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 (’.’),

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 keybord) 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 cate gories 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 uumberB. 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 lan guage. They may, however, occur in symbol definitions of the object language or in comments.

1.1.3. Hardware representation

The hardware representation of 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 ← ㅍ should be read: the internal value n is assigned to the symbol which looks like ㅍ.
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 then one rule is appplicable), the rule appeared in the program earlyer (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 additioia, 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 storeed 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 identifieri 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.