

コンパイラレポート 構文解析

7JFC1121 佐藤圭一

1 . 目的

コンパイラの動作原理の中の一部に、「解析」がある。与えられた命令を低級言語に変換すべく構文の内容を解析する事は、コンパイラそのものと言っていいほど、コンパイラの中では重要な位置を占めている。

今回は、教科書にて定義されている「PASCAL-T」の構文解析の仕組みを解きほどいてみる。

2 . 概要

PASCAL-T 内の「expression」手続きを今回は解いていくのだが、expression 手続きがどのような動きをしているのか、簡単に説明する。

expression 手続きは、前出の構文判断手続きによりソースコード内に「数式」が現れた場合、数式を再帰下降解析して数式を演算する手続きである。

expression 手続きは、内部に 4 つの孫手続きを持っている。

以下、各手続きの概要を説明する。

手続き通し番号 27 : expression(構文解析)

内容 : 式における構文解析とコード生成を行う。

手続き通し番号 28 : call(低級言語呼出処理定義)

内容 : 手続きを呼び出す処理を低級言語に置きかえる。

また、引数の型を調べ、矛盾が生じた場合はエラーを返す。

手続き通し番号 29 : simpleexpression(加減式構文解析)

内容 : 加減式演算のコード生成を行う。

主に加算、減算、論理和を演算、その他は term 手続きに渡す。

手続き通し番号 30 : term(その他式構文解析)

内容 : simpleexpression で処理できない演算のコード生成を行う。

式中の乗算、除算、論理積を演算する。

また、括弧、論理否が与えられた場合、手続き factor を呼び出す。

手続き通し番号 31 : factor(式区切り構文解析)

内容 : 式中の括弧、論理否など、式の区切りとなる物についてのコード生成を行う。

3 . 動作内容

各手続きの動作内容を説明する。

- expression 手続き
 - insymbol 手続きを使い、式要素を分解する。
 - simpleexpression を実行。(ここで構文は解析される)
 - 必要であれば別のスタックにデータを逃がし、式を戻す。
 - 式がなくなるまで simpleexpression を再帰呼び出しする。
 - スタックに矛盾を生じさせない為の低級言語をためこむ。
 - 以上の処理で何らかのエラーが発生した場合、エラーを返す。
- call 手続き
 - 渡されたテーブルの内容を適当な低級言語に変換する。
 - エラーが起きた場合、エラーを返す。
- simpleexpression
 - 式を解析し、適宣 term 手続きを呼び出す。
 - 加減式の場合、内部で低級言語化する。
- term
 - 式を解析し、適宣 factor 手続きを呼び出す。
 - 乗除式の場合、内部で低級言語化する。
- factor
 - 式の順位を制御する。
 - 構文に矛盾が生じた場合、エラーを出す。

4 . 構文解析の手法

Pascal-T では、「再帰下降解析」という解析方法を採用している。

再帰下降解析とは、下向き構文解析法であり、解析ルーチンが構文規則ときれいに対応している解析方法である。

解析方法は、木を使い、式の並べ替え、最適化を実行、式の頭からお尻に向けてきれいに演算順序をまとめていく方式である。

演算順序をまとめるに当たり、括弧や演算式の順位も考慮にいれる必要がある。よって、一度定義した式を遡り、演算式の埋め込みなおしを行う必要が出てくる。これをバックトラックという。

単純に「戻る」「進む」だけで論理演算を行うため、構文自体が無限ループに陥ってしまう場合がある。これは式内容の並び替えを行うことで回避できる。

5 . 考察

一般に構文解析は(文法の)木構造のルートから解析するトップダウンパーサ(top down parser) と、リーフから解析する ボトムアップ パーサ(bottom up parser) の 2 種類がある。PASCAL-T では、「トップダウンパーサ」を行い、一部繰り上がることで構文解析を実現しているのだと思う。が、枝から見て行った場合、無限ループなどの対策を施さなくてもなんとかなってしまうのは気のせいであろうか？

