Janusz MADEJSKI

The Institute of Metal Science
The Silesian Technical University, Gliwice, Poland

CONCEPT OF THE OFF-LINE TEXTUAL PROGRAMMING SYSTEM FOR THE INDUSTRIAL ROBOTS

Summary. The paper presents the concept of a system for an off-line textual programming of industrial robots. The main assumption justifying the project is the fact that nearly 50% of labour consumption comes from the program design, mostly of its logical structure. A brief outline of a proposed Pascal-like programming language is presented along with some considerations how to implement the idea of its translator in PROLOG. The output of the translator are the ASCII text files containing the labelled robot instructions with relevant comments. Possible labour time savings of the robot program design, due to the programming system in question, are likely to reach 80-90% according to author's job-shop experience.

1. Introduction

The textual off-line programming of robots is a widely recognized method of cutting the down time of the robotized manufacturing cells. Thanks to the fact that the most time consuming tasks may be completed in advance and outside of the manufacturing system the time necessary to changeover from one production batch to the next one may be decreased. Many robot programming languages emerged from the real-life needs [1÷3,6] that have helped to solve the task of the robot program generation. The author has noticed a significant gap in the non-complicated programming tools'group that would assist the robot programmer in a proper design of the program. Figure 1 shows the structure of time consumption of a chain of tasks resulting in development of a debugged and running robot program albeit not always properly designed. The job-shop practice shows that the the most time consuming tasks are: designing of the program and its subsequent modifications. The most of the significant design problems arise when the programmer has to implement the flexible algorithms enabling the robot to fill the system with raw workpieces, tend the machine

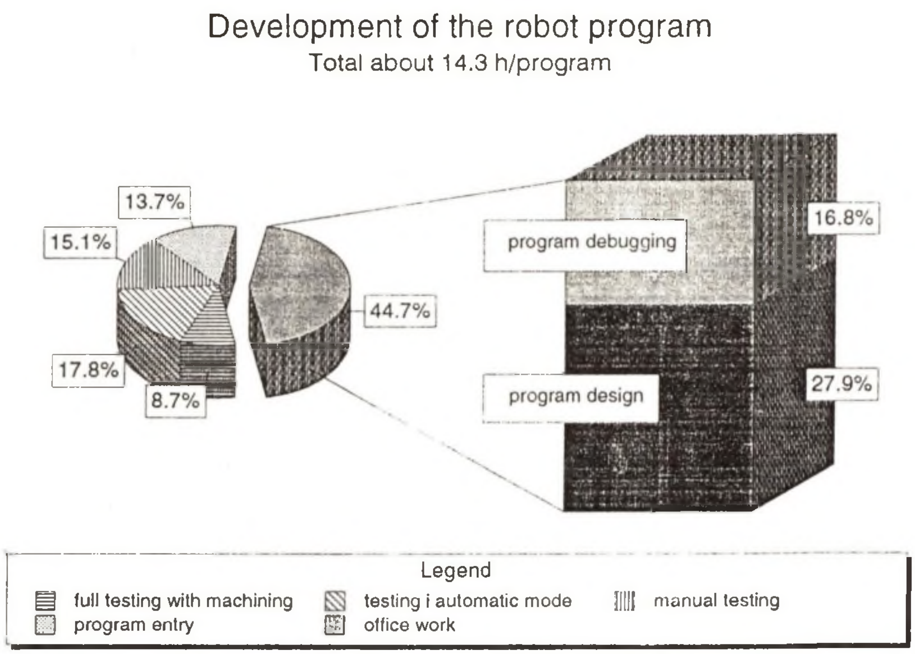

Figure 1. Structure of the average time necessary to develop a fully functional robot program for machine tending (data from 5 [11] robot programs developed an run in industry in a turning cell)

tools and other technological equipment. Properly designed robot program should first of all be able to respond to the results of quality inspection of workpieces on various stages of the technological process, it should also incorporate the capability to shut down the production in the cell when necessary. The actual input of the robot program into its memory is not a complicated task, and provided there are no bugs in it, featuring rather straightforward instruction input to the robot numerical control.

