PL ISSN 0209-1593

1
'86

prace naukowo-badawcze

Instytutu Maszyn Matematycznych

rok XXVIII

Druk IMM zam. 3/87 nakł. 250 egz.

I n s t y t u t M a s z y n M a t e m a t y c z n y c h

p r a c e n a u k o w o - b a d a w c z e
I n s t y t u t u
M a s z y n
M a t e m a t y c z n y c h

Warszawa 1986

Adres redakcyjny: Instytut Maszyn Matematycznych
Branżowy Ośrodek INTE
02-078 Warszawa, ul. Krzywickiego 34
tel. 28-37-29 lub 21-84-41 w. 244

PL ISSN 0209-1593

Prace naukowo-badawcze Instytutu Maszyn Matematycznych

A model of complex job control computer systems

by Zdzisław WRZESZCZ

Abstract

Principal notion in building the model of complex job control (CJC) systems is based on matching the structure of a computer control means to the structure of controlled activity despite the field the activities come from. For these reasons we started with an abstract form called operation, a primitive element of activity. The operation and a concept of time orderring are taken as constituents for structures such as simple activity and complex activity. Then an abstract operation device called OpDevice is introduced. Using the OpDevice construction we obtain sort of bridge towards sequential process i.e, the control form of operation, thus the control form of activity. Hence the sequential process is a representation of a simple activity (simple job) and complex activity is nothing more but a cooperation of sequential processes. The definition of a CJC system behavior may then be done using appropriate computer language. As the CJC language predecessor the CSP concept of Hoare is taken. Several examples of complex activities are presented by means of the CJC language. The examples are chosen so as to show up the system management constitution. A complex system contains:

- CJ process - a coordinated bundle of JobProcesses,
- IP process-an interface between the CJ process and the USER,
- USER-a human manager of the system,
- DSC-an AI DBS type controller for consecutive actions of the CJ process.

Short presentation of these systems constructions is given. The system model design relies on a philosophy according to which the CJ processes are responsible for the routine tasks and the intelligent, supervisory part of control is dedicated to USER equipped with a DSC.

1. INTRODUCTION

Computer automation of complex human activities is a problem which has been absorbing the attention of computer scientists and engineers a long time. Here the computer is treated as the means to integrate complex activities. Main part of the research is observed in automation of manufacturing [14], [31]. Other examples come from design engineering [3], office activities [21], [29] general medicine practice [35], intensive therapy [19] to quote only some prominent examples.

We are assured that the computer integrated automation may step almost into every field of human endeavor; it may free the human from doing routine jobs, thus allowing more time for real creative occupation. Owing to the constant progress in computer technologies, especially in VLSI technology [37], we are able to obtain the computer parts (processors, i/o devices etc) in a form of small dimension modules with complete function repertoir. Such parts may be built into an operation device. Local area networks are technical means for communication among dispersed processors [28].

Principal notion in building a model of complex job control systems is based on matching the structure of computer processes to the complex activity structure. Such an approach allows for treating the problems of activity management in the domain of sequential computing processes. This delivers us the possibility to formulate problems of complex job control in a homogenic way despite the diversity of fields from which the jobs come.

From the methodological point of view our model is an abstract representation of real object behaviours. The model encompasses such elements, properties and relations which are important to a purposeful behaviour of entities and animated devices. Thus the analogy between the modelled object and the model has the character of functional isomorphism [38].

2. SIMPLE AND COMPLEX ACTIVITY STRUCTURES

Establishing the structure of a complex activity is a matter of decision made by the activity realization organizer. Simple activity (simple job) has been the subject of study since XVIII-th century. Immanuel Kant describing the human activity [1] distinguishes a behaviour factor calling it a general imperative of activity. According to Kant such imperatives appear as a practical consequence of science and the imperatives dictate the proper way how to do, say a glass container which will not break despite pouring the hot water in it. The particular way of doing such a container is the consequence of knowledge in chemistry and mechanics.

Modern technology allows to describe precisely the operations necessary to obtain different sort of tasks in the field of human activity. The operations are the most primitive entities of an activity organization. To discuss the nature of operations we assume that an operation is performed by an actor which may be either a device or a human operator. The operations are purposeful acts; the purpose of any operation performance is to influence and estimate the behaviour status of defined object in the world under observation (world W). The object status is defined by an established set of characteristics of the object. The status change due to an operation realisation takes form of a time function, possibly - many valued time function. So when we say of observing an operation OPi we in fact observe either the status change of a defined point (area) of the world W or the operation actor status change as the actor itself is also a member of the world W.

The performances of operations (time dependent status changes) are ordered in time so as to build simple activities and these in turn compose time structures called complex activities (complex jobs). To present the simple and complex activities in a more convincing (quantitative) way we introduce the following definitions.

OPi - the name of an operation;
sOPi - OPi operation status change in time;
- a representation of OPi behaviour when the operation is realized;

§sOPi - an event of the beginning change of the operation OPi status;

sOPi§ - an event of the end change of the operation OPi status;

t§sOPi, tsOPi§ - the time values for which the events §sOPi, sOPi§ appear; the time values are real numbers.

To establish the time precedence relation between two operations OPi, OPj

$$sOPi \longrightarrow sOPj \qquad (2.1)$$

the following clause must be true

$$(t\S sOPi < tsOPi\S < t\S sOPj < tsOPj\S)$$

The operation performance and times are difficult to predict, and also the operation performance duration =tsOPi§-t§sOPi may in principle, acquire different values for different realizations.

- Performances of two operations OPi, OPj are concurrent

$$sOP \;||\; sOPj \qquad (2.2)$$

if and only if

$$(\neg(sOPi \rightarrow sOPj)) \wedge (\neg(sOPj \rightarrow sOPi))$$

Using the above formulae we can describe a simple activity J composed of operations taken from a set $OP = (OP_\alpha, \ldots, OP_\xi)$ if we establish for every pair of operations the time precedence relation (2.1). The simple activity J has then a first operation realization sOP_α defined by, say, OP_α, a second operation realization sOP_β etc, in relation 2.1.

- two simple activities J1, J2 are concurrent

$$J1 \;||\; J2 \qquad (2.3)$$

if and only if there exists $sOP1 \in J1$ and $sOP2 \in J2$ such that $sOP1 \;||\; sOP2$.