無論、ボトムアップの問題点も多々有ると思うが。

6 . 該当部分のソースコード

```

procedure expression;
  var op      : symbolkind;
      y      : attribute;
      mop1, mop2 : mopt;
(*-----*)
(* 29 *)

procedure simpleexpression(var x : attribute; fsys : symbolset);
  var op      : symbolkind;
      y      : attribute;
      mop1, mop2 : mopt;
(*-----*)
(* 30 *)

procedure term( var x : attribute ; fsys : symbolset );
  var
    op      : symbolkind ;
    y      : attribute ;
    mop1 , mop2 : mopt ;
(*-----*)
(* 31 *)

procedure factor(var x : attribute; fsys : symbolset);
  var i : 0 .. tmax;
      mop1 : mopt;
begin
  x.attype := notype;
  if not (sy in facbegsys) then
    begin
      error(14);
      skip(fsys + facbegsys)
    end;
  while sy in facbegsys do
    begin case sy of
      intliteral:
        begin with x do
          begin
            attype := intt;
            atk := cons;
            atvalue := inum
          end;
          insymbol
        end;
      identifier:
        begin
          i := loc(id);
          insymbol;
          with idtable[i] do case object1 of
            oconstant: with x do
              begin
                attype := idtype;
                atk := cons;
                atvalue := idvalue
              end;
            ovariable: with x do
              begin
                attype := idtype;
                atk := vari;
                atlevel := idlevel;
                ataddr := idaddr
              end;
            otype: error(20);
            ofunction:
              begin
                call(true, i, fsys);
                with mop1 do
                  begin
                    mopk := index;
                    rind := 6;
                    xind := -12
                  end;
                  emit2('MOV', mop1, r0);
                  with x do
                    begin
                      attype := idtype;
                      atk := expr
                    end;
                    ntemp := ntemp + 1
                  end;
                oprocedure : error(20)
              end
            end;
          end;
          synot:
            begin
              insymbol;
              factor(x, fsys);
              if x.attype <> notype then

```

コンパイラレポート - 構文解析

```

if x.attype = boolt then
  begin
    if x.atk <> expr then
      begin
        push;
        attomop(x, 2, mop1);
        emit2('MOV', mop1, r0)
      end;
      emit2('MOV', one, r1);
      emit2('XOR', r1, r0);
      x.atk := expr
    end
  else begin
    error(23);
    x.attype := notype
  end
end;
symparen:
  begin
    insymbol;
    expression(x, fsys + [symparen]);
    if sy = symparen
      then insymbol
      else error(3)
    end
  end;
if not (sy in fsys) then
  begin
    error(15);
    skip(fsys + facbegsys)
  end
end
end (* factor *);

(*****)

begin (* term *)
  factor ( x, fsys + [ symul, sydiv , syand ] );
  while sy in [ symul ,sydiv ,syand ] do
    begin
      op := sy ;
      insymbol ;
      factor( y,fsys + [ symul, sydiv, syand ] );
      if ( x.attype <> notype )
        and ( y.attype <> notype ) then
        begin
          attomop( x, 2, mop1 );
          attomop( y, 3, mop2 );
          case op of
            symul :
              if ( x.attype = intt )
                and ( y.attype = intt ) then
                begin
                  if x.atk <> expr then
                    if y.atk <> expr then
                      begin push ;
                        emit2( 'MOV', mop1, r0 ) ;
                        emit2( 'MUL', mop2, r0 )
                      end
                    else emit2( 'MUL', mop1, r0)
                      else if y.atk <> expr then
                        emit2( 'MUL', mop2, r0 ) else
                          begin
                            emit2( 'MUL',stackinc, r0 ) ;
                            pop
                          end ;
                        emit2( 'MOV', r1, r0 )
                      end else
                        begin
                          error(24);
                          x.attype := notype
                        end ;
                    sydiv :
                      if ( x.attype = intt )
                        and ( y.attype = intt ) then
                        if x.atk <> expr then
                          if y.atk <> expr then
                            begin push ;
                              emit2( 'MOV', mop1, r1 ) ;
                              emit1( 'CLR', r0 ) ;
                              emit2( 'DIV', mop2, r0 )
                            end else
                              begin
                                emit2( 'MOV', r0, r3 ) ;
                                emit2( 'MOV', mop1, r1 ) ;
                                emit1( 'CLR', r0 ) ;
                                emit2( 'DIV', r3, r0 )
                              end
                            end
                          else if y.atk <> expr then
                            begin
                              emit2( 'MOV', r0, r1 ) ;
                              emit1( 'CLR', r0 ) ;
                              emit2( 'DIV', mop2, r0 )
                            end
                        end
                    end
                end
            end
          end
        end
      else
        error(24);
        x.attype := notype
      end ;
    end ;
  end ;
end ;