Figure 2 illustrates the attitude taken towards the off-line programming of robot. It has been decided that the designed system's tasks should include only those that constitute nearly 45% of the time consumption. The expression "time consumption" does include more than 15% of the system down time during the inevitable program flow

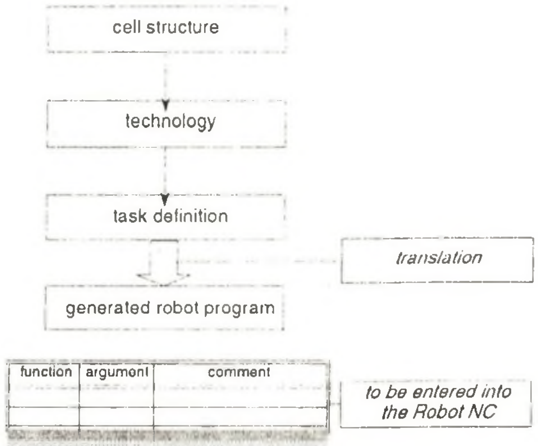

Figure 2. Robot program generation path

modifications. This time may be also cut down provided the program logical structure reflects the system job flow right from the very beginning. This may be achieved when the programmer is equipped with a relatively high level language enabling him to easily define the robot tasks in the cell and producing a listing of the crucial robot program fragments in detail. According to the author's experience the attempt to produce complete robot program including exact positioning of the robot's arm shall result in more time consumption in refining the actual positions than would be necessary to teach-in them *in situ*.

2. Analysis of the necessary robot programming language instructions

It has been found [4,5,10] that it is convenient to describe - and subsequently follow up - the robot program by means of the graphs (see Fig.3). What has been lacking was the convenient way of coding this logical flow of robot's task in the cell. Analysis of existing solutions was carried out [1,3,6,8] and the decision was made to design a language that would enable the programmer to describe the robot task and would give results reflecting the necessary robot program flow basing on its instruction set. ASEA IRb robot was chosen as an example as it is still fairly popular and its instruction set is quite comprehensive.

The mostly used robot program modules consist of series of instructions intended to check the status of a number of input circuits (including memory flags), setting output signals' status, manipulation with the grippers and workpieces in the robot's working enevelope, movement of the robot's arm, etc. The robot programmer has to define its job in the system by means of the robot instruction set. However these instructions are rather awkward to use especially when implementing the non-trivial programs including the analyses calling for multiple logical functions like IF ... THEN ... ELSE. Taking this into consideration and following the ideas in [7,9] the definition of the robot task definition language has been proposed. This language consists of simple commands specifying the robot's tasks' details more generally, and what is important, the programmer is free from careful instruction number/label checking that result often in hard to debug programming errors. Using the instruction set suggested in Table 1 fragment of the robot task description presented in a graph form in Fig.3 is given below:

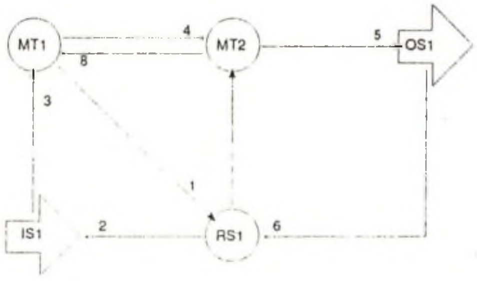

Figure 3. Example of simple robot program graph. Symbols MT1, MT2 (machine tools), IS1, OS2 (input/output storages) and RS1 (reorientation stand) denote the system components.

```
MT1_status :=  1 // declaration of numeric constants pointing to the input
IS1_status := 12 // circuits' IDs
MT1_chuck  :=  2
Operator_interrupt := 9
```

```
while Operator_interrupt = 0 do # do until the operator stops the robot

   begin

   watch MT1_status, 0 # wait until the machining operation MT1 is over
   move_to 4, go to MT1 # take the workpiece after the 1st operation
   take 0, 1.5 # grasp the workpiece and free it from the chuck
   output MT1_chuck, on, 20 # open the MT1 chuck
   move_to 4 # go to the neutral point outside of MT1
   move_to 3 # go to the reorientation stand
   put 0, 1.5 # leave the workpiece for further reorientation
   move_to 2 # go to the input storage IS1
   watch IS1_status, 1 # check if there are workpieces in the storage
   ...

   end
```