- Complex activity CJ is a set $(J1, \ldots, JM)$ of concurrent simple activities denoted as

$$J1 \;||\; J2 \;||\; \ldots \;||\; JM \qquad (2.4)$$

which means that $J1 \;||\; Jj$ for each $1 < i,j < M$.

Up to the moment we can say not to much about concurrency among the operations next to the first operation performance. The simple jobs in a concurrent activity may start together but they may proceede with their own time rate provided there are no other restriotions. The restrictions may appear as a consequence of communication among the simple jobs in the complex. This is the problem we intend to present more thouroghly in the following chapters.

Simple construots, such as the operation, must be managed by simple means. We assume that, despite their highly differentiated tasks the operations have the same primitive form dictated by structure composition demands. Assuming an automated performance of an operation we assign it the following attributes;

1. Every operation is performed due to a special actor dedicated to the operation type.
2. Operation performance may be commenced only in response to a signal InSgnl coming from outside the operation actor.
3. When an operation performance is ended an output signal OutSgnl is generated by the operation actor.
4. Two operations OPi, OPj are in time precedence retation if the signal InSgnl is generated to OPj after receiving the OutSgnl from the operation Opi.

The above four simple attributes are sufficient to establish necessary activity structures i.e, simple and complex jobs. A mechanism must be formulated to receive the signals InSgnl and to generate the signals OutSgnl inside every simple activity and among the activities.

The described concept of time ordering allows for running a complex activity in a fully asynchronous way. The courses of activities depend completely on courses of their components - operations and the courses of operations may depend on conditions being external to the control. Thus the accepted approach allows for

- running an activity according to its natural stimuli,
- elimination of clocks in every control process and the problem of synchronization.

Suppose the operation task is to draw a line 5 units long, laid at 45° to the ax +X. So the signal InSgnl received by the plotter should carry the following values

x q y q q q q q x z

```
OpProcedure  LINE  (X1, Y2, X2, Y2: INTEGER);

(*  Declaration part *)

    var
        Tvar,A,B,x,y:INTEGER;
        ---------------------
        ---------------------

(*  Computation part *)

    begin
        Tvar:= 2 * ( y2-y1) -  (x2-X1):
        A:= 2 *  (y2-y1):
        B:= 2 *  (y2-y1) - (x2+x1):
        y:= y1:

*       Effector step computation *

         for   x:= x1+1  to  x2-1  do

               begin

                  if (Tvar <Ø) then   Tvar:= Tvar+A
                  else  Tvar:= Tvar+B ;
                  y:= y+1 ;

(*      Effector step driving part *)
                  InSgnl ( x,y)
                  OutSgnl

               end:

    end:

( * End of operation *)
```

Figure 3.2 Operation procedure example for drawing LINE of arbitrary length and inclination (the computation part is based on Bresenham algorithm)

It is easy to see that drawing more complicated figures than the above LINE makes programming of operations a difficult and prone to errors task. Solution of the problem is delivered by modern programming languages [23] in a from of procedure which may be a prototype for operation procedure - a program module to control the operation device behaviour.

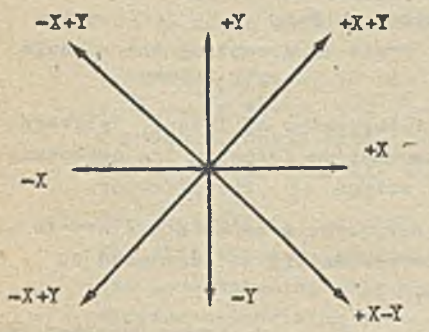
The operation procedure module consists of three parts:

- a declaration part defining variables, constants, types etc, but also the types of components of InSgnl and OutSgnl are defined,
- a computation part to calculate parameters for the consecutive effector steps,
- a control part from which the consecutive signals are sent to the operation device (InSgnl) and received from the device (OutSgnl).

Such a construction we call OpProcedure and an example of it, pertaining the plotter, is presented in Figure 3.2.

3. OPERATION DEVICE MODEL

Presenting the nature of simple and complex activities we have pointed the operation as a basic structure element. Devices realizing the operations should have attributes enumerated at the end of the previous chapter. Looking for practical examples one may observe:



/3.1a/

Command name - InSgnl content		Command representation in ASCII code
pen move in	+Y	P
" "	+X+Y	q
" "	+X	r
" "	+X-Y	S
" "	-Y	t
" "	-X-Y	u
" "	-X	V
" "	-X+Y	w
no pen driving		X
pen driving		y
waiting		z

/3.1b/

Figure 3.1. Commands codes for a plotter effector

- in industral manufacturing processes: different sort of numerical control lathing machines, assembly manipulators, quality status monitors etc., [27] , [24] ,
- in automated design (so called CAD-computer aided design) we meet picture input devices (different sort of digital coders), picture output devices (graphic monitors, terminals, plotters) etc., [3] ,

to mention only two fields of the human occupation.

In every automatic system aside to the special problem oriented devices, there must be standard computer modules e.g., processors, external memories, general use i/o devices, [16] communication network modules etc., which may be included as operation devices if an activity analysis dictates such an inclusion.

Despite the rather large number of operation types and variety of constructions, the operation devices have the following common features: the operation devices

1 are controlled by means of digital sequences (signals),

2 perform a repertoir of elementary actions,

3 compose an operation from elementary actions, according to a defined program.

To illustrate the operation device behaviour let us consider plotting devices e.g., xy-plotter or CRT graphic monitor. The elementary action effector in such a device is a motor driven drawing pen (an electron gun in monitors). The elementary actions form sort of dashes which may be drawn in several directions, see Figure 3.1a. The directed dashes may be programmed by means of simple tray of codes shown in Figure 3.1b.

A processor for storing and realizing the operation procedures is introduced as a separate system module. Such a processor module (JProcessor) is responsible for the complete simple job performance; we shall discuss this problem in chapter 8.

