A COMPILER ORIENTED SYNTAX DEFINITION

Ernő FARKAS
Computer and Automation Institute,
Hungarian Academy of Sciences
Budapest, Hungary

It is well-known that the meta-language was a very important discovery on the way of the more precise description of programming languages, and of the development of common translation techniques. However, it is well-known too, that the meta-language is not suitable for each language and even in the languages described well by the meta-language there are parts of the syntax which are out of the definition, for example: if there is an array in the program declared as two dimensional and we use it with three indexes then most of the compilers send an error message, however, this fact may not be established on the basis of the meta-language.

Here in after we want to give a syntax definition based on the meta-language, although the definitional rules are also taken into consideration. Here the "definitional" attribute is used in a very wide sense. The scheme written below makes it possible to examine such properties of the program which were earlier considered as a part of the semantics or a tool of the program debugging. For example, we may check whether an index variable of a cycle is modified inside the cycle, or the fact that in a part of the program which variable can get a value and so on. So we have the possibility to send one error message for one error, in that point where the mistake is most striking. This type of the definition does not mean a new type translation technique but this step allows to get a higher compatibility between the different implementations of the language:

1. We have the possibility to decide more precisely, which kind of program is correct formally and which is erroneous.
2. What kind of errors are required to be detected in the level of translation /and what kind of error in the running/.
3. It is possible to create a uniform error diagnostic system for a language.

THE SYNTAX DEFINITION

Let be "A" the set of the permitted symbols of the language, and we will denote with "A^*" the set of the finite strings from the elements of A.

"A language is a set of such strings from A^* which are corresponded to prototypes" derived by a meta language.

$L \subset A$ will be a language if it fulfils the definition below:

Let be the triplet $<B,s,\gamma>$ a meta-language, where $B=T \cup N$ and $T \cap N=0$. T is the set of the terminal symbols and N is the set of the nonterminal symbols.
$s \in N$ is the beginning symbol.
γ is a finite set of substituting rules, in the form $n \rightarrow x$, where $n \in N$ and $x \in (T \cup N)^*$.
Let be further $T=A \cup E$ and $A \cap E=0$, where A is the set of permitted symbols, as above; and E is the set of so-called elementary objects. Hence

$$A \subset T \subset B \ .$$

Let be given in addition an infinite enumerable set, V /the set of the states of the vocabulary/ and v_0 its special element the beginning state. Let be $F \subset V$ the set of the legal final states.

At the end, let be $g \in \{(A \times V \times E) \to V\}$ a partial function the so-called vocabulary function.

Let $l \in L \subset A^*$, if and only if there exists a partition of

$$l = x_1, x_2, \ldots x_n \quad (x_i \in A^*)$$

so that there exists such a $t \in T^*$ which can be derived from s by the rules of the meta-language, /in the usual way/, and

$$t = y_1, y_2, \ldots y_n \quad (y_i \in T^*)$$

and "l" match to "t" in the sense:

if $y_i \in A^*$ then $x_i = y_i$

else $y_j \in$ and

$$g(x_{j1}, v_0, y_{j1}) = v_{j1}$$
$$g(x_{j2}, v_{j1}, y_{j2}) = v_{j2}$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$g(x_{jk}, v_{jk-1}, y_{jk}) = v_{jk}$$

for all $y_j \in E$, and $v_{jk} \in F$.

This means informally:

By means of the meta-language we are forming a tree structure On the leaves of the tree, there are either strings from A^* /key words/ or elementary objects /labels, variables, etc./. It must be an one-one correspondence between the key words in the tree and the key words in the object language. If there is an elementary object on the leaf of the tree, we have to decide by the function "g" whether the corresponding string "a" is compatible with the elementary object "e" and with the present state "v" of the vocabulary. If it is so, then we can go on, and the vocabulary gets a new state. If they are not

compatible, we have several ways for sending error messages and it is advisible to define also these ways at the forming of the syntax. Finally, the vocabulary must have a state in which all the references are satisfied, i.e. in the program there may not occur any object or attribute of an object which was referred but not established.

The vocabulary function is shown in the Appendix in a rather tedious example.

APPENDIX

Let us suppose that we have a language, which is very close to the FORTRAN II. /The FORMAT, EQUIVALENCE, COMMON instructions are not involved into the example, but they may be realized without any difficulties. The only restriction is that the label at the end of a cycle must be the label of a CONTINUE instruction./ It is important for the fact that no elementary objects of the body of the cycle may appear in the program after that point where the label indicates the end of the cycle.

