Group	: Computational Linguistics
Topic	: software

Title	: **Syntax for Compiler Writing**
Author	: René Leermakers

Doc.Nr.	: 0172
Date	: 11-02-87
Status	: concept
Supersedes	: -
Distribution	: Software Group
Clearance	: Project
Keywords	: compiler generator, syntax

Institute for Perception Research

Contents

1 Introduction

This document describes the syntax of the input expected by an implementation of the compiler generator, described in R0167. In order to define a compiler with this generator, two files are to be written. One is called the Domain file and contains declaration type of information, i.e. reserved words and symbols, names of terminals and non-terminals, attributes, output files, etc. The other is called the Grammar file and contains the grammar that defines the syntax of the language under consideration, and its translation into the target language.

For an introduction to the basic ideas, see doc R0167. The relation to the present syntax is hopefully self-evident. In section 2, the syntax is described of the two files that form the input of the generator. This is followed by a specification of a small compiler in section 3. The implementation of the generator itself will be documented in a subsequent paper.

2 Syntax of Input

2.1 Domain File

2.1.1 Syntax

The syntax of the domain file is as follows.

```
utt              =["DOMAIN"].alphabetspec.[symbolspec].[wordspec].
                   [typespec].[setspec].[recordspec].[categoriespec].
                   [outputfiles]."END"
alphabetspec     ="ALPHABET".":"."<".{['].character}.">"
symbolspec       ="SYMBOLS".":"."<".{symboldefinition}.">"
wordspec         ="WORDS".":"."<".{worddefinition}.">"
typespec         ="TYPES".":"."<".{typedefinition}.">"
recordspec       ="ATTRIBUTES".":"."<".{recorddefinition}.">"
categoriespec    ="CATEGORIES".":"."<".{catdefinition}.">"
setspec          ="SETS".":"."<".{setelement}.">"
outputfiles      ="OUTPUTFILES".":"."<".{identifier}.">"

setelement       =identifier."="."<".{attribname.":".typename}.
                                    ">"
symboldefinition =catname."=".['].character
worddefinition   =catname."=".['].identifier
typedefinition   =(enumdef|subrangedef|integerdef|booleandef|setdef)
enumdef          =typename."="."(".identifier.{,identifier}.")".";"
```

```
subrangedef      =typename."=".identifier.".",".".identifier.";"
integerdef       =typename."=."INTEGER".;"
booleandef       =typename."=."BOOLEAN".;"
setdef           =typename."=."SET"."OF".typename.";"
typename         =identifier
character        =identifier (of length 1)

recorddefinition =recordname."=."<"."SURFACE"."<".{attribname.":".typename.
                                    ":".defaultvalue}.
                                                ">"
                                     "PROC"."INH"."<".{attribname.":".typename.}
                                                  ">"
                                     "PROC"."SYN"."<".{attribname.":".typename.}
                                                  ">"
                                  ">"
attribname       =identifier
defaultvalue     =(identifier|"[".[identifier{",".identifier}]."]")
recordname       =identifier

catdefinition    ="<".{catlist}.">".
catlist          =catname.{",".catname}.":".recordname
catname          =identifier
```

2.1.2 Example

The following domain contains all relevant topics.

```
DOMAIN

ALPHABET: < a b c d e f g h i j k l m n o p q r s t u v w x y z
            A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
            '1 '2 '3 '4 '5 '6 '7 '8 '9 '0 >
SYMBOLS: < PLUS = '+
           TIMES = '*
           ROUNDOPEN = '(
           ROUNDCLOSE = ')
         >
WORDS: <
          SIMPLE = 'SIMPLE
          SOMMETJE = 'SOMMETJE
       >
TYPES: <
       int = INTEGER;
       bool = BOOLEAN;
```

```
       enum = (enum1,enum2,enum3);
       digit = 1..9;
       setje = SET OF enum;
       >
SETS: <  entry = < number1:int
                   str:STRING_string
                   b:bool
                 >
      >
ATTRIBUTES: <
       uttrecord = < SURFACE < str:STRING_string:empty
                               table:setofentrys:empty
                             >
                     PROC INH <
                              >
                     PROC SYN <
                              >
                   >
       sumrecord  =  < SURFACE < numofmults:int:0
                              >
                     PROC INH < indent:int
                              >
                     PROC SYN < wrotefile:bool
                              >
                     >
       multrecord  =  < SURFACE <
                              >
                     PROC INH <
                              >
                     PROC SYN <
                              >
                     >
         >
CATEGORIES: <
            <UTT:uttrecord>
            <SUM:sumrecord>
            <MULT:multrecord>
            <SIMPLE,SOMMETJE,PLUS,TIMES,ROUNDOPEN,ROUNDCLOSE: TERMINAL>
            >
OUTPUTFILES: < of1
             >
END
```

2.1.3 Explanation

Below each section in the Domain file is explained.

