CS445/CS545 Compiler Construction

The Dragon Project

Context-Free Grammar

Here is a context-free grammar for a programming language that is given in Appendix A in Dragon 1e.

```
program:
        program ident ‘(‘ identifier_list ‘)’ ‘;’
        declarations
        subprogram_declarations
        compound_statement
        ‘.’

identifier_list: ident
        | identifier_list ‘,’ ident

declarations: declarations var identifier_list ‘:’ type ‘;’
        |

type: standard_type
        | array ‘[‘ number ‘..’  number ‘]’ of standard_type

standard_type: integer
        | real

subprogram_declarations: subprogram_declarations subprogram_declaration ‘;’
        |

subprogram_declaration:
        subprogram_head
        declarations
        compound_statement

subprogram_head: function ident arguments ‘:’ standard_type ‘;’
        | procedure ident arguments ‘;’

arguments: ‘(‘ parameter_list ‘)’
        |

parameter_list: identifier_list ‘:’ type
        | parameter_list ‘;’ identifier_list ‘:’ type
```

compound_statement: **begin** optional_statements **end**

optional_statements: statement_list
|

statement_list: statement
| statement_list ‘;’ statement

statement: variable **assignop** expression
| procedure_statement
| compound_statement
| **if** expression **then** statement **else** statement
| **while** expression **do** statement

variable: **ident**
| **ident** ‘[‘ expression ‘]’

procedure_statement: **ident**
| **ident** ‘(‘ expression_list ‘)’

expression_list: expression
| expression_list ‘,’ expression

expression: simple_expression
| simple_expression **relop** simple_expression

simple_expression: term
| sign term
| simple_expression **addop** term

term: factor
| term **mulop** factor

factor: **ident**
| **ident** ‘(‘ expression_list ‘)’
| **number**
| ‘(‘ expression ‘)’
| **not** factor

sign: ‘+’ | ‘-’

Lexical Constructs

Here are the definitions of all the tokens valid in the language.

1. Comments are surrounded by either a pair of ‘{‘ and ‘}’ or by a pair of “(*” and “*)”
Comments may span several lines.

2. Whitespaces are defined to be spaces, tabs and newlines. Whitespaces between tokens are optional. But keywords must be surrounded by whitespaces.

3. Token **ident** for user-defined idenfitiers are defined as a letter followed by a sequence of letters or digits. There is no limit on the length of an identifier.

ident ::= **letter** (**letter** | **digit**)*
letter ::= [a-zA-Z]
digit ::= [0-9]

4. Keywords are reserved (may not be used for anything else). These are:

program	*starts the main program*
begin	*starts a new block*
end	*ends a block*
var	*starts a list of identifier names*
array	*signals an array type*
of	*used in array type declaration*
integer	*basic integer type*
real	*basic real (float) type*
function	*starts a function (returns values,have no side-effects) declaration*
procedure	*starts a procedure (returns no values, may have side-effects) declaration*
if	*starts an IF statement*
then	*part of an IF statement*
else	*starts the ELSE part of an IF-THEN-ELSE statement*
while	*starts a WHILE statement*
do	*the DO part of a WHILE-DO statement*
not	*logical NOT (negation)*

5. The relational operator **relop** include:

=	*equal*
<>	*not-equal*
<	*less-than*
<=	*less-or-equal*
>	*greater-than*
>=	*greater-or-equal*

6. The additive operator **addop** include:

+	*addition for both integer and real arguments*
-	addition for both integer and real arguments
or	*logical OR*

7. The multiplicative operator **mulop** include:

*	*multiplication for both integer and real arguments*
/	*division for both integer and real arguments*
div	*quotient in integer division*
mod	*remainder in integer division*
and	*logical AND*

8. The *lexeme* for the assignment operator **assignop** is

:=

9. The token **number** matches unsigned integers.

number ::= **digits** | **digits** '.' **digits**
digits ::= **digit digit***

Add-Ons

1. Allow nesting of subprograms within each other
2. Adjust the "sign" problem
3. Allow array access to appear on right-hand side of an assignment statement
4. Allow IF-THEN statement without the ELSE option
5. (optional) Add a FOR-DO statement

```
for index := 1 to size do
    result := result + index
```

Semantic Checks

Scoping

1.1. A local name may not be used more than once within the same scope.
1.2. A local name may potentially hide a non-local name.
1.3. A non-local name is visible from an inner scope (unless a local name with the same name is defined in the inner scope).
1.4. A subprogram name (function or procedure) exists in the scope they are defined (not in their own scope).
1.5. Local object no longer exists once their scope ceases to exist.

Expressions

2.1. An expression returns a typed value.
2.2. A name must be declared before it is used in an expression.
2.3. Names of different types may not appear in the same expression (strong typing).

Statements

3.1. A statement does not return any value.
3.2. The type of an expression used in an IF or WHILE statement is Boolean (although there is no explicit Boolean type).
3.3. The ELSE clause always binds to the closest enclosing IF statement.

Arrays

4.1. The type of the indexing expression must be integer.
4.2. In the declaration of an array, the lower range is smaller or equal to the upper range.

Functions

5.1. A function return type must be either integer or real (only basic types).
5.2. A function must contain a return statement within its body (the return statement is of the form <function name> := <expression>).
5.3. The sequence of expressions used in a function call must match (in number and in types) the sequence of arguments defined in the header of that function.
5.4. A function may not update the value of a non-local object (no side effects).

Procedures

6.1. A procedure call does not return a value.
6.2. The sequence of expressions used in a procedure call must match (in number and in types) the sequence of arguments defined in the header of that procedure.

Sample programs

- Computing the Greatest Common Divisor of two integers:

```
(* This is a Pascal program for Euclid’s GCD algorithm *)
program main( input, output );
    var a, b, c: integer;
    function gcd( a,b : integer ) : integer;
        var r: integer;
    begin
        if a < b then
            gcd := gcd( b, a )
        else if b = 0 then
            gcd := a
        else
            begin
                r := a mod b;
                gcd := gcd(b, r)
            end
    end;
begin
    read( a );
    read( b );
    c := gcd( a, b );
    writeln( c )
end.
```

```
program main(input, output);
    var a,b: integer;
    var x,y: real;
    var p: array[5 .. 13] of integer;
    var q: array[1..10] of real;
begin
    read(a);
```

```
          read(b);
          p[p[b]] := p[p[p[b+a]]]
end.
```