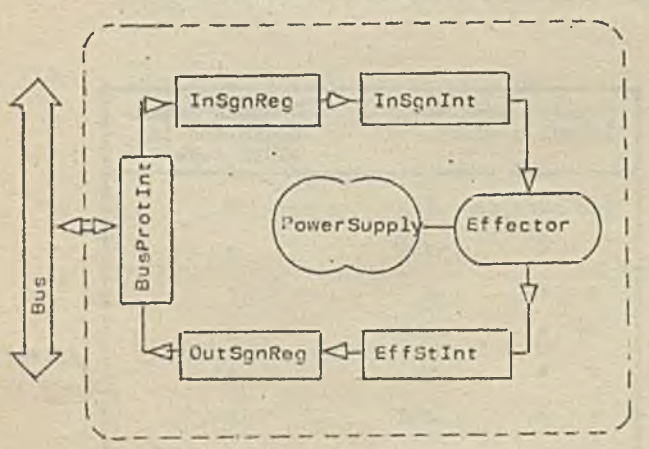


Figure 3.3 The structure of OpDevice

In Figure 3.3 is presented an abstract operation device structure (OpDevice structure), elements of the structure have the following functions.

- Bus ProtInt: protocol interpretation of signals InSgnl coming from the JProcessor and signals OutSgnl sent to JProcessor via the communication Bus,
- InSgnlReg: register accepting the signals InSgnl,
- InSgnlnt: interpreter of InSgnl, delivers a set of parameters defining the necessary elementary action of the Effector.
- Effector: device/or a human operator/ performing the elementary act demanded by JProcessor,
- Power-Supply: source of energy supplied to the Effector,
- EffStInt: interpreter of Effector status and generator of the signal OutSgnl,
- OutSgnReg: register for the signal OutSgnl, sends the signal to BusProtInt,
- Bus communication means linking the JProcessor with its OpDevices, the communication principle is established by means of particular BusProtocol [20] .

An operation device, with a structure like the above OpDevice, connected to a JProcessor, memory of which contains operation procedures (OpProcedures), proffers implementation means for a control command which in turn is an elementary part of a sequential process. A program of such a sequential process may be written as a sequence of command language expressions among which the operation command names e.g., AA (X1,...), BB (Y1,...) take over a prominent place in the process program:

JOBN:: ... <expr1> ; <expr2> ;...; AA (X1,...); AA (X1,...) ; BB (Y1,...) ; .

The concept of process as means for decomposing complex systems has been absorbing the attention of many authors [12, 6],[10, 30, 12] , for quite a long time. Thus it seems to be natural that the process concept has been obtaining different definitions. For our purposes we accept a process definition borrowed from Andrews [1] : "A sequential program specifies sequential execution of a list of statements; its execution is called a process". To the above we add: Execution of statements is done by a processor; a real processor in real world system or an abstract processor when it belongs to a model".

4. ELEMENTS OF THE COMPLEX JOB CONTROL LANGUAGE

The are many concepts of sequential process communication. A good review of the problem is given by Andrews and Schneider [1] . Choosing an interprocess communication model we have to remember that the simple jobs may be disposed in an area e.g., a factory. Hence the JobProcesses may be subdued to the same displacement. Owing to the tremendous progress in microelectronics [37] it becomes possible to deliver separate processors microprocessors for every important system process e.g., for every JobProcess, altho gh in practical sit ations some JobProcesses may share one physical processor.

4.1 Communication principle

For the communication among JobProcesses and for their course synchronization we accept a mechanism called message passing as the mechanism matches our time ordering concept for operations and activities. At present the message passing mechanism is used in many languages e.g., PLITS [11] , ADA [36], SR [2] , but the first thoroughly formulated concept belongs to Hoare [15] ;

The advantage of using the message passing as an interprocess communication vehicle becomes evident if we notice that the signals InSgnl, OutSgnl in the OpDevice have exactly the character of a message and the OpDevice is in fact a device process [18,25] .

Here we present a short outline of Hoare' s communicating sequential process system [15], modified so as to suit the demans of our problem.

I A complex process CJ controlling a complex activity, is composed of sequential processes J1,..., Jn where each Ji (i=1,...n) controls a simple activity. The sequential processes run concurrently after their initiation by means of a parallel command. The process CJ ends its existence when all the processes Ji are terminated.

II The sequential processes J1,... Jn do not communicate by means of global variables [1] . For this purpose input and output commands are used. Two concurrent processes Js and Jr are communicating only when the output command located in one of the processes, say Js, defines a receiving process Jr and an input command in Jr names Js as the sending process.

III The receiving process Jr stops its course on its input command as long as the associated sending process Js reaches its output command. In such a time Js sends its message directed to Jr and received here by the input command.

IV Guarded commands [9] can be used as a selecting mechanism in a concurrent bundle of processes. In the guards input commands can appear. A guard is selected if and when the sending process named in the guard's input command is ready. When several guards heve ready senders, then only one is selected and the choice is arbitrary. This shows us how indeterminism is signed in a complex process behaviour.

V A repetitive command may encompass a single process but also a bundle of concurrent processes ray be subdued to repetition. The processes may contain input commands. Such a concurrent process runs over its component processes as long as either

- all the component processes terminate or
- a distinguished component process fires up.

4.2 The CJC language specification

The above presented modified system of sequential processes we use as the principal notion for cooperation of job control processes encompassing the OpDevices which in our model are the lowest form of processes. The lot may be described in more rigorous, language style. The CJC language structure description will show how the primitive elements may be used to compose larger constructions.

For syntactic notation the Backus-Naur form is used. But because of many existing variations of the BNF [33] we give a short summary of the chosen form in Appendix A.

Following the earlier practices [15, 25] the general language constructs like types, declaration, expressions, variables etc., are borrowed from PASCAL [34] and will not be treated here.

First of all we present here the process construction because of its principal character in the CJS system model. Then follow definition of messages, commands, alternation commands and operation commands as these have vital meaning to the model.

