of reducing the
    # if-then without the else clause.
    ('nonassoc', 'ELSE'),
    # The following fixes the shift/reduce conflict between shifting the
    # token in the p_expr_array production (array literal) or reducing
    # the p_lvalue_id production.
    ('nonassoc', 'LVALUE_ID'),
    ('nonassoc', 'LBRACKET'),
    # Operator precedence.
    ('nonassoc', 'ASSIGN'),
    ('left', 'OR'),
    ('left', 'AND'),
    ('nonassoc', 'EQ', 'NE', 'LT', 'LE', 'GT', 'GE'),
    ('left', 'PLUS', 'MINUS'),
    ('left', 'TIMES', 'DIVIDE'),
    ('right', 'UMINUS'),
)
```

Un grupo de conflictos *SHIFT-REDUCE* es producido por todas las reglas de la gramática que terminen en el no terminal `expr` y las reglas utilizadas para los operadores binarios. Se utilizan reglas de precedencia para indicar que en estos casos se debe introducir el *token* correspondiente al operador en la pila en lugar de reducir las reglas que terminan en `expr`.

Además, se produce un conflicto *SHIFT-REDUCE* al reconocer las expresiones `if-then` y `if-then-else`. Este problema es conocido en la literatura como *dangling else* y se puede solucionar introduciendo el *token else* en la pila, indicando que este *else* corresponde a la estructura *if-then* anterior.

Otro conflicto *SHIFT-REDUCE* se produce con la producción que reconoce un identificador como un *lvalue* y el *token* `LBRACKET` en la declaración de un literal de *array*. En este caso se indica que se introduzca el token `LBRACKET` en la pila para tratar de reconocer la declaración de un literal de *array*.

El resto de las reglas de precedencia se utilizan para garantizar la precedencia y asociatividad entre los operadores.