Part of the above robot program fragment would be translated to the following fully commented robot code (bolded items feature the actual robot program text, instruction labels are chosen arbitrarily as an example only):

```
 910 TEST JUMP 9  do until the operator stops the robot
 920 JUMP 2000
 930 TEST WAIT 1  wait until the machining operation MT1 is over
 940 PTPC 6       take the workpiece after the 1st operation
 950 PTPC 6
 960 PTPC 4
 970 PTPF 4
 980 GRIPPER 1.5  grasp the workpiece and free it from the chuck
 990 OUTPUT ON 2  open the MT1 chuck
1000 WAIT 20
1010 OUTPUT OFF 2
1020 PTPC 4       go to the neutral point outside of MT1
1030 PTPC 6
1040 PTPC 6
1050 PTPC 6
1060 PTPC 6       go to the reorientation stand
1070 PTPC 6
1080 PTPF 4
...
... etc.
```

The key time savings result in the efficient translation of the logical interdependencies among the several robot tasks (see detailed robot program algorithm in Fig.4):

```
MT1_service := 22  // declaration of numeric constants pointing to the input
MT2_service := 21  // circuits' ID
MT1_operational := 26
...
move_to 1 # withdraw to the neutral position next to MT1

  if MT1_service
          then  # MT1 tending procedure
                begin
                ...
                end
  if MT2_service move_to 1 # adjust arm position
  if MT2_operational
          then  # MT2 tending procedure
                begin
                ...
                end
          else ...
```

```
1470 PTPC 4               1474 TEST JUMP 21         1478 TEST JUMP 26
1471 TEST JUMP 22         1475 JUMP 1477            1479 JUMP 1680
1472 JUMP 1474            1476 JUMP 1590            1480 JUMP 1410
1473 JUMP 1480            1477 PTPC 4               etc...
```

note:

21, 22 and 26 are the robot memory flags employed to store intermediate system status data

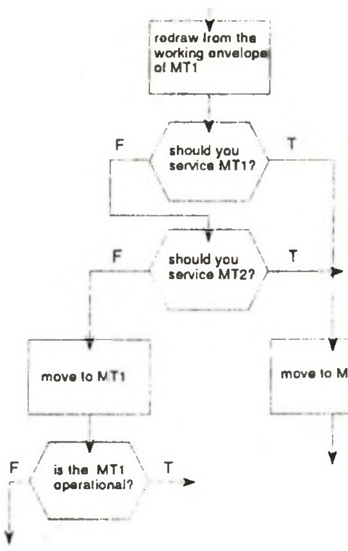

Figure 4. Detailed fragment of the flexible robot program algorithm - note that MT1 and MT2 may be used by the robot as needed, switching them *On* and *Off* is done automatically depending on the program execution stage

3. Programming language definition

The detailed analysis of many robot programs (machine tools' tending was taken into consideration only) has resulted with a proposal of a set of instructions presented in Table 1. Definitions of some exemplary instructions are given further in a Backus-Naur form. The instructions' set may be extended to as needed - nevertheless its most commonly used elements are included. Fig. 6 presents a listing of one of the PROLOG predicates - solution similar to the one published in [9] - being one of the crucial tranlator's elements. The overall translator program flow is as follows: text file containing the robot program description created using the proposed language is being processed by the translator written in PROLOG. Processing includes three main stages: lexical analysis, syntactic analysis and generation of the text file. This text file is in fact a programming instruction for the robot programmer. This instruction contains all details of the logical structure of the robot program plus indications, along with relevant comments, as to where should be introduced robot positioning instructions.