4.2.1 Processes

⟨process⟩ ::= ⟨process label⟩:⟨command list⟩
⟨command list⟩ ::= ⟨declaration part⟩ ⟨command⟩ { ; ⟨command⟩ }
⟨command⟩ ::= ⟨simple command⟩|⟨structural command⟩

⟨simple command⟩ ::= ⟨null command⟩|
⟨assignment command⟩|
⟨input command⟩|
⟨output command⟩
⟨operation command⟩|
⟨SKIP command⟩|
⟨CONT command⟩ |
< ‡‡ command >

⟨structural command⟩ ::= ⟨alternative command⟩|
⟨repettive command⟩|
⟨parallel command⟩

⟨parallel command⟩ ::= [⟨process⟩{ || ⟨process⟩}]
⟨assignment command⟩ ::= ⟨ordinary assignment⟩|
⟨message assignment⟩
⟨declaration part⟩ ::= ⟨type definition part⟩
⟨variable definition part⟩
⟨message definition part⟩
⟨message variable definition part⟩
⟨operation definition part⟩

4.2.2. Messages

Messages constitute the very essence of input and output commands. The message construction enables packing the information of differentiated type and to distinguish particular representative in a lot.

⟨message difinition part⟩ ::=⟨message⟩ ⟨message name⟩(⟨wrap list⟩) {,⟨message name⟩ (⟨wrap list⟩)}
⟨message name⟩ ::= ⟨identifier⟩
⟨wrap list⟩ ::= ⟨empty list⟩|⟨wrap⟩ ,{⟨wrap⟩}
⟨wrap⟩ ::= ⟨wrap name⟩ : ⟨wrap type⟩

⟨wrap name⟩ ::= ⟨identifier⟩
⟨wrap type⟩ ::= ⟨type⟩

Example 4.1

As an example let us consider the following description
message InSgnl (start: STEP_TYPE, att: COLOUR).
OutSgnl (end_sgnl: SIGNAL);

The description defines two messages InSgnl end OutSgnl which are dedicated for an OpProcedure which controls an OpDevice.The first message contains two wraps: start – which defines what sort of elementary step the OpDevice is wanted to do, and att – an attribute of the elementary step.

With such a construction of message we can define message variables

⟨message variable definition part⟩ ::=
    message var ⟨message name⟩ : ⟨message type⟩ { , ⟨message name⟩ : ⟨message type⟩ }
⟨message name⟩ ::= ⟨identifier⟩
⟨message type⟩ ::= ⟨identifier⟩

Example 4.2

Following the Example 4.1 we can formulate several message variables of the same type

message var Ins1, Ins2 : InSgnl;
            Outs1, Outs2 : OutSgnl ;

Message variables, as ordinary variables, are supposed to carry definitive values. This may be done by assigning values to the wraps in the message. Here is the assignment definition

⟨message assignment⟩ ::= ⟨message variable⟩ :=< message expression >

⟨message expression⟩ ::= ⟨message variable⟩|
        ⟨message name⟩ (⟨message list⟩)
⟨message list⟩ ::= ⟨empty list⟩ | ⟨message component⟩ { , ⟨message component⟩ }

⟨message component⟩ ::= ⟨wrap name⟩ :: ⟨expression⟩

Example 4.3

A message Ins accepts defined values if we let
        Ins := Ins1 (start :: q, att :: RED)

Refering to an individual wrap of a message we use a wrap selector, in which the message variable is suffixed by a period and the wrap identifier.

Example 4.4.
    x:= Ins1.start;

4.2.3 Input and output commands

Input (output commands consist of two parts: the first one which defines the sending) receiving process and the second, consisiting of a message.

⟨output command⟩ ::= ⟨receiving process⟩ ! ⟨message expression⟩
⟨receiving process⟩ ::= ⟨process label⟩
⟨input command⟩ ::= ⟨sending process⟩ ? ⟨message variable⟩ ;

A process containing an output command executes the command and proceeds despite the named receiving process may not be ready to accept the message.
It is different, as we know (chapter 4.1), with a process containing an input command: the process stops when meeting the command and keeps waiting as long as named sending process sends its message.

A receiving process Jr may be addressed by sending processes beyond the time of the Jr action on the input command. For this reason some means must be given to store messages of particular construction arriving to the Jr. The means must be placed in an implementation processor (see Chapter 8 describing the JProcessor with InComBuf – a buffer for incoming messages managed by JProcCPU). The messages of particular construction are processed in the order they were received and stored. When the Jr is through with processing one input message, the process proceeds to the next command despite that the InComBuf may contain other messages of the same construction. These may be eventually processed in a next turn of the Jr process.

4.2.4 Guarded and repetitive commands

The inside process control is performed by means of guarded commands [9].

⟨guarded command⟩ ::= ⟨guard⟩ → ⟨command list⟩ | ⟨guard⟩ → #

⟨guard⟩ ::= ⟨guard list⟩ ; ⟨input command⟩ |
⟨guard list⟩ | ⟨input command⟩

⟨guard list⟩ ::= ⟨boolean expression⟩ { ; ⟨boolean expression⟩ }

⟨alternative command⟩ ::=
[⟨guarded command⟩ { □ ⟨guarded command⟩ }]

⟨repetitive command⟩ ::= * ⟨alternative command⟩

Let us remind that the command list in a guarded command may contain different sort of commands. This is then a possibility to obtain a nested alternative command. We then introduce a command SKIP which, when placed in a command list of a nested guarded command, allows to skip all the following commands up to the last square bracket of the outer alternative. The command CONT in the other arm allows to continue the following commands without skipping. The usefulness of such a mechanism will be shown later. The distinguished command # allows to escape the following repetitions provided the pertinent guard fails.

4.2.5 Operation command

An operation command disposes the operation performance which is composed of a series of actions controlled by OpProcedure and executed by OpDevice. We give here a definition of the operation. Such a definition will be used in declaration part of every JobProcess.

⟨operation definition⟩ ::= ⟨operation head⟩
⟨operation body⟩

⟨operation head⟩ ::= OpProcedure ⟨IDENTIFIER⟩ (
⟨parameter: type⟩ { ⟨parameter: type⟩ });

⟨operation body⟩ ::= ⟨declaration part⟩
⟨command sequence⟩

⟨declaration part⟩ ::= ⟨constant definition part⟩
⟨variable definition part⟩
⟨type definition part⟩
⟨message definition part⟩
⟨OpDevice definition part⟩
⟨message variable definition part⟩

⟨compound statement⟩ ::= ⟨statement⟩ { ; ⟨statement⟩ }
⟨output command to OpDevice⟩
⟨input command from OpDevice⟩

⟨statement⟩ ::= ⟨simple statement⟩ | ⟨structured statement⟩
⟨simple statement⟩ ::= ⟨assignment⟩
⟨assignment⟩ ::= ⟨identifier⟩ := ⟨expression⟩
⟨structured statement⟩ ::= ⟨compound statement⟩ |
⟨IF statement⟩ |
⟨WHILE statement⟩ |
⟨FOR statement⟩ |
⟨CASE statement⟩ |
⟨REPEAT statement⟩

⟨output command to OpDevice⟩ ::=
⟨OpDevice identifier⟩ ! ⟨message expression⟩

⟨input command from OpDevice⟩ ::=
⟨OpDevice identifier⟩ ? ⟨message variable⟩

The IF statement and the following structured statements have exactly the meaning formulated for PASCAL language [34] .
A similarity between the just presented operation definition and the procedure definition in PASCAL language pertains rather the general outlook. There are important differences. In case of OpProcedure:

- there exists communication with the real world by means of OpDevices,
- an OpProcedurs is dedicated to a particular JobProcess.

