TWS - A translator writing system

(Ján Šturc)

INTRODUCTION

The Translator Writing System (TWS) is a programming language designed to automate programming of tasks involving parsing. TWS can be used advantageously in all text processing problems, from simple investigations of the local characteristics of textual information to writing compilers for general or specialized languages. In the statistical practice it will be suitable for forxnatting-reformatting tasks and text editing purpose. Although TWS is a general programming tool, its use for parsing a language whose gram mar is originally in nondeterministic form requires much more effort than if the grammar were in deterministic form.

Because TWS itself is a compiler the method of self-compilling has been used to implement it. This method also suggested to us that the TWS language could be used to des cribe it syntax in this manual. This decision made the manual more instructive, but on the other hand, it forces the user to read it more carefully and at least twice.

The manual consists of three parts. The first of them describes the language for the description of the syntax of our object compiler. The second deals with the procedures corresponding to individual right-hand sides (phrases). The third is the description of the interface to the MASTER operating system. Finally, some examples are given as appendices.

Note:

1. In the original version of the TWS lexical analysis was a part of syntactical one. In the version presented here, we separated the vocabulary (alphabet) and lexica from the syntax .
2. Maybe, more appropriate then term hardware representation used below would be term internal representation.

1. **Formal definition of the TWS vocabulary and lexica.**

1.1. BASIC SYMBOLS

1.1.1. Syntax

```
ANY(  TERMINAL(LETTER( [‘A’ – ‘Z’])
                 DIGIT ( ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’)
                 OPERATOR (‘|‘,‘-‘),
                 DELIMITER(   STARTCOMMENT (‘<‘), ENDCOMMENT (‘>’),
                              COMMA (’,’),
                              PERIOD (’.’),       < DOT >
                              SEMICOLON (’;’)’
                              ASSIGN (‘=‘),
                              QUOTE (  ‘ ’ ’  ),
                              ASTERISK (‘*‘),
                              BLANK (‘␣’, TAB, LF, CR ),
                              LEFTPAR (’(’), RIGHTPAR(‘)’)
                              DOLLAR(‘$’), COLON(‘:’),
                              LEFTBRA(‘[‘), RIGHTBRA(‘]’),
                              LEFTCURLY(‘{‘), RIGHTCURLY( ’}’ ),
                 OTHER( List of the symbols, which are not applicable in TWS )
      VARIABLE ( ID, IDENTIFIER, NIJM, NUMBER, MLE, ELEMENT,
                 PROGRAM, DECLARPART, DEFI NITION, RW,
                 RESERVEDWORDS, PRODPART, DECLARATION,
                 STARTDECL, DECL, RSW, PRODUCTION,
                 LEFTSIDE, RIGHTPARP, TERMINATION,
                 EXPRESSION, LCOMPONENT) )
 $ RWORD (BEGIN, END, PRODUCTIONS, RWORD);
```

Notes:

1. ‘卄’ means the internal value of the alphabet symbol which on the display (paper or keyboard) looks like 卄.
2. Internal values are integers.
3. To variables are assigned internal values immediately followed last terminal symbol. They are enumerated in the order they appears in variable declaration.

1.1.2. Semantics

Basic symbols form the alphabet of terminal symbols of the

grammar, and meaning is derived from the grammatical categories which they describe.

In general: Letters constitute the basic atoms out of which identifiers are constructed. Digits, on the other hand, are the building blocks for numbers. The operators land ← have the meaning of set union and difference respectively. The period is used for concatenation. Parentheses serve to en close lists or expressions. Comment brackets distinguish comments from the rest of the text. All the other delimiters serve to separate the various constructs of the language. The class of other symbols are not symbols of the TWS language. They may, however, occur in symbol definitions of the object language or in comments.

