СООБЩЕНИЯ
ОБЪЕДИНЕННОГО
ИНСТИТУТА
ЯДЕРНЫХ
ИССЛЕДОВАНИЙ
ДУБНА

E10 - 10549

D.C.Marinescu

CDL AS A SYSTEM IMPLEMENTATION LANGUAGE IN SCIENTIFIC ENVIRONMENT.

Part II

1977

E10 - 10549

D.C.Marinescu

CDL AS A SYSTEM IMPLEMENTATION LANGUAGE IN SCIENTIFIC ENVIRONMENT.

Part II

Маринеску Д.К. **E10 - 10549**

Использование CDL в качестве языка описания и внедрения систем математического обеспечения в научных целях. Ч. II

В работе характеризуется специальный язык, который удобен для описания системных программ: трансляторов, программ - редакторов, мониторов и т.п. Приводится описание его основных синтаксических конструкций и особенностей использования. Транслятор с этого языка внедрен автором работы на ЭВМ CDC-6500 ОИЯИ, и в работе описаны способы обращения к данному транслятору и возможности использования языка на машинах других типов.

Работа выполнена в Лаборатории вычислительной техники и автоматизации ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна 1977

Marinescu D.C. **E10 - 10549**

CDL as a System Implementation Language in Scientific Environment. Part.II

The special language is characterized which is convenient for system program description: compilers, editors, monitors, supervisors, etc. The definition for its basic syntactical constructions and some notes about the possible usage are described. The version of the compiler for this language has been implemented by the author for the CDC-6500 at JINR, and this report contains the description for the access to this compiler and some possibilities of using this language on different computers.

The investigation has been performed at the Laboratory of Computing Techniques and Automation, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna 1977

The pertinent properties of CDL are:

1. CDL is a reccursive language. This feature is required in order to give to the language the power to describe well structured collections of objects. This implies that a stack to hold all quantities local to a rule must exist.
2. The language is highly machine and operating system independent. Machine independence results from the fact that CDL has been designed to be translated towards an abstract machine which can be easily implemented on practically every computer. On the other hand the interaction with the operating system (especially when performing I/O operations) takes place via external procedures which must be written (usually in a low level language) taking into account the particularities of the machine.
3. A CDL program achieves a tremendous simplicity: a running environment is created by the user which can define the basic operators he needs, by means of macro actions, flags and predicates. Thus, the CDL program is highly readable. Since there is no restriction in naming the different objects (variables,constants,procedures,etc.) in the computational space the name gives information about the function.

As stated previously, a high degree of flexibility results from the fact that CDL is translated towards an abstract machine; whenever a practical implementation of a CDL compiler-compiler is to be made, only the instructions of the abstract machine must be constructed, usually as a set of macro instructions to be executed by the processor under consideration.

The functions of the abstract machine and the corresponding abstract machine instructions are presented below.

- 1. To reserve space for the two types of data structures (lists and pointers) there are two instructions (abstract machine instructions) :

```
ZLISTDEC  P1,P2,P3
      P1  is the coded  name of the list
      P2  is the starting address
      P3  is the ending address.
ZVARDECL  P1
      P1  is the coded name of the variable.
```

For example the CDL program on the left is translated into the abstract machine instructions at the right :

```
'POINTER'  ALPHA .                         ZVARDECL   (GO)
'MACRO''POINTER' MINTEXT=100001,
                 MAXTEXT=100101.
'LIST' TEXT(MINTEXT:MAXTEXT).             ZLISTDEC   (G1),(100001),
                                                      (100101)
```

The COMPASS expansion of the macro instructions, from CDC-6500 implementation follows:

```
 ZVARDECL  MACRO P1
           USE   DATA
P1         BSS   1
           USE   0
           ENDM
```

```
 ZLISTDEC   MACRO  P1,P2,P3
            USE    LISTS
 P1         CON    P$LIST-P2
            CON    P2
            CON    P3
 P$LIST     SET    P$LIST+P3-P2+1
            USE    0
            ENDM
```


- 2. To reverse space for local labels and to provide jumps to them, either conditionally or unconditionally, there are the following abstract machine instructions :

```
ZLABDECL  P1,P2
ZJUMP     P1,P2
ZPOSJUMP  P1,P2
ZNEGJUMP  P1,P2
```

Here P1 stands for the sequence number (or the name of the label) inside the procedure body.

P2 stands for the coded name of the procedure inside which the label occured.

For example if in the rule with the coded name G15 appears a label declaration :

```
......ABC:.....
```

the corresponding abstract machine instruction is:

```
ZLABDECL     (ABC),(G15)
```

and a local label is generated: LABCG15 .

A request to jump to the previous label is written according to CDL syntax as :

```
.......... ,:ABC , ...
```

and it is translated as :

```
ZJUMP    (ABC),(G15)
```

The conditional jump instructions depend upon the state of a condition code.