5. EXAMPLES OF CJC DESCRIPTIONS

The presented elements of CJC language may now be used to formulate descriptions of some complex job control processes.

Example 5.1

The complex job pertains to manufacturing a metal cylinder of defined height, diameter and smoothness of its surface. A metal rod of the desired diameter is cut into pieces of defined height. The pieces are then polished to the desired surface smoothness. According to the presented technology, the cylinder manufacturing contains two distinct jobs: Job1 i.e., cutting the rod and Job2-polishing the rod pieces. Supposing the jobs are to be controlled by means of pertinent control processes J1 and J2, the processes in turn consist of the following operations:

- operations of J1:

  A1 - fastening the rod for cutting,
  A2 - cutting the rod,
  A1 - transporting the cut piece of rod to the container;

- operations of J2:

  T2 - taking a piece of rod from the container,
  P1 - fastening the piece for polishing,
  P2 - polishing,
  P3 - taking the polished cylinder from the polishing device and transporting it for storage.

The two simple processes may be described as follows:

J1:: A1; A2; T1.
J2:: T2; P1; P2; P3.

The description components are the names of operation commands. When the implementation of J1, J2 processes is done then every operation command is supported by its OpDevice realization: the job processes J1, J2 have appropriate support by means of JProcessors (e.g., JPR1, JPR2) which contain necessary operation procedures in their OpProcStores. Cooperation of the two processes J1, J2 may take form of a simple sequence:

CJ:: J1; J2.

if the realization time of the processes is approximately equal and constant. But when the assumptions are not true the sequential organization may appear uneffective.

Example 5.2

Suppose the mean value of performance time of J1 is three times the time value of J2. Potential loss of time when J2 waits for the end of J1 we can eliminate giving three processes of the J1 type:

```
J1a:: A1a; A2; T1a.
J1b:: A1b; A2bł T1b.
J1c:: A1c; A2; T1c.
```

The three processes J1a, J1b, J1c, are supposed to run concurently delivering its products (the pieces of rod)to the buffering container from which the J2 process takes the pieces for further treatment. To obtain high effectiveness of the complex job we assume that the access to the buffer is asynchronous and nondeterministic. For these ressons it is feasible to introduce a third process J3 with the task: buffer access management. We shall present this new process together with the other cooperating processes.

```
(* Buffer management process *)
J3:: * [ J1a? r1a; (B <BMAX)→J1a! m1a() ; B:=B+1□
         J1b? r1b; (B <BMAX)→J1b! m1b() ; B:=B+1□
         J1c? r1c; (B <BMAX)→J1c! m1c() ; B:=B+1□
         J2 ? r2 ; (B>0)    →J2 ! m2 () ; B:=B-1]

(* Buffer supplying processes *)
J1a:: *[A1a ; A2a ; J3! r1a ( ) ;J3? m1a ; T1a]
J1b:: *[A1b ; A2b ; J3 ! r1b( ) ; J3? m1b ; T1b]
J1c:: *[A1c ; A2c ; J3! r1c ( ) ; J3? m1c ; T1c]

(* Buffer emptying process *)
J2:: * [ J3! r2 ( );  J3? m2 ; P1 ; P2 ; p3]
(*  The complex process *)
CJ:: [J1a || J1b || J1c || J2 ||  J3]
```

The J3 process construction is based on checking the buffer access by means of two element guards: the first guard element is the input command from the aces needing processes J1a, J1b, J1c, J2 ; as the second guard element serves the buffer status. The buffer status is to show up a free place (B < BMAX) for the supplying processes (J1a, J1b, J1c) and have some content (B>Ø) for the emptying process (J2). The output commands in the supplying and emptying processes are to ask for the access permission and the input commands - to receive the permiesion from the management process.

All the component processes repeat infinitely and run concurently forming in such a way the complex process CJ.

Example 5.3

In an observation post the information logging of objects is performed by means of operations A1a, A1b, A1c. The logging product i.e., information of different sort is normalized with operations A2a, A2b, A2c and then stored in the buffer with T1a, T1b, T1c. Process J2 pertains to operations: T2 - emptying the buffer, P1 - package organization, P2 - extraction of features, P3 - classification of objects to which the information belongs. Cooperation among buffer supplying and buffer emptying processes may have the same scheme as the one in Example 5.2.

6. INTERFACE PROCESS

A CJ process, like the one in Examples 5.2, 5.3, must be equipped with interfacing means between the human operator and the CJ process. For this purpose an extra process - IF process - is introduced which we want to perform the following functions:

a inserting the primary values for variables in JOBProcesses,
b inserting the arguments for guards,

c receiving the messages from JobProcess; in particular the end message with parameters of the CJ process task,

d changing the primary values of JobProcess variables and guard parameters,

e halting the course of JobProcesses.

Thus the IF process is supposed to receive reports on the CJ task and to manage the CJ process course and all its components, the JobProcesses.

6.1 The JobProcess management

We add to every JobProcess a port in the form of input command which is receiving messages from the IF process. The message contains two groups of wraps: the first group carries the primary values for JobProcess variables, the second one carries the guard arguments.

