The parsing procedure

```
procedure parse (goal: hpointer;  var match: boolean);
    var s : pointer;
begin s : = goal ^. entry ;
    repeat
        if s^ . terminal then
                begin if s^ .tsym = sym then
                        begin match : = true; getsym
                        end
                    else match : = (s^ .tsym = empty)
                end
        else parse(s^ .nsym, match);
        if match then s : = s^ .suc else s : = s^ .alt
    until s = nil
end
```

A recognizer for the language

```
program parser(input, output);
label 99;
const emp y =' *' ;
var sym: char;

procedure getsym;
    begin
    repeat read(sym); write(sym) until sym  ≠ ' ' ,
end {getsym} ;

procedure error;
    begin writeln;
    writeln (' INCORRECT INPUT'); goto 99
end {error} ;
```

```
procedure term;
    procedure factor;
    begin
        if sym in ['A' . . 'Z', empty] then getsym
        else if sym = '[' then
                        begin getsym; term;
                                if sym = ']' then getsym else error
                        end else error
    end {factor} ;
begin
    factor;
    while sym in ['A' . . 'Z', '[', empty] do factor
end {term} ;

procedure expression;
begin
    term;
    while sym = ',' do begin getsym; term end
end {expression} ;
```

```
begin {main program}
    while eof(input) do
    begin
        getsym;
        if sym in ['A' . . 'Z'] then getsym else error;
        if sym = ' =' then getsym else error;
        expression;
        if sym ≠ '.' then error;
        writeln; readln;
    end;
99: end .
```

Translator of Language

```
program generalparser (input, output);
label 99;
const empty = ' *';
type pointer = ^node;
       hpointer = ^header;
       node = record suc, alt: pointer;
         case terminal: boolean of
                    true: (tsym: char);
                    false: (nsym: hpointer)
      end;
header record sym: char;
                    entry: pointer;
                    suc: hpointer
         end;
```

```
var list, sentinel, h: hpointer;
     p: pointer;
     ok: boolean;
     sym: char;

procedure getsym;
     begin
     repeat read(sym); write(sym)
     until sym <>' '
end {getsym};

procedure error;
     begin writeln;
     writeln ('INCORRECT SYNTAX');
     goto 99
end {error} ;
```

```
procedure find(s: char; var h: hpointer);
{locate nonterminal symbol s in list, if not present,
insert it}
    var h1: hpointer;
begin h1: = list; sentinel ^.sym : = s;
    while h1 ^.sym ≠ s do h1 : = h1 ^.suc;
    if h1 = sentinel then
        begin {insert} new (sentinel);
            h1 ^.suc : = sentinel; h1 ^. entry : = nil
        end;
    h := h1
end {find};
```

```
procedure term (var p,q,r: pointer);
    var a,b,c: pointer;
    procedure factor (var p,q: pointer);
        var a,b: pointer; h: hpointer;
    begin if sym in [' A' . . 'Z', empty] then
        begin {symbol} new(a);
                if sym in [' A' . . 'H '] then
                begin {nonterminal} find(sym,h);
                        a^. terminal : = false; a^.nsym : = h
                end else
                        begin {terminal}
                        a^. terminal : = true; a^.tsym : = sym
                        end;
                p := a; q := a; getsym
        end else
```

```
                if sym = '[' then
                begin getsym; term(p,a,b); b^ .suc : = p;
                new(b); b^. terminal : = true; b^.tsym : = empty;
                a^.alt : = b; q : = b;
                if sym = ']' then getsym else error
        end else error
        end {factor} ;
begin
        factor(p,a); q : = a;
        while sym in [' A' . . 'Z', '[', empty] do
        begin factor (a^.suc, b); b^ .alt : = nil; a : = b
        end;
        r := a
end {term} ;
```

```
procedure expression (var p,q: pointer);
        var a,b,c: pointer;
begin
term(p,a,c);
c^.suc : = nil;
while sym = ',' do
        begin
            getsym;
            term(a^.alt, b, c): c^.suc : = nil; a : = b
        end;
q := a
end {expression};
```

```
procedure parse (goal: hpointer;  var match: boolean);
        var s : pointer;
begin s : = goal ^. entry ;
        repeat
                if s^ . terminal then
                        begin if s^ .tsym = sym then
                                begin match : = true; getsym
                                end
                              else match : = (s^ .tsym = empty)
                        end
                else parse(s^ .nsym, match);
                if match then s : = s^ .suc else s : = s^ .alt
        until s = nil
end
```

```
begin {productions}
getsym; new(sentinel); list : = sentinel;
while sym ≠ '$' do
   begin find(sym,h) ;
        getsym; if sym = ' =' then getsym else error;
        expression (h^. entry, p); p^.alt : = nil;
        if sym ≠ '.' then error;
        writeln; readln; getsym
   end;
{check whether all symbols are defined}
h : = list; ok : = true;
while h ≠ sentinel do
    begin if h^. entry = nil then
        begin writeln('UNDEFINED SYMBOL' h^.sym);
                ok : = false
        end;
     h : = h^.suc
    end;
```

```
if ¬ ok then goto 99;
{goal symbol}
getsym; find(sym,h); readln; writeln;
{sentences}
while ¬ eof (input) do
    begin write(' '); getsym; parse(h,ok);
    if ok /\ (sym ='.') then writeln (' CORRECT')
                        else writeln (' INCORRECT');
    readln
    end;
99: end .
```