- Under the caption ALPHABET the characters are given that are allowed to occur in reserved words and identifiers. A quote may be put before each character in this list, but it is obligatory in the cases '<, '>, '(, '), '[, '], '=, ':, '; ', . These are the reserved symbols of the Domain file syntax itself.
- Under the caption SYMBOLS, the reserved symbols are mentioned and coupled to token names. The symbol characters must be characters that are not in the alphabet of words and identifiers. A declaration like PLUS = '+ means that the scanner will create the token PLUS if it sees the plus character. Again a quote before the symbol character is optional, but is obligatory before the above mentioned characters.
- Under the caption WORDS the reserved words are defined and coupled to token names. In the above example, the word "SIMPLE" is to be recognized by the scanner. If it is encountered the token SIMPLE is to be created. The reserved-word definitions are to be given in alphabetical order (The words themselves are to be ordered, not the token names). Quotes before the reserved-word string are obligatory if one of the symbols "<", ">", "(", ")", "[", "]", "=", ":", ";", "," occurs in it.

The above three sections of the Domain file together define the scanner of the compiler that is generated. The next sections are to define the possible attribuations of non-terminal syntax-tree nodes.

- Under the caption TYPES pascal-like type definitions can be specified. The possibilities are integer, boolean, subrange and enumeration types.
- Under the caption SETS one can define (structured) types one wants to define set operations on. In the above example the type SETOFentrys will be created automatically as an abstract data type with four operations. Let S be a variable of type SETOfentrys and let E be of type entry. Then

  INITsetofentrys(S) makes S empty.

  APPENDentry(E,S) adds E to S.

  TAKEentry(E,S) assigns to E an element of S, which is removed from the set.

  STILLentrys(S) is a boolean which is true iff S is not empty.

  These set data types are implemented as lists, and may thus also be used as tuples or tables. One should then know that after APPENDentry(E,S), E is the last element of S, and that TAKEentry(E,S) takes the first element out.

- Under the caption ATTRIBUTES, feature bundles are defined, containing three kinds of attributes, surface attributes and procedural attributes which are of inherited or synthesized kind. The roles of these attributes are discussed below. The attributes are to be chosen out of the simple types defined under TYPES, or are of the abstract set data types declared under SETS, or the externally defined string type STRING_string. Surface attributes also have a default value. The default value empty means that the attribute does not have a default.
- Under CATEGORIES the non-terminals of the grammar are to be specified, and coupled to the attribute bundles defined under ATTRIBUTES. Also the terminals are given with as feature bundle TERMINAL. This bundle contains one (surface) attribute "str" of type STRING_string. Identifiers, hence, have this attribute, and the scanner assigns to it the string that gives rise to the identifier. The starting-symbol non-terminal is to be called UTT, and the corresponding feature bundle may only contain surface attributes.
- Under OUTPUTFILES the names of any number of files can be given. The output files are global variables. This is for implementation reasons, but one can view a file as an attribute of each non-terminal, which is both inherited and synthesized.

2.2 Grammar file

Again, first the syntax is given and after that an example file consisting of a rule of the grammar of the final section. The rule uses files, terminals, non-terminals, attributes and types according to the above Domain file.

2.2.1 Syntax

```
utt = {rule}
rule = "%".catname."BASIS RULE".barerule."SURFACE PART".surfrule.
      "PROCEDURAL PART".procrule."&"

barerule = ident."=".graph.{helpgraph}
graph = concgraph.{"|".concgraph}
concgraph = elementarygraph.{".".elementarygraph}
elementarygraph = "(".graph.")" | "[".graph."]" | "{".graph."}" |
                   ident | ident."/".number
helpgraph = ident."=".graph
number = '1'|'2'| .. |'99'

surfrule = ["VAR".{varlist.("::".domaintype|":".pascaltype)}].
           "BEGIN"."<*".initblock1.{block1}.finalblock1."*>"."END;"
initblock1 = "INIT:".compoundstatement.";"
```

```
block1 = number.":"."<*"."LOCALCONDITION:".booleanexpression.
                        "GLOBAL:"."#CONDITION:".booleanexpression.
                                  "#ACTION:".compoundstatement.
                    "*>"
finalblock1 = "FINAL:"."#CONDITION:".booleanexpression.
                      "#ACTION:"."BEGIN"."SEND_"cat.{statement}."END"

procrule = "VAR".[{varlist.("::".domaintype|":".pascaltype)}].
           "BEGIN".initblock2."<*".{block2}."*>".finalblock2."END;"
initblock2 = "INIT:".compoundstatement.";"
block2 = number.":"."<*"."SEND:".compoundstatement.
                         "!"cat".".
                        "RECEIVE:".compoundstatement.
                    "*>"
finalblock2 = "FINAL:".compoundstatement
```