Table 1 Language instructions set

Instruction	Meaning	Example
`input`	*checking status of the input circuit or memory flag*	`input MT1_status`
`output`	*setting status of the output circuit or memory flag, time span of activation*	`output MT2_chuck_activate`
`take`	*predefined sequence of robot instructions, requires two parameters: output ID and delay time*	`take 0, 1.5`
`put`	*predefined sequence of robot instructions, requires two parameters: output ID and delay time*	`put 0,2.0`
`move_to`	*predefined sequence of robot instructions, requires one parameter: number of intermediate points on the trajectory polyline*	`move_to 3`
`watch`	*testing of a condition, results in halting the robot until the condition evaluates true, requires two parameters: input ID and following delay time if needed*	`watch MT1_status, 1`
`delay`	*pausing the execution of robot program for a set time interval*	`delay 2`
`if ... then ... else ...`	*basic logical function employed to evaluate the system status*	`if IS1_full then take 0, 1 else watch IS_full, 1`
`while ... do`	*continuous loop carried out provided some logical condition is met*	`while OS1_full do watch OS1_full, 2`
`begin ... end`	*denote program modules*	

The exemplary instruction definitions (below) may be easily translated by the relevant PROLOG predicates - each of the instructions may be followed by a comment separated from the instruction by "#":

```
<instruction> ::= <name> := <expression> |
                  input<name> |
                  output<name,name> |
                  move_to<name> |
                  ...
                  if<condition> then <instruction> else <instruction> |
                  ...

<condition> ::= <expression> <comparison> ::= <expression>
```

```
<comparison> ::= > | < | = | <>
etc.
```

Exemplary PROLOG predicate beginning implementing the "if" instruction is as follows:

```
DOMAINS
...
data = reference symbol*
...
instruction = ... if_(expression,instruction,instruction)...

PREDICATES
....
expression(integer,data,data,expression)
instruction(data,data,instruction)
instruction_coding(istruction,Robot_instructions_list,integer,integer)
...

CLAUSES
...
instruction(["if"|Symbols_list],Tail,if_(Condition,Then,Else))
  :- condition(Symbols_list,["then"|Tail1],Condition),
     instruction(Tail1,["else"|Tail2],Then),
     instruction(Tail2,Tail,Else),!.
```

4. Conluding remarks

The abovementioned conception of a textual robot programming language may be extended by a simple text editor and file manager to create a fully functional robot programming system. The debugging facilities are still to be designed as the input program may not be bug-free so the resulting robot program generated would neither be correct. It was a definite design decision not to try to employ any real world scene numerical description data in the resulting robot program text. What may be saved is the labour consumption at the design stage and at its further modification. Teaching the robot the exact positions in the real scene has been left to the operator intentionally. The proposed language may be now easily fully implemented according to its brief outline presented.

REFERENCES

[1] C. Blume, W. Jakob - "Programming Languages for Industrial Robots", Springer-Verlag, Würzburg, 1986, pp 31-193

[2] M. Groover - "Automation, Production Systems and Computer Integrated Manufacturing", Prentice-Hall International, Inc., pp.319-337

[3] G. Kost - Programowanie robotów przemysłowych IRb wspomagane komputerowo. III Krajowa konferencja robotyki, Wrocław 1990, v.1, pp.77÷83.

[4] M. Lee - "Intelligent Robotics", John Wiley & Sons, New York-Toronto, 1989, pp 142-147

[5] J. Madejski, R. Zdanowicz - "Computer Supervisory Control of Small Robotized Manufacturing System", Proceedings of the International Conference on Computer Integrated Manufacturing CIM'92, Zakopane, March 24-27 1992, pp 243-251

[6] J. Poblet - "Sistemas CAD/CAM/CAE. Diseño y fabricacion por computador", Marcombo Boixareu Editores, Barcelona-Mexico, 1986, pp 259-302

[7] H. Schildt - "Advanced Turbo Prolog™: Version 1.1", Osborne McGraw-Hill, Berkeley, 1987, pp 167-198

[8] B. Skolud, G. Kost - "Computer Aided Detection of Solids Collision in FMS with Robot by CAD Simulation", Proceedings of the International Conference on Computer Integrated Manufacturing CIM'92, Zakopane, March 24-27 1992, pp 359-366

[9] J. Szajna, M. Adamski, T. Kozłowski - "Turbo Prolog. Programowanie w języku logiki", WNT W-wa 1991, pp 108-130

[10] J. Wójcikowski, J.Madejski, R.Zdanowicz - "Zrobotyzowane gniazdo tokarskie o strukturze elastycznej w ZBMD - Zabrze", "Nowoczesne Technologie w Fabrykach Maszyn Górniczych" 1/15, Gliwice 1987, pp 1-7

Revised by: Jan Szadkowski