The COMPASS expansion of the corresponding macros is:

```
ZLABDECL  MACRO  P1,P2
L→P1→P2   BSS    0
          ENDM
```

```
ZJUMP     MACRO  P1,P2
          EQ     L→P1→P2
          ENDM
```

```
ZNEGJUMP    MACRO   PL,P2
+           EQ      L↱P1↱P2
+           BSS     0
            ENDM
```

```
ZPOSJUMP    MACRO   P1,P2
+           EQ      *+2
+           EQ      L↱P1↱P2
            ENDM
```

- 3. The abstract machine must be provided with a condition code switch which can be set and reset. The instructions to do that are :

```
ZRETURNT
ZRETURNF
```

- 4. The language allows the user to define flags, i.e., switches and the abstract machine must be able to test such flags. The instruction to do that is :

```
ZTEST  P1
```

P1 stands for the coded name of the flag.

A flag has an address reserved (with a ZVARDECL instruction refering to the coded name of the flag,for example G28) and depending upon a preestablished convention some positive value stands for 'true' and some negative stands for 'false'. ZTEST simply tests if the content of the address is positive or not.

- 5. The abstract machine must be able to branch to a certain rule and to return from it. The instructions to perform such operations are :

```
ZCALL  ADDR,LINE
ZRETURN
```

Here ADDR is the coded name of the rule.

LINE is the line number of the source CDL program on which the call occured.

We are now in the position to understand the differences between the two basic rules available in CDL, the action and the predicate.

Let A be an action with two alternatives , A1 and A2.

```
'ACTION'  A.
A = A1 ; A2 .
```

This might be translated as:

```
ZBLOPEN   (G13),(2),(6),(123)
ZLOBND    (2),(3)
-----------------------------------
body of the first alternative ,A1
-----------------------------------
ZJUMP     (999),(G13)
ZLABDECL  (1),(G13)
-----------------------------------
body of the second alternative ,A2
-----------------------------------
ZLABDECL  (999),(G13)
ZRETURN
```

The body of the first alternative is so expanded, that if it fails, the code of the second alternative is executed. If A1 succeeds then a jump to the end of the rule is performed. If the second alternative also fails then a diagnostic informs : 'may be false action , actionname ' .

If we do not inform the CDL compiler-compiler that A is to be treated as an action , then by default, it is considered a predicate and translated as :

```
ZBLOPEN   (G13),(2),(6),(123)
ZLOBND    (2),(3)
-----------------------------------
body of the first alternative, A1
-----------------------------------
ZJUMP     (999),(G13)
ZLABDECL  (1),(G13)
-----------------------------------
body of the second alternative, A2
-----------------------------------
ZLABDECL  (999),(G13)
ZRETURNT
ZLABDECL  (2),(G13)
ZRETURNF
```

Here if the first alternative succeeds a branch to the label L999G13 occures and the rule returns 'TRUE' . The same thing happens if the first alternative fails but the second succeeds. But if the second fails too then the rule returns 'FALSE'.

To be more explicit we shall give an example:

```
'PREDICATE' ANDREI.
ANDREI+X+Y:
     EQUAL+X+0,MAKE+Y+10;
     LESS+X+0 ,MAKE+Y+100.
```

In this example EQUAL and LESS are flags and MAKE is an action; all of them are declared as system macros. The translation is:

```
ZBLOPEN   (G13),(2),(3),(148)
ZLOBND    (2),(3)       -----------
EQUAL     (X2),(=0)
ZNEGJUMP  (1),(G13)
ZLOBND    (3),(4)                  body of the first alternative,
                                     EQUAL+X+0,MAKE+Y+10
MAKE      (X3),(=10)----------
ZJUMP     (999),(G13)
ZLABDECL  (1),(G13)     -----------
LESS      (X4),(=0)
ZNEGJUMP  (2),(G13)                body of the second alternative
                                     LESS+X+0,MAKE+Y+100
ZLOBND    (5),(6)
MAKE      (X5),(=100)----------
ZLABDECL  (999),(G13)
ZRETURNT
ZLABDECL  (2),(G13)
ZRETURNF
```

It is now transparent that each alternative of a predicate must contain either another predicate or a flag. In this example the flag EQUAL leads to a sequence containing the expansion of the macro EQUAL followed by a conditional jump to the end of the first alternative. This illustrates the idea that if a member of an alternative fails then the whole alternative fails.

On the other hand, for an action each alternative but the last must be capable of failing (must contain either a flag or a predicate as a member).

Since a rule is translated in different ways depending upon its type (action or predicate) it results that when defining a rule we must use only other rules which have been previously defined.

Received by Publishing Department
on April 4, 1977.