2.2.2 Example

```
%SUM
BASIS RULE
  SUM = MULT/2.{othersum}
  othersum = PLUS/1.MULT/2
SURFACE PART
VAR multcounter::int;
BEGIN
<*
 INIT:BEGIN multcounter:=0
       END;
 1    :<*
        LOCALCONDITION:TRUE
        GLOBAL: #CONDITION: TRUE
                #ACTION:  BEGIN END
       *>
 2    :<*
        LOCALCONDITION: TRUE
        GLOBAL: #CONDITION: TRUE
                #ACTION: BEGIN multcounter:=multcounter+1 END
       *>
 FINAL: #CONDITION: TRUE
         #ACTION: BEGIN
                  SEND_SUM;
                  $SUM.numofmults:=multcounter
                  END
```

```
*>
END;
PROCEDURAL PART
VAR numofmults,multcounter,indent::int;
BEGIN
INIT: BEGIN multcounter:=0;numofmults:=$SUM.numofmults;indent:=$$SUM.indent
      END;
<*
1: <*SEND: BEGIN END
     !.
     RECEIVE: BEGIN END
   *>
2: <* SEND: BEGIN multcounter:= multcounter + 1;
                  IF multcounter<>numofmults THEN
                      BEGIN
                      BEGIN \of1. "SUM(" END;
                      indent:=indent+4
                      END;
                  %%MULT.indent:= indent
            END
      !MULT.
      RECEIVE: BEGIN IF numofmults<>multcounter THEN
                      BEGIN \of1. "," \\ TAB(of1,indent) END
               END
   *>
*>
FINAL:BEGIN WHILE multcounter<>1 DO
               BEGIN
               multcounter:=multcounter - 1;
               BEGIN \of1. \\ TAB(of1,indent-1); ")" END;
               indent:=indent-4
               END
      END
END;
&
```

2.2.3 Explanation

- Under BASIS RULE, the non-terminal mentioned after "%" at the start of the rule is 'defined' by an arbitrary regular expression of terminals and non-terminals. Each (non)terminal in the regular expression is accompanied by a number. Each non-terminal may have only one such definition. The r.h.s. of a basis rule may contain (and does, in the above rule) help non-terminals that

are defined by help rules below the principal rule.

- Under SURFACE PART, the surface attributes of the non-terminal at the l.h.s., also called the rule non-terminal, of the basis rule are evaluated in terms of the surface attributes of the r.h.s. (non)terminals. Identifiers in this section are either parameters or attributes. Parameters are defined after VAR, and receive initial values under INIT. Parameters may be of PASCAL types or of types defined in the Domain File. The latter kind of parameters are declared with "::" between parameter name and typename, the former with ":".

  Each numbered block corresponds to one or more elements of the regular expression. If the element is an identifier or non-terminal, a unique block corresponds to it and the attributes referred to in the block are the attributes of this element, which called the block non-terminal or block identifier. Attributes are prefixed with "%" followed by the block non-terminal, and a "." . The surface attribute "str" of an identifier is referred to as %.str. A numbered block contains three parts; a local condition involving attributes only, a global condition involving attributes and parameters and an action part in which parameters are assigned values in terms of parameters and attributes. Under FINAL the attributes of the rule non-terminal are set in terms of the rule parameters. The first statement of this section has to be "SEND_", followed by the rule non-terminal. The attributes in FINAL are referred to as above but with "$" instead of "%".

- Under PROCEDURAL PART, the procedural attributes (the attributes declared under PROC INH and PROC SYN in the Domain File) and outputfiles are evaluated. Parameters are used just as in the surface part of the rule. Under INIT the parameters get values in terms of the PROC INH and SURFACE attributes (with $$,$ before the rule non-terminal,respectively). Under FINAL the PROC SYN attributes of the rule non-terminal get a value in terms of parameters.

  Each numbered block again consists of three parts; a SEND part in which the PROC INH attributes (with %% before the block non-terminal) and parameters get values, a wait statement - the name of the block non-terminal between ! and . - , which causes a wait for a calculation of synthesized attributes of the block non-terminal and a RECEIVE section in which parameters get values in terms of parameters and the calculated PROC SYN attributes and surface attributes(with %%, % before the block non-terminal, respectively). If a numbered block corresponds to a terminal, the wait statement must be the empty one (!.).