Let us go back to Example 5.3 and consider, say, the J1a process which obtains the following form:

```
J1a:: *[IF ? mes1 ;x1:=mes1.W1 ; g:=mes1.W2;
        [ G1(g) SKIP□   G2(g)→CONT ];
          A1(x1); A2 ;  J3!m1A(); J3?m1a  ; T1a];
```

In the input command IF ? mes1, the message has the construction: message mes1 (W1: FEATURE, W2: GUARD_ARGUMENT). The same construction has the message in pertinent output command in the IF process. The wrap V/1 is to establish the name of FEATURE to be measured by operation A1 ; the wrap W2 carries an argument for the two guards G1( ) and G2( ). There are two new commands connected with the guards. The command SKIP which says that all the following commands up to the square brackett ending the repetition command, should be omitted. The command CONT is an alternative to the command SKIP i.e., it orders to perform all the commands up to the brackett. In both situations the process J1a returns to its beginning and halts on the input command IF ? mws1 until a new message mes1 comes from the managing IF process.

6.2. The IF process structure

The IF process plays a role of interface between an automated CJ process and a human operator USER who estimates results from CJ process and inserts data to manage the course of the CJ process. The human operator is equip d with a terminal which is an OpDevice or device process TRMNL.

Let us return to our Example 5.3 and suppose that the information logging system has a task to observe the appearing objects. The observation proceedes until appears an object belonging to the defined class, then the observation is halted. For such a task our IF process may have the following fuctions:

- to receive a message mes2 (W1:REPORT) which contains data on the class membership of the object; the message comes from J2,
- to pass the message mes2 ( ) to the USER through TRMNL

Along with the above the IF process has to

- inform the USER when the following cycle begins (TRMNL ! wakeup),
- receive from the USER (TRMNL ? mes1) a message message mes1 (W1, W2, W3, W4: GUARD-ARGUMENT) containg data for the observation processes J1a, J1b, J1c and the classification process J2,
- pass the massage mes1 to the observation processes and the classification process to establish their actual course.

The IF process may then have a program

```
IF:: *[TRMNL ! wakeup; TRMNL? mes1 ;
        J1! mes1 ( );  J1b! mes1( ) ; J1c!mes1( ) ; J2!mes 1( );
        J2? mes2: TRMNL ! mes2( )]
```

An input port, similar to the one introduced into J1 processes, we have to insert also in J2

```
J2:: * [IF ? mes1 : h:= mes1.W4 ;
        G3(h)⊣SKIP □ G4(h)⊣ CONT ;
       J3!r2( ); J3?m2 ; P1; P2; P3; IF!mes2( )].
```

Such a simple construction as the one above allows the USER to obtain a complete supervision of the CJ process. The USER estimates the reports pertaining the complex job performance and with the information he decides about the process behaviour.

7. DECISION SUPPORT SUBSYSTEM

When the CJ process has many concurrent processes and when the reports come not only from the final task completing process (as the process J2 in Example 5.3) but also from other components of the CJ process, the USER may need a help in storing the current reports and then in reasoning before he commences a successive cycle of the CJ process.

Modern solution to the problem may be a decision support subsystem (DSS) [5] which enables to mechanize the two functions. In particular we consider a special form of DSS: a production rule system (PRS) which is originated in the work of Post [22] .

According to the PRS philosophy [8 , 26] it is possible to capture the knowledge of an expert of particular profession in a form of production rules. It is also important that the expert knowledge in the production rule, is a small selfcontained part of the whole knowledge volume. When we interprete one rule then the knowledge of the rest of rules is not required.

During the last few years the interest in the PRS design has been strongly increased. The systems obtained a new commercially attractive name "expert system" [13]. An expert system acquires a typical design: it contains

1 a bank of production rules sometimes called a knowledge base,
2 a working memory in which all the data elements are contained, either given as facts before starting the problem solution or created during the solution,
3 an inference engine, in older descriptions-interpreter for the rules [8]; the engine is responsible for the system control.

A rule has two distinct parts: "situation" - "action" forming a conditional statement

```
IF  ⟨antecedent   assertion no 1 is true⟩
    ⟨antecedent   assertion no 2 is true⟩
     .
     .
     .
THEN⟨consequent assertion no 1 is true⟩
    ⟨consequent assertion no 2 is true⟩
```

There are many ways to represent facts and rules. Because we are interested in the mechanized inferencing we confine our attention to the ways concluded in a computer language form. Among the computer languages in artificiel intelligence (AI) the language LISP plays an unquestionably outstanding role. Like other AI languages [5], LISP is in constant evolution. Nevertheless the form called COMMON LISP [32] is the one sufficiently rich for complex problems and also as the language version accepted by many reserchers. Thus LISP may be treated as a sort of basic language for the design of expert systems.

Let us have a short insight into the basic mechanism descriptions. Following the Example 5.3 we deliver a simple representation of facts and rules written */ in COMMON LISP.