1.1.3. Hardware representation

The hardware representation of the basic symbols are all non negative integers in binary form. In our implementation we have limited ourselves to 64 basic symbols to which numbers 0 — 63 have been assigned as follows:

0 ← 0	13 ← ≤	26 ← <	39 ← P	52 ← U
1 ← 1	14 ← “	27 ← .	40 ← Q	53 ← V
2 ← 2	15 ← [	28 ←)	41 ← R	54 ← W
3 ← 3	16 ← +	29 ← ≥	42 ← ∀	55 ← X
4 ← 4	17 ← A	30 ← ¬	43 ← $	56 ← Y
5 ← 5	18 ← B	31 ← ;	44 ← *	57 ← Z
6 ← 6	19 ← C	32 ← -	45 ← ↑	58 ←]
7 ← 7	20 ← D	33 ← J	46 ← ↓	59 ← ,
8 ← 8	21 ← E	34 ← K	47 ← >	60 ← (
9 ← 9	22 ← F	35 ← L	48 ← ⊔	61 ← →
10 ← :	23 ← G	36 ← M	49 ← /	62 ← ≡
11 ← =	24 ← H	37 ← N	50 ← S	63 ← ∧
12 ← ≠	25 ← I	38 ← O	51 ← T	

Note:

1. Definition n ← キ should be read: the internal value n is assigned to the symbol which looks like キ.
2. In times of the TWS design, there was no commonly accepted norm for characters encoding.

1.1.4. Error diagnostics

In case one of the class of other symbols occurs in the source text outside the symbol definition or comment, it has a delimiting effect equivalent to the effect of a blank and the diagnostic ILLEGAL SYMBOL is produced.

1.2 IDENTIFIERS AND NUMBERS

1.2.1 Syntax

```
    IDENTIFIER. 2 = ID.(ANY-LETTER)
    ID      =       ID.LETTER
    ID      =        LETTER
    NUMBER. 2   =  NUM.(ANY-DIGIT)
    NUM   =  NUM.DIGIT
    NUM   =  DIGIT
          =  BLANK  <Last of all the blanks are expunged.>
```

Note:

In case of conflict (more than one rule is applicable), the rule appeared in the program earlier (higher) is chosen.

1.2.2. Semantics

Identifiers do not have any special significance in the language, they mean themselves, and they are used to denote terminal symbols and variables or their classes. Numbers are used to indicate right-hand side items which are rewritten on the left-hand side or for the identification of semantic procedures. In addition numbers are used to re present symbols in their internal representation (e.g. BLANK (48)).

1.2.3. Hardware representation

Identifiers are represented as sequences of basic symbols stored one behind the other. Identifiers are stored by means of a hash table. This table always gives data on their length of the identifier, i.e. on the number of basic symbols of which it consits, and the address of its first basic symbol. The dimensions of the table as well as the space reserved for the storage of identifiers are installation parameters.

Numbers are stored in binary form. The range of a number is $0 \leq n < 2^{15}$.

1.2.4. Error diagnostics

No syntactic errors can occur.

If the number of identifiers exceeds the range of the hash table, the message “TOO MANY IDENITIPIERS” is produced.

If the space reserved for storage of identifiers is occupied, the message “TOO LONG IDENTIFIERS” is produced.

No diagnosis is provided for checking the number range.

1.2.5. Examples

The string SANA211LX means, in the sense of the syntax defined above, identifier SANA, number 211 and identifier LX.

2. Syntax of the TWS langauge

2.1. DECLARATIONS

2.1.1. Syntax

ELEMENT =   ELE.QUOTE

ELE = QUOTE. TERMINAL

DECLARATION = DECL. RIGHTBRACKET

STARTDECL = IDENTIFIER. LEPTBRACKET

DECL = DECL. COMMA. (DECLARATION | ELEMENT | NUMBER)

DECL. 4 = DECL. COMMA. IDENTIFIER. (COMMA | RIGHTBRA)

DECL = STARTDECL. (DECLARLTION | ELEMENT | NUMBER)

DECL. 3 = STARTDECL. IDENTIFIER. (NUMBER | RIGHTBRACKET)

2.1.2. Semantics

Declarations are used to define terminal symbols or variables or classes of them. An identifier denotes a defined class and a list in parenthesis defines the symbols of which the defined class consists. In particular, if a class consists of just one symbol, this symbol is defined. A symbol can be defined as a character; this definition defines the character as. a number corresponding to its hardware representation or as a quoted character which means the hardware representation of the character in the given installation.