  Outputfiles are of type FILES_text and strings of type STRING_string. The procedures and functions defined on these data types can be found in the archive files GENERAL:FILES.ENV and GENERAL:STRING.ENV. Because the use of these functions sometimes becomes somewhat tedious, a few shorthands exist for writing in files. Shorthands appear in a compound write statement which is to start with a declaration of the output file name,

```
BEGIN \of1. {shorthand} END,
```

where shorthand is

```
shorthand = \\ | "string" | ""stringvar"" | """integervar""".
```

The first option starts a new line. The second writes the string "string", the third the value of the variable stringvar, which is of type STRING_string. The fourth and last shorthand writes the value of the integer variable integervar. The statement TAB(of1,i) writes i spaces in file of1. This statement may be put either inside or outside a compound write statement. Writing in output files is legitimate in each block of the procedural part.

3 A Small Compiler

The following compiler translates an arbitrary arithmetic expression containing only the product and sum operations into a binary representation, e.g.:

```
SIMPLE SOMMETJE (a*b+c*(p+q)+e*f)*3 +4

->

SUM(PROD(SUM(PROD(a,
                  b
                 ),
             SUM(PROD(c,
                      SUM(p,
                          q
                         )
                     ),
                 PROD(e,
                      f
                     )
                )
            ),
         3
        ),
    4
   )
```

3.1 Domain File

```
DOMAIN

ALPHABET: < a b c d e f g h i j k l m n o p q r s t u v w x y z
            A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
            '1 '2 '3 '4 '5 '6 '7 '8 '9 '0 >
SYMBOLS: < PLUS = '+
           TIMES = '*
           ROUNDOPEN = '(
           ROUNDCLOSE = ')
         >
WORDS: <
          SIMPLE = 'SIMPLE
          SOMMETJE = 'SOMMETJE
```

```
       >
TYPES: <
       int = INTEGER;
       >
SETS: <
      >
ATTRIBUTES: <
       uttrecord = < SURFACE <
                              >
                     PROC INH <
                              >
                     PROC SYN <
                              >
                   >
       sumrecord  =  < SURFACE < numofmults:int:0
                              >
                     PROC INH < indent:int
                              >
                     PROC SYN <
                              >
                     >
       multrecord  =  <SURFACE < numofelements:int:0
                               >
                      PROC INH < indent:int
                               >
                      PROC SYN <
                               >
                      >
       elementrecord = < SURFACE <
                                  >
                          PROC INH < indent:int
                                   >
                          PROC SYN <
                                   >
                      >
         >
CATEGORIES: <
            <UTT:uttrecord>
            <SUM:sumrecord>
            <MULT:multrecord>
            <ELEMENT:elementrecord>
            <SIMPLE,SOMMETJE,PLUS,TIMES,ROUNDOPEN,ROUNDCLOSE: TERMINAL>
            >
OUTPUTFILES: < of1
```