*/ See Appendix B

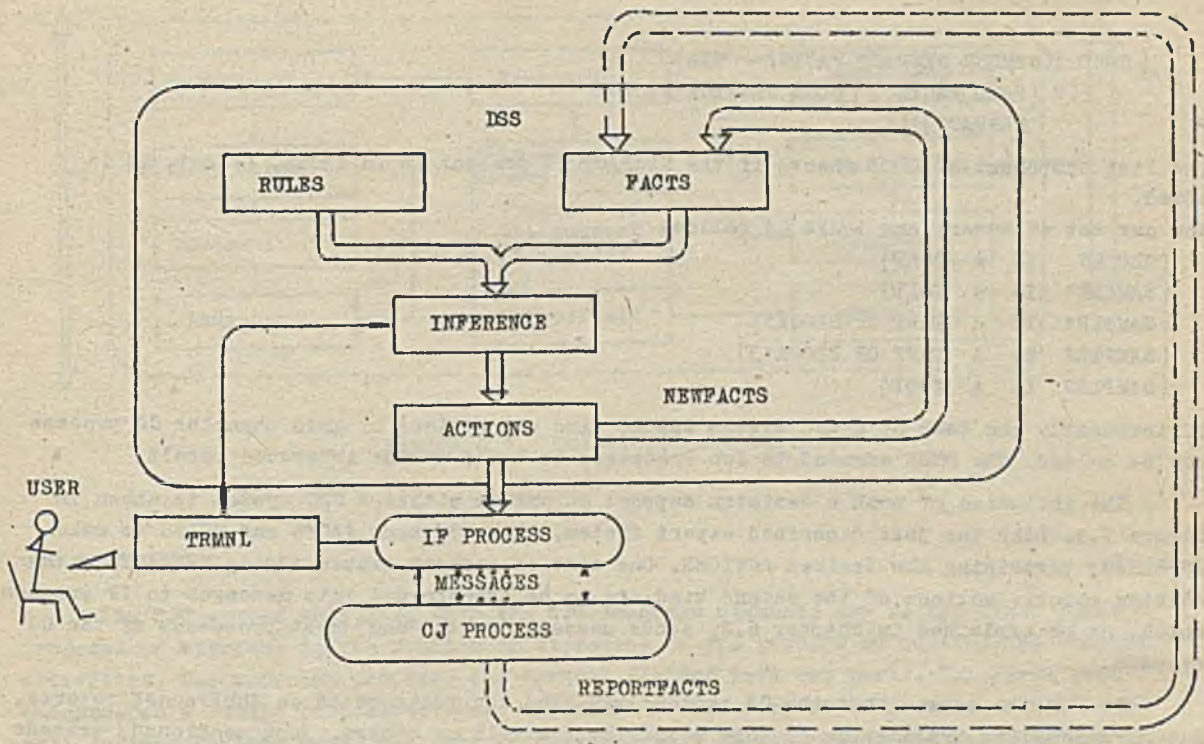


Figure 7.1 Cooperation of DSS with CJ process.

In the begining we have to formulate a bank of facts:

```
(SETQ  FACTS
       '((SAMPLE1 IS A GOLD)
         (SAMPLE1 IS A PART OF BLOCK5 )
         (SAMPLE2 IS A PART OF BLOCK13)
         (SAMPLE2 IS A WOOD)))
```

Starting with such a collection of assertions called FACTS an expert system tries all possessed rules matching the assertions and producing new ones as long as no rule exists that matches the assertions.

The procedure of obstaining new assertions is done as forward chaining. Here the problem solving actor looks for rules that depend only on already existing assertions. Say we have rule that enables qualification of objects on the basis of known sample features

```
(RULE   IDENTIFY23
        (IF      ((?SAMPLE ) IS   A (> TYPE))
                 ((?SAMPLE ) IS   A  PART OF(> OBJECT)))
        (THEN    ((<OBJECT ) IS   A(<TYPE ))))
```

Using the rule IDENTIFY23 and the facts included in FACTS one can conclude that the object BLOCK5 is made of gold

```
(BLOCK5 IS A GOLD)
```

and the above assertion we must be able to introduce into ASSERTIONS. For this purpose we need a new procedure INCLUDE which may be constructed in the following way

```
(DEFUN INCLUDE ( NEWFACT)
```

```
( COND ((MEMBER NEWFACT FACTS)   NIL)
       (T (SETQ FACTS   (CONS NEWFACT FACTS))
          NEWFACT)))
```

The list component of COND checks if the NEWFACT is present in in FACTS, if not, it is added.
Now our set of assertions looks as follows

(BLOCK5 IS A GOLD)
(SAMPLE1 IS A GOLD)
(SAMPLE1 IS A PART OF BLOCK5)
(SAMPLE2 IS A PART OF BLOCK13)
(SAMPLE2 IS A WOOD)

If incidently the task of a CJC system was to find out a piece of gold then the CJ process may be halted. The USER command to Job Processes is based on the interence result.

The inclusion of such a decision support subsystem within a CJC system is shown in Figure 7.1. Like the just described expert system, the DSS uses FACTS and RULES to make INFERENCE pertaining the desired ACTIONS. One sort of actions causes adding NEWFACT to the working memory; actions of the second kind are to be transformed into messages to IF process which, as we explained in chapter 6.2, sends messages to the component processes of the CJ process.

We further assume that the CJ process may send new facts based on JobProcess reports. But more complete explanation of this mechanism (as well as others, just mentioned) exceeds the frames of this work.

8. ON IMPLEMENTATION OF JOB PROCESSES

Although the discussion on JobProcess implementation clearly goes beyond the predetermined subject of this paper (modelling), nonetheless we want to sketch an abstract construction of a processor dedicated to execution of commands composing a JobProcess. The presentation may help to imagine the technicalities concerned with eventual CJC system realization.

Any JProcessor performs three kinds of tasks. Obviously the main task is to execute the current commands of the JobProcess program contained in the JProcMemory. When the current command is an operation command the appropriate OpProcedure is transfered from OpProcStore to a special field OpProcField in JProcMemory, and of course the control is passed to the field. The OpProcStore contains all the OpProcedures necessary for the current JobProcess program to be performed.

The JProcessor is supposed to communicate with other JProcessors. The communication service is done by the following modules. LanProtInt is a module which accepts all the messages that come along the local area network medium (Lan) and are addressed to the JProcessor; it also organizes the messages to other JProcessors. The messages must obey particular transfer protocol. As the cooperation with Lan runs concurrently with the program interpretation, the messages coming to the JProcessor will be stored in the buffer InComBuf. The registers InComReg, OutComReg help the message transfer to be more affective.

The modules InSgnReg, OutSgnReg and busProtInt are to service the communication with OpDevice connected to the Bus. Here a buffer register is not needed as the JProcessor, after sending the InSgnl to the pertinent OpDrvice, halts on the next command waiting for the message OutSgnl from the OpDevice.

The JProcCPU module is the basic processing means necessary for managing the other modules in the structure. A general view of the JProcessor structure is given in Figure 8.1.