The nested class definition means that the outer class consists of disjunction of all the symbols in the inner classes.

Finally, variables can be declared by the occurrence of their identifier and the compiler will assign to them as their hardware representation the next integer in order.

2.1.3. Hardware representation

Symbol classes are internally represented by a binary column vector of a length equal to the number of all the symbols, the i-th item of which has the value 1 if the symbol whose hardware representation is i belongs to the given class; otherwise its value is zero.

2.1.4. Error diagnostics

If left and right parentheses in definitions do not match, an error has occurred and one of the messages LEFT PARENTHESIS MISSING or RIGHT PARENTHESES MISSING is generated as many times as there are parentheses missing. In case an element is not distinguished by the right quote, the quote is added and the message QUOTATION is produced. If there is an extra left quote, which means the right quote is missing, or if an error is detected involving the separating comma, the compiler sends out the message DELIMITER.

2.1.5. Examples

PLUS (16) declares the symbol PLUS whose physical representation is the binary number 16. DIGIT ‘0’, ‘1’, ‘2’, ‘3’ declares the class containing the symbols 0, 1, 2, 3. VARIABLE IDENTIFIER, SYMBOL, DECLARATION declares the class consisting of variables whose names are IDENTIFIER, SYMBOL, DECLARATION, if these have not been declared so far they will be considered variables and the compiler will allot an internal representation to them.

ANY(TERMINAL (LETTER(‘A’, ‘B’, ‘C’), DIGIT (‘0’, ‘1’), OPERATOR(16, 32)), NONTERMINAL(E, T, P))

declare the following classes and symbols

Letter = { A, B, C}

Digit = {0, 1}

Operator = {+, -} symbols whose internal representetion is 16, 32

{E}, {T}, {P} variables

Terminal = {A, B, C, 0, 1, +, - }

Nonterminal = {.E, T, P}

ANY = {A, B, C, 0, 1, +, -, E, T, P }.

2.2. EXPRESSIONS

2.2.1. Syntax

EXPRESSION = LCOMPONENT. RIGHTBRA

LCOMPONENT = LEFTBRACKET. IDENTIFIER

LCOMPONENT = LCOMPONENT. OPERATOR. IDENTIFIER

2.2.2. Semantics

Expressions serve to form new classes of symbols combining the symbols and classes already declared. New classes are formed with the help of set operations; union (|) and difference (-). Both operators have the same priority and expressions are evaluated left to right. Nesting of parenthesis is not permitted.

2.2.3. Physical representation

These operations are represented by operations on vectors of the appropriate classes. Operations are performed item by item in accordance with the following tables:

|	0	I
0	0	I
I	I	I

-	0	I
0	0	0
I	I	0

2.2.4. Error diagnostics

The following errors are diagnosed: missing parentheses

(LEFT PARENTHESES MISSING or RIGHT PARENTHESES MISSING) wrong operator symbol or missing operator (OPERATOR MISSING), superfluous operator (OPEBAND MISSING)

2.2.5. Examples

VAR IABLE - IDENTIFIER

TERMINAL - SEPARATOR | PARENTHESES

LETTER | DIGIT

Note: Nesting of parentheses is not permitted. The operations are evaluated from left to right.

2.3. PRODUCTIONS

2.3.1. Syntax

RIGHTPART = RIGHTPART. PERIOD. (IDENTIPIER | EXPRESSION)