Let us have a small program:

```
      DIMENSION X(50)
      READ K
      DO 110 I=1,K
      READ X(I)
110   CONTINUE
      Y=0
      Z=0
      DO 120 1=1,K
      IF(X(I))111,120,112
111   Y=Y+X(I)
      GO TO 120
112   Z=Z+X(1)
120   CONTINUE
      WRITE Y,Z
      STOP
      END
```

And let us suppose that we are able to derive by the meta-language the string:

```
          DIMENSION e1 (e2)
          READ e8
          DO e3  e5=e6,e6
          READ e15(e10)
    e7    CONTINUE
          e13=e10
          e13=e10
          IF(e15(e10)) e4,e4,e4
    e7    e13=e14+e15(e10)
          GOTO e4
    e7    e13=e14+e15(e10)
    e7    CONTINUE
          WRITE e12,e12
          STOP
          END
```

Where the elementary objects mean:

- e_1 **array in declaration**
- e_2 **integer number**
- e_3 **reference for a label in a DO instruction**
- e_4 **reference for a label in a jump instruction**
- e_5 **index variable of a DO cycle**
- e_6 **parameter of a DO cycle**
- e_7 **label**
- e_8 **integer variable**
- e_9 **integer variable which get value**
- e_{10} **integer value /variable or number/**
- e_{11} **integer array**
- e_{12} **real variable**
- e_{13} **real variable which get value**
- e_{14} **real value /variable or number/**
- e_{15} **real array**

The vocabulary V is formed as a pairlist, i.e. a list of sub-lists where the head /CAR/ of the sublists is an element and the tail /CDR/ is its attributes.

The attributes are:

VARI	variable
NUMB	number
INT	integer
REAL	real
ARRAY	array
CLOSED	may not use it in an active rol
DO	the label of a non complete DO cycle
EXIST	existing label

The vocabulary has a final state if all the labels in it are existing.

Figure 1 shows the TRANS function which is the vocabulary function defined in pure Lisp. Figure 2 shows the states of the vocabulary during the checking of the current program.

The program, has been executed by the R10 minicomputer in a 16K byte version of the Lisp interpreter.