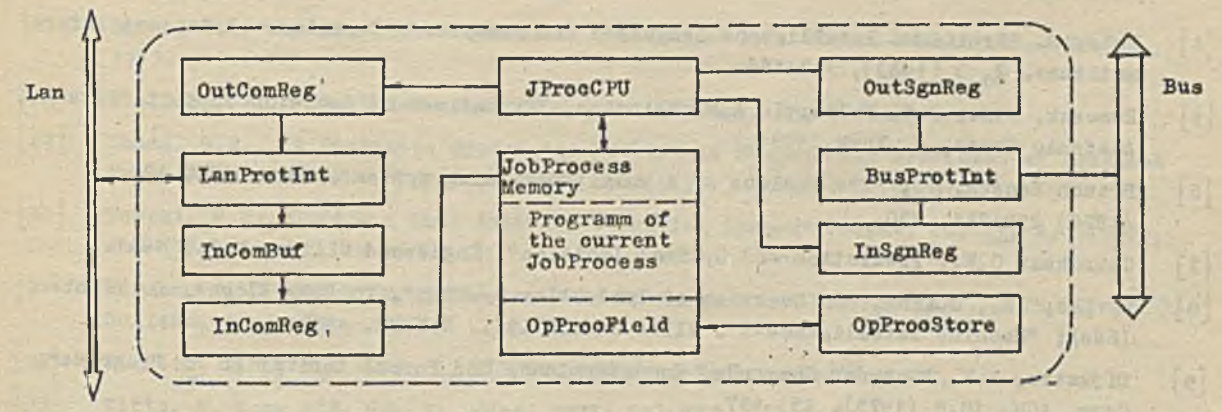


Figure 8.1. The JProcessor structure

9. CONCLUSIONS

The CJC system modelling with the CJC language elements are in the first place a proposal of approach to the problem of approcach to the problem of controlling complex activities. The approach proffers the control devided into two parts. The first part encompasses a bundle of JobProcesses, organized in the form of CJ process, which do a well defined routine work. The more intelligent part of control is dedicated to the system manager (USER) who equipped with a decision support subsystem handles all the exceptional events, changes routine courses of JobProcesses, makes other system services.

Such an architecture seems to be adventageous for all the new system designs consolidating large number of computer means possibly with diversity of functions wanted to run concurrently and having a difficult task to do.

A further research upon the CJC modelling and language may pertain to language implementations and search for an adequate expert system architecture with an AI language for the decision support system. The second direction we consider very important as it may help to promote the CJC language from the level of job control to a level of task formulating and problem solving language.

ACKNOWLEDGMENT

The author wishes to thank Professor R. Trechciński of the Institute of Nuclear Problems (Polish Academy of Sciences) and Mr J.Janowski of the Institut of Mathematical Machines for the interesting conversation and valuable remarks on the subject.

References

[1] Andrews, G.R., F.B. Schneider; "Concepts and Notations for Concurrent Programming". ACM Computing Surveys, IS,1 (1983), 3-43.

[2] Andrews G.R. "The Synchronization Resources". ACM Trans. Prog. Lang. Syst., 2, 4, (1981), 405-430.

[3] Besant, C.B. "Computer Aided Design and Manufacture" Ellis Horwood Ltd., New York, 1983.

[4] Boley H. "Artificial Intelligence Languages and Machines". Technique et Science Informatiques. 2, 3 (1983), 145-166.

[5] Bonczek, R.H., C.W. Holsapple A.W. Whinston, "Fundations of Decision Support Systems". Academic Press, N. York, 1981.

[6] Brinch Hansen, P., "The Nucleus of a Multiprogramming System". Comm. ACM, 13, 4, (1970) 238-241, 250.

[7] Churchman C.W., "Prediction and Optimal Decision". Englewood Cliffs, N.J., 1961.

[8] Davies, R., J.King, "An Overview of Production Systems", in E.W. Elcock, D. Michie (Eds); "Machine Intelligence 8". Ellis Herood Ltd., N.York. 1977.

[9] Dijkstra, E.W. "Guarded Commands, Nondeterminacy and Formal Derivation of Programs", Comm. ACN, 18,8 (1975), 453-457.

[10] Dijkstra, E.W. "The Structure of 'T E' Multiprogramming System". Comm, ACM, 11, 5 (1986), 341-346.

[11] Feldman, J.A., "High Level Programming for Distributed Computing". Comm. ACM, 22, 6 (1979) 353-368,

[12] Filman, R.E., D.P. Friedman, "Coordinated Computing, Tools and Techniques for Distributed Software", McGraw - Hill, New York, 1984.

[13] Hayes-Roth, F. et al (Eds), "Building Expert Systems". Addision-Wesley, London, 1983.

[14] Halevi,G. "The Role of Computers in Manufacturing Processes". J. Wiley, New York, 1980.

[15] Hoare, C.A.R. "Communicating Sequential Processes". Comm. ACM, 8, 21. (1978) 666-677.

[16] Hohenstein, L. "Computer Peripherals for Minicomputers, Microcomputers and Personal Computers., McGraw Bill. 1980.

[17] Horning, J.J., B. Randell, "Processes Structuring" ACM Comput. Surveys, 5,1 (1973) 5-30

[18] Krull, F.N., "Experience with ILLAD: A High-Level Process Control Language". Comm. ACM, 24, 2, (1981), 66-72.

[19] Laudet, M. et al (Eds.), "Medical Computing". Proc. Int. Symp. IRIA, Trulouse, March 1977.

[20] Lines, M.V., Minicomputer Systems". Wintrop Publishers, Reding, Ma., 1980.

[21] Paddock, C.E, R.W. Scamell, "Office Automation Projects and Their Impact on Organization, Planning and Control". ACM Transactions on Office Information Systems, 2, 4 , (1984) 289-302.

[22] Post, E., "Formal Reduction of the General Combinational Problem", Amer. J. Math., 65 (1943).

[23] Pratt, T.W., "Programming Languages: Design and Implementation". Prentice-Hall, Englewood Cliffs. N. Jersey, 1984.

[24] Ranky, P. "The Design and Operation of FMS Flexible Manufacturing Systems. "North-Holland, 1983.

[25] Roper, T.J., C.J. Barter, "A Communicating Sequential Process Language and Implementation". Software - Practice and Experience, 11 (1981), 1213-1234.

[26] Shortliffe, E.H., "Computer-Based Medical Consultation: Mycin". Elsevier Computer Science Library, N. York, 1976.

[27] Simon, W., "The Numerical Control of Machine Tools". Edward Arnold Ltd., London, 1973.

[28] Stalling, W., "Local Networks", ACM Comp. Surveys, 16, (1984), 3-41.

[29] Thoma, G.R., "A Prototype System for Electronic Storage and Retrieval of Document Images", - ACM Trans. Office Inform. Syst., 2, 3 (1985) 279-291.

[30] Turski, W.M. "SODA - A Dual Activity Operating System" Comput. J., 11, 2, (1986), 148-156.

[31] Warman, E.A. "Computer Applications in Production and Engineering", CAPE'83, North-