RIGRTPART = ASSIGN. IDENTIFIER/EXPRESSION

LEPTSIDE = LEPTSIDE. PERIOD. NUMBER/IDENTIFIER

LEFTS IDE   NUMBER/IDENTIFIER

TERMINATION = ASTERISK. NUMBER. SEMICOLON

TERMINATION = SEMICOLON

PRODUCTION = LEFTSIDE. RIGHTPART. TERMINATION

PRODUCTION = RIGHTPART. TERMINATION

2.3.2. Semantics

The TWS program seeks in the source text the leftmost occurrence of some right-hand side. In case several productions have the same right-hand side, the program selects the first one in order. This right-band side is replaced by the corresponding left side of the selected production and the semantic procedure identified in the termination of the production is performed. This cycle goes on until the final production is reached, i.e. the production which in its termination defines the program which terminates the TWS job. The right-hand side of the production which instead of symbol identifiers contains class symbols identifiers or expressions actually describe some of the right-hand sides which we get by replacing the proper class or expression by individual symbols from it. Numbers on the left-hand side serve to rewrite the current symbols from the right side by identifying their position on the right-hand side.

2.3.3. Hardware representation

The right parts of the production are represented by a matrix consisting of the column vector of its item. The left-hand sides are given by the left-hand side table. This table has as many entries as there are productions. The data on the left side of the n-th production are stored at the n-th entry of the table. In the table the length of the left-hand side is stored together with the pointer to the strings representing the left-hand side. In the working space the actual left sides are stored as follows: If the element of the left side is a symbol, this symbol is stored; if the element is a number, the relative pointer from the top of the stack to the corresponding symbol is stored.

2.3.4. Error diagnostics

The following errors are diagnosed:

missing period or some other symbol instead of the period (CATENATION ERROR), incorrect termination of production or missing asterisk, semicolon, or number (TERMINATION), empty right side (EMPTY RIGHT SIDE).

2.3.5. Examples

```
ID = ID. LETTER * 2;
NUMBER. 2 = NUM. (ANY—DIGIT) * 1700;
  = BLANK     *1;
DECL. 4 = DECL. COMMA. IDENTIFIER. (COMMA/RIGHTBRACKET) *15;
2.1 = A. B * 13;          <change order of symbols>
```

2.4. RESERVED WORDS

2.4.1. Syntax

```
RW = RW. COMMA. IDENTIFIER
RW = RWORD, LEPTBRACKET. IDENTIFIER
RESERVEDWORDS = RW. RIGHTBRACKET. SEMICOLON
```

2.4.2. Semantics

Reserved words have the same meaning as symbol declarations,

but TWS initiates the hash table of the object program with these reserved words. Then in the semantic procedure an identifier just read may be replaced by its internal value, if it was a reserved word. Our program automatically establishes coercion between the variable denoted by the same identifier as the reserved word, and this reserved word.

2.4.3. Hardware representation

Hardware representation is the same as in the case of the declaration. Moreover, a table is generated as described in 2.1.3.

2.4.4. Error diagnostics

The following errors are diagnosed:

LEFT PARENTHESIS MISSING, RIGHT PARENTHESIS MISSING, DELIMITER, the description is given in 2.1.3 SEMICOLON MISSING if a semicolon is missing after the definition of reserved words.

2.4.5. Examples

BEGIN, END, RWORD. PRODUCTIONS;

The coercion between a reserved word and the corresponding metavariable is done automatically by the TWS.

2.5. PROGRAM

2.5.1. Syntax

DECLARATIONPART = DECLARATIONPART. DECLARATION. SENICOLON

DECLARATIONPART = BEGIN

PRODUCTION PART= PRODUCTIONPART. PRODUCTION

PRODUCTION PART PRODUCTIONS

PROGRAM = DECLAPAPIONPART. PRODUCTIONPART, ENDSYMBOL

PROGRAM

= DBCLABATIONPART. RESERVED WORDS. PRODUCTION PART. END

2.5.2. Semantics

Semantics is the sum total of the semantics of the individual items. Productions of this kind serve above all to ensure that the program preserves its structure in the language. If this requirement were neglected, we can only analyze the individual items quite independently.

2.5.3. Does not apply

2.5.4. Error diagnostics

The following error messages are produced:
BEGIN if no initial symbol was found
END if the input file was emptied and the termination symbol was not found.

PRODUCTIONS in case the separating symbol is missing SEMICOLON MISSING if the semicolon separating two declarations is missing.

2.5.5. Examples of programs See Appendix B