Figure 1.

```
(DEFINE(QUOTE((TRANS (LAMBDA(E,X,V)
(COND
((EQ Q E1)(COND
((FIND X V) ERROR)
(I (CONS (LIST X (INT X),*ARRAY*) V))))
((EQ E E2)(COND
((AND (IN *INT**(GET X,V))(IN *NUMB*(GET X, V)))(UPDATE(GET X,V)V))
(T ERROR ) ))
((EQ E E3)(COND
((FIND X V) (COND
((IN *DO*(FIND X V) )(CONS(LIST X,*DO*)V))
(T ERROR) ))
(T (CONS(LIST X*DO*) V)) ))
((EQ E E4)(COND
((NOT (FIND X V))(CONS(LIST X) V))
(( IN *CLOSED* (FIND X V)) ERROR)
(T V) ))
((EQ E E5)(COND
((AND(AND(IN *VARI* (GET X,V))(lN *INT*(GET X,V)))
                  (NOT (IN *CLOSED* (GET X,V))))
(CONS(TAIL (CAR V)X)(UPDATE
(TAIL (GET X V) *CLOSED*)(CDR V))))
(T ERROR) ))
((EQ E E6)(COND
((IN *INT* (GET X V)) (COND
((IN *VARI* (GET X V))(CONS (TAIL(CAR V) X)(UPDATE(TAIL
           (GET X V)*CLOSED*)(CDR V))))
((IN *NUMB* (GET X V))(CONS(CAR V)(UPDATE(GET X,V)(CDR V))))
(T ERROR)))
(T ERROR)))
((EQ E E7)(COND
((NOT (FIND X,V))(CONS (LIST X,*EXIST*)V))
((IN *DO* (FIND X,V))(CLOSE X,V))
((IN *EXIST*(FIND X,V)) ERROR)
(T (CHEK X,V))))
((EQ E E8)(COND
((AND (IN *INT* (GET X V))(IN *VARI*(GET X V)))(UPDATE
           (GET X V)V))
(T ERROR)))
((EQ E E9) (COND
((AND(AND(IN *INT* (GET X,V))(IN *VARI*(GET X,V)))
           (NOT(IN *CLOSED*(GET X,V))))(UPDATE (GET X V)V))
(T ERROR)))
((EQ E E10)(COND
((AND(IN *TNT*(GET X,V))(NOT(IN *ARRAY*(GET X,V))))(UPDATE
                  (GET X V)V))
(T ERROR) ))
```

```
((EQ E E11)(COND
((AND (IN *INT* (FIND X V))(IN *ARRAY*(FIND X V))) V)
(T ERROR)))
((EQ E E12)(COND
((AND (IN *REAL* (GET X V))(IN *VARI*(GET X V))) (UPDATE
                  (GET X V)V))
(T ERROR)))
((EQ E E13)(COND
((AND(AND(IN *REAL*(GET X,V))(IN *VARI*(GET X,V)))
           (N0T(IN*CLOSED*(GET X,V))))(UPDATE (GET X V)V))
(T ERROR)))
((EQ E E14)(COND
((AND(IN *REAL*(GET X,V))(NOT(IN *ARRAY*(GET X,V)))) (UPDATE
                  (GET X V)V))
(T ERROR)))
((EQ E E15) (COND
((AND (IN *REAL*(FIND X V))(IN *ARRAY*(FIND X V))) V)
(T ERROR)))
(T ERROR2)
))))))
(DFINE(QUOTE(
(CLOSE (LAMBDA (X,V)(COND
((NOT (FIND X,V))V)
((EQ X(CAR(CAR V)))(OPEN(CDR(CDR(FIND X,V)))(CONS
                  (LIST X,*EXIST*,*CLOSED*)(CLOSE X (CDR V)))))
((IN *EXIST*(CAR V)) (CONS(TAIL (CAR V),*CLOSED*)
                  (CLOSE X (CDR V))))
(T(CONS(CAR V)(CLOSE X (CDR V)))) )))
(CHEK(LAMBDA (X,V)(COND
((IN *DO*(CAR V)) ERROR)
((EQ X (CAR(CAR V)))(CONS(LIST X,*EXIST*)(CDR V)))
(T(CONS(CAR V)(CHEK X,(CDR V)))) )))
(CURTAIL (LAMBDA (X) (COND
((EQ(CDR Y)NIL)NIL)
(T(CONS(CAR Y)(CURTAIL(CDR Y)))) )))
(OPEN(LAMBDA(Y,V)(COND
((NULL Y) V)
(T(OPEN(CDR Y) (UPDATE(CURTAIL(FIND(CAR Y) V)) V))) )))
(TAIL (LAMBDA (S,Y)(COND
((NULL S) (LIST Y) )
(T(CONS (CAR S)(TAIL(CDR S)Y))) )))
(GET (LAMDA (X,V)(COND
((FIND X V) (FIND X V))
(T (LIST X (INT X)(VARI X))))))
(IN(LAMBDA (P,L) (COND
((NULL L) NIL)
((EQ (CAR L) P) T)
(T(IN P (CDR L))))))
```

```
(UPDATE (LAMBDA (L V)(COND
((NOT(FIND (CAR L) V))(CONS L V))
(T (COND
 ((EQ(CAR L) (CAR(CAR V)))(CONS L (CDR V)))
 (T (CONS(CAR V) (UPDATE L (CDR V))))
    )) )))
)))
```