```


コンパイラレポート - 構文解析

```

        end else
        begin
            emit2( 'MOV', r0, r3 );
            emit2( 'MOV', stackinc, r1 );
            emit1( 'CLR', r0 );
            emit2( 'DIV', r3, r0 );
            pop
        end
    else
    begin
        error(24);
        x.attype := notype
    end;
syand :
    if ( x.attype = boolt )
and ( y.attype = boolt )then
    begin
        if x.atk <> expr then
            if y.atk <> expr then
                begin push ;
                    emit2( 'MOV', mop1, r0 );
                    emit2( 'MOV', mop2, r1 );
                end
            else emit2( 'MOV', mop1, r1 )
            else if y.atk <> expr then
                emit2( 'MOV', mop2, r1 )
            else begin
                    emit2( 'MOV', stackinc, r1 );
                    pop
                end ;
                emit2( 'MOV', one, r2 );
                emit2( 'XOR', r2, r1 );
                emit2( 'BIC', r1, r0 )
            end else
            begin
                error(23);
                x.attype := notype
            end
        end (* case *) ; x.atk := expr
    end else x.attype := notype
    end
end (* term *) ;
(.....)
begin (*simpleexpression*)
if sy in [syplus, syminus] then
begin
op := sy;
insymbol;
term(x,fsys + [syplus,syminus]);
if x.attype <> notype then
if x.attype = intt then
begin
if op = syminus then
begin
attomop(x, 2, mop1);
if x.atk <> expr then
begin
push;
emit2('MOV',mop1, r0);
emit1('NEG',r0); x.atk := expr
end
else emit1('NEG',r0)
end
end
else begin
error(24);
x.attype := notype
end
end
else term(x, fsys + [syplus, syminus, syor]);
while sy in [syplus, syminus, syor] do
begin
op := sy;
insymbol;
term(y, fsys + [syplus, syminus, syor]);
if (x.attype <> notype) and (y.attype <> notype) then
begin
attomop (x, 2, mop1); attomop(y, 3, mop2);
case op of
syplus:
if (x.attype = intt) and (y.attype = intt) then
if x.atk <> expr then
if y.atk <> expr then
begin
push;
emit2('MOV', mop1, r0);
emit2('ADD', mop2, r0)
end
else emit2('ADD', mop1, r0)
else
if y.atk <> expr then emit2('ADD', mop2, r0)

```

コンパイラレポート - 構文解析

```

        else begin
            emit2('ADD', stackinc, r0);
            pop
            end
        else
            begin
            error(24);
            x.attype := notype
            end;
        syminus:
        if (x.attype = intt) and (y.attype = intt) then
            if x.atk <> expr then
                if y.atk <> expr then
                    begin push;
                        emit2('MOV', mop1, r0);
                        emit2('SUB', mop2, r0)
                    end
                else begin
                    emit2('MOV', r0, r1);
                    emit2('MOV', mop1, r0);
                    emit2('SUB', r1, r0)
                    end
                else if y.atk <> expr then
                    emit2('SUB', mop2, r0) else
                    begin
                        emit2('MOV', r0, r1);
                        emit2('MOV', stackinc, r0);
                        emit2('SUB', r1, r0);
                        pop
                    end
                else
                    begin
                    error(24);
                    x.attype := notype
                    end;
            end;
        syor:
        if (x.attype = boolt) and (y.attype = boolt) then
            if x.atk <> expr then
                if y.atk <> expr then
                    begin
                    push;
                    emit2('MOV', mop1, r0);
                    emit2('BIS', mop2, r0)
                    end
                else emit2('BIS', mop1, r0)
                else if y.atk <> expr then
                    emit2('BIS', mop2, r0)
                else begin
                    emit2('BIS', stackinc, r0);
                    pop
                end
            else begin
                error(23);
                x.attype := notype
            end
        end(*case *);
        x.atk := expr
        end
    else x.attype := notype
    end
end (* simpleexpression *);

begin(*expression*)
    simpleexpression
    (x, fsys+[sylv, syle, syne, syeq, sygt, syge]);
    if sy in [sylv, syle, syne, syeq, sygt, syge] then
        begin op := sy;
        insymbol;
        simpleexpression(y, fsys);
        if (x.attype <> notype) and (y.attype <> notype) then
            if (x.attype = intt) and (y.attype = intt) then
                begin
                    emit2('MOV', one, r1);
                    attomop(x, 2, mop1); attomop(y, 3, mop2);
                    if x.atk <> expr then
                        if y.atk <> expr then
                            begin push; emit2('CMP', mop1, mop2) end
                        else emit2('CMP', mop1, r0)
                    else if y.atk <> expr then
                        emit2('CMP', r0, mop2)
                    else
                        begin
                            emit2('CMP', stackinc, r0); pop end;
                    case op of
                        sylv: emitbra('BLT'); syle: emitbra('BLE');
                        syne: emitbra('BNE'); syeq: emitbra('BEQ');
                        sygt: emitbra('BGT'); syge: emitbra('BGE')
                    end;
                    newilabel: code[c].bralab := mlcount;
                    emit1('CLR', r1);
                    code[c+1].ilabel := mlcount;
                    emit2('MOV', r1, r0);
                end
            end
        end
    end
end

```

コンパイラレポート - 構文解析

```
        x.attype := boolt; x.atk := expr
      end
    else begin error(25); x.attype := notype end
  else x.attype := notype
end
end(*expression*);
```