12

```
                >
END
```

3.2 Grammar File

```
%UTT
BASIS RULE
  UTT = SIMPLE/1.SOMMETJE/1.SUM/2
SURFACE PART

BEGIN
<*
 INIT:BEGIN
       END;
 1    :<*
        LOCALCONDITION:TRUE
        GLOBAL: #CONDITION: TRUE
                #ACTION:  BEGIN
                          END
       *>
 2    :<*
        LOCALCONDITION:TRUE
        GLOBAL: #CONDITION: TRUE
                #ACTION:  BEGIN
                          END
       *>
 FINAL: #CONDITION: TRUE
         #ACTION: BEGIN
                  SEND_UTT;
                  END
*>
END;
PROCEDURAL PART
VAR indent::int;
BEGIN
INIT:BEGIN indent:=0; FILES_open(of1,'outputfile',10,3) END;
<*
   1: <* SEND: BEGIN END
         !.
         RECEIVE: BEGIN
                  END
      *>
   2: <* SEND: BEGIN %%SUM.indent:=indent END
         !SUM.
         RECEIVE: BEGIN END
      *>
*>
```

```
FINAL: BEGIN END
END;
&
%SUM
BASIS RULE
  SUM = MULT/2.{PLUS/1.MULT/2}
SURFACE PART
VAR multcounter::int;
BEGIN
<*
 INIT:BEGIN multcounter:=0
       END;
 1    :<*
        LOCALCONDITION:TRUE
        GLOBAL: #CONDITION: TRUE
                #ACTION:  BEGIN END
       *>
 2    :<*
        LOCALCONDITION: TRUE
        GLOBAL: #CONDITION: TRUE
                #ACTION: BEGIN multcounter:=multcounter+1 END
       *>
 FINAL: #CONDITION: TRUE
         #ACTION: BEGIN
                  SEND_SUM;
                  $SUM.numofmults:=multcounter
                  END
*>
END;
PROCEDURAL PART
VAR numofmults,multcounter,indent::int;
BEGIN
INIT: BEGIN multcounter:=0;numofmults:=$SUM.numofmults;indent:=$$SUM.indent
      END;
<*
1: <*SEND: BEGIN END
     !.
     RECEIVE: BEGIN END
   *>
2: <* SEND: BEGIN multcounter:= multcounter + 1;
                  IF multcounter<>numofmults THEN
                      BEGIN
                      BEGIN \of1. "SUM(" END;
                      indent:=indent+4
```

```
                    END;
                  %%MULT.indent:= indent
            END
      !MULT.
      RECEIVE: BEGIN IF numofmults<>multcounter THEN
                    BEGIN \of1. "," \\ TAB(of1,indent) END
               END
   *>
*>
FINAL:BEGIN WHILE multcounter<>1 DO
        BEGIN
                multcounter:=multcounter - 1;
                BEGIN \of1. \\ TAB(of1,indent-1); ")" END;
                indent:=indent-4
        END
      END
END;
&
%MULT
BASIS RULE
  MULT = ELEMENT/2.{TIMES/1.ELEMENT/2}
SURFACE PART
VAR elementcounter::int;
BEGIN
<*
 INIT:BEGIN elementcounter:=0
       END;
 1    :<*
        LOCALCONDITION:TRUE
        GLOBAL: #CONDITION: TRUE
                #ACTION:  BEGIN END
       *>
 2    :<*
        LOCALCONDITION: TRUE
        GLOBAL: #CONDITION: TRUE
                #ACTION: BEGIN elementcounter:=elementcounter+1 END
       *>
 FINAL: #CONDITION: TRUE
         #ACTION: BEGIN
                  SEND_MULT;
                  $MULT.numofelements:=elementcounter
                  END
*>
END;
```

```
PROCEDURAL PART
VAR numofelements,elementcounter,indent::int;
BEGIN
INIT: BEGIN elementcounter:=0; numofelements:=$MULT.numofelements;
            indent:=$$MULT.indent
      END;
<*
1: <*SEND: BEGIN END
     !.
     RECEIVE: BEGIN END
   *>
2: <* SEND: BEGIN elementcounter:= elementcounter + 1;
                  IF elementcounter<>numofelements THEN
                     BEGIN
                     BEGIN \of1. "PROD(" END;
                     indent:=indent+5
                     END;
                  %%ELEMENT.indent:= indent
            END
      !ELEMENT.
      RECEIVE: BEGIN IF numofelements<>elementcounter THEN
                        BEGIN \of1. "," \\ TAB(of1,indent) END
               END
   *>
*>
FINAL:BEGIN WHILE elementcounter <> 1 DO
                BEGIN
                elementcounter:=elementcounter -1;
                BEGIN \of1. \\ TAB(of1,indent-1); ")" END;
                indent:=indent-5
                END
      END
END;
&
%ELEMENT
BASIS RULE
  ELEMENT = (ROUNDOPEN/1.SUM/2.ROUNDCLOSE/1|IDENTIFIER/3)
SURFACE PART
<*
 INIT:BEGIN
       END;
 1    :<*
        LOCALCONDITION:TRUE
        GLOBAL: #CONDITION: TRUE
```

```
                #ACTION:  BEGIN
                          END
       *>
 2     :<*
         LOCALCONDITION: TRUE
         GLOBAL: #CONDITION: TRUE
                 #ACTION: BEGIN
                          END
       *>
 3     :<*
         LOCALCONDITION: TRUE
         GLOBAL: #CONDITION: TRUE
                 #ACTION: BEGIN
                          END
       *>
 FINAL: #CONDITION: TRUE
          #ACTION: BEGIN
                   SEND_ELEMENT;
                   END
*>
END;

PROCEDURAL PART
VAR indent::int;str:STRING_string;
BEGIN
INIT: BEGIN indent:=$$ELEMENT.indent END;
<*
1: <* SEND: BEGIN END
      !.
      RECEIVE: BEGIN END
   *>
2: <* SEND: BEGIN %%SUM.indent:= indent END
      !SUM.
      RECEIVE: BEGIN END
   *>
3: <* SEND: BEGIN END
      !.
      RECEIVE: BEGIN str:=%.str; BEGIN \of1. ""str"" END END
   *>
*>
FINAL:BEGIN END
END;
&
```