Figure 2.

```
V0 NIL

V1=TRANS[E1;X;V0]=

((X *REAL* *ARRAY*) )

V2=TRANS[E2;50;V1]=

((50 *INT* *NUMB*) (X *REAL* *ARRAY*))

V3=TRANS[E9;K;V2]=

((K *INT* *VARI*) (50 *INT* *NUMB*) (X *REAL* *ARRAY*))

V4=TRANS[E3;110L;V3]=

((110L *DO*)(K *INT**VARI*)(50 *INT* *NUMB* )(X *REAL* *ARRAY*))

V5=TRANS[E5;I;V4]=

((110L *DO* I) (I *INT* *VARI* *CLOSED*) (K *INT* *VARY*)
(50 *INT* *NUMB*) (X *REAL* *ARRAY*))

V6=TRANS[E6;1;V5]=

((110L *DO* I) (1 *INT* NUMB ) (I *INT* *VARI* *CLOSED*) (K *INT*
*VARI*) (50 *INT * *NUMB*) (X *REAL* *ARRAY*))

V7=TRANS[E6;K;V6]=

((110L *DO* I K)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT* *VARI*
*CLOSED *)( 50 *INT* *NUMB*)(X *REAL* *ARRAY*))

V8=TRANS[E15;X;V7]=

((110L *DO* I K)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT* *VARI*
*CLOSED*)(50 *INT* *NUMB*)(X *REAL* *ARRAY*))

V9=TRANS[E10;I;V8]=

((110L *DO* I K)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT* *VARI*
*CLOSED*)(50 *INT* *NUMB*)(X *REAL* *ARRAY*))

V10=TRANS[E7;110L;V9]=

((110L *EXIST* *CLOSED*)(1 *INT**NUMB*)(I *INT *VARI*)(K *INT*
*VARI*)(50 *INT**NUMB*)(X *REAL* *ARRAY*))

V11=TRANS[E13;Y;V10]=

((Y *REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT*
*VARI*)(K *INT* *VARI*)(50 *INT* *NUMB *)(X *REAL* *ARRAY*))
```

```
V12=TRANS[E10;0;V11]=

((0 *INT**NUMB*)(Y *REAL**VARY*)(110L *EXSIST**CLOSED*)(1 *INT*
*NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)(50 *INT* *NUMB*)(X*REAL*
*ARRAY*))

V13=TRANS[E13;Z;V12]=

((Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL* *VARI*)(llOL *EXSIST*
*CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)(50 *INT*
*NUMB*)(X *REAL* *ARRAY*))

V14=TRANS[E10;0;V13]=

((Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL* *VARI*)(llOL *EXSIST*
*CLOSED*)(1 *INT**NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)(50 *INT*
*NUMB*)(X *REAL* *ARRAY*))

V15=TRANS[E3;120L;V14]=

((120L *DO*)(Z *REAL* *VARI*) (0 *INT* *NUMB*)(Y *REAL* *VARI*) (
110L *EXSIST* *CLOSED*) (1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT*
*VARI*)(50 *INT* *NUMB*) (X *REAL* *ARRAY*))

V16=TRANS[E5;i;V15]=

((120L *DO* I)(Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL* *VARI*) (
110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT*
*VARI*)(50 *INT* *NUMB*) (X *REAL* *ARRAY*))

V17=TRANS[E6;1;V16]=

((120L *DO* I)(Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL* *VARI*)(
110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*) (I *INT* *VARI* *CLOSED*)(K
*INT* *VARI*)(50 *INT* *NUMB*)(X *REAL* *ARRAY*))

V18=TRANS[E6;K;V17]=

((120L *DO* I K)(Z *REAL* *VARI*)(0 *INT* *NUMB*) (Y *REAL* *VARI*)
(110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K
*INT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X *REAL* *ARRAY*))

V19=TRANS[E15;X;V18]=

((120L *DO* I K)(Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL* *VARI*)
(110L *EXSIST* *CLOSED*)(l *INT* *NUMB*)(I *INT* *VARI* *CLOSED) (K
*INT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X *REAL* *ARRAY*))

V20=TRANS[E10;I;V19]=

((120L *DO* I K)(Z *REAL* *VARI*) (0 *INT* *NUMB*)(Y  REAL* *VARI*)
(110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K
*INT* *VARI* *CLOSED*)(50 *INT**NUMB*)(X *REAL* *ARRAY*))
```

```
V21=TRANS[E4;111L;V20]=

((111L)(120L *DO* I K)(Z *REAL* *VARI*) (0 *INT* *NUMB*)(Y *REAL*
*VARI*)(l10L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(1 *INT* *VARI*
*CLOSED*)( K *INT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X *REAL*
ARRAY))

V22=TRANS[E4;120L;V21]=

((111L)(120L *DO* I K)(Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL*
*VARI*)(110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*
*CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X *REAL*
*ARRAY*))

V23=TRANS[E4;112L;V22]=

((112L)(111L)(120L *DO* I K)(Z *REAL* *VARI*)(0 *INT* *NUMB*) (Y
*REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT*
*VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X
*REAL* *ARRAY*))

V24=TRANS[E7;111L;V23]=

((112L)(111L*EXIST*)(120L *DO* I K)(Z *REAL* *VARI*)(0 *INT*
*NUMB*)(Y *REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I
*INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X
*REAL* *ARRAY*))

V25=TRANS[E13;Y;V24]=

((112L)(111L *EXIST*)(l20L *DO* I K)(Z *REAL* *VARI*)(0 *INT*
*NUMB*)(Y *REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1*INT* *NUMB*)(I
 *INT* *VARI* *CLOSED*)(K *INT* *VARI**CLOSED*)(50 *INT**NUMB*)(X
 *REAL* *ARRAY*))

V26=TRANS[E14;Y;V25]=

((112L)(111L *EXIST*) (120L *DO* I K)(Z *REAL* *VARI*)( 0 *INT*
*NUMB*)(Y *REAL* *VARI*)(110L *EXSISIST* *CLOSED*)(1 *INT**NUMB*)(I
*INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X
*REAL* *ARRAY*))

V27=TRANS[E15;X;V26]=

((112L)(111L *EXIST*)(120L *DO* I K)(Z *REAL* *VARI*)(0 *INT*
*NUMB*)(Y *REAL* *VARI*)(110L *EXSIST**CLOSED*)(1 *INT**NUMB*)(I
 *INT* *VARI* *CLOSED*)(K *NT* *VARI* *CLOSED*)(50 *TNT**NUMB*)(X
 *REAL* *ARRAY*))

V28=TRANS[E10;I;V27]=

((112L(111L *EXIST*)(120L *DO* I K)(Z *REAL* *VARI*)(0 *INT*
*NUMB*)(Y *REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I
*INT* *VARI* *CLOSED*)(K *TNT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X
*REAL*  ARRAY*))
```

```
V29=TRANS[E4;120L;V28]=

(112L)(111L *EXIST*)(120L *D0* I K)(Z *REAL* *VARI*)(0 *INT*
*NUMB*)(Y *REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I
*INT**VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT* *NUMB*)(X
*REAL* *ARRAY*))

V30=TRANS[E7;112L;V29]=

((112L *EXIST*)(111L *EXIST*)(120L *D0* I K)(Z *REAL* *VARI*)(0
*INT* *NUMB*)(Y *REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1 *INT*
*NUMB*)(I *INT* * VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT*
*NUMB*)(X *REAL* *ARRAY*))

V31=TRANS[E13;Z;V30]=

(112L *EXIST*)(111L *EXIST*)(120L *DO* I K)(Z *REAL* *VARI*)(0
*INT* *NUMB*)(Y *REAL* *VARI*)(110L *EXSIST* * CLOSED*)(1 *INT*
*NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT*
*NUMB*)(X *REAL* * ARRAY*))

V32=TRANS[E14;Z;V31]=

((112L *EXIST*)(111L *EXIST*)(120L *DO* I K)(Z *REAL* *VARI*)(0
*INT* *NUMB*)(Y *REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1 *INT*
*NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT*
*NUMB*)(X *REAL* *ARRAY*))

V33=TRANS[E15;X;V32]=

((112L *EXIST*)(111L *EXIST*)(120L *DO* I K)(Z *REAL* *VARI*)(0
*INT* *NUMB*)(Y *REAL* *VARI*)(110L *EXSIST* *CLOSED*)(1 *INT*
*NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT*
*NUMB*)(X *REAL* *ARRAY*))

V34=TRANS[E10;I;V33]=

((112L *EXIST*)(lllL *EXIST*)(120L *D0* I K)(Z *REAL* *VARI*)(0
*INT* *NUMB*)(Y *REAL* *VARI*)(110L *EXOST* *CLOSED*)(1 *INT*
*NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(50 *INT*
*NUMB*)(X *REAL* *ARRAY*))

V35=TRANS[E7;120L;V34]=

((112L *EXIST* *CLOSED*)(111L *EXIST* *CLOSED*)(120L *EXIST*
*CLOSED*)(Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL* *VARI*)(110L
*EXIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)
(50 *INT* *NUMB*)(X *REAL* *ARRAY*))

V36=TRANS[E12;Y;V35]=

((112L *EXIST* *CLOSED*)(111L *EXIST* *CLOSED*)(120L *EXIST*
*CLOSED*)(Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL* *VARI*)(110L
*EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)
(50 *INT* *NUMB*) (X *REAL* *ARRAY*))

V37=TRANS[E12;Z;V36]=

((112L *EXIST* *CLOSED*)(111L *EXIST* *CLOSED*)(l20L *EXIST*
*CLOSED*)(Z *REAL* *VARI*)(0 *INT* *NUMB*)(Y *REAL* *VARI*)(110L
*EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)
(50 *INT* *NUMB*)(X *REAL* *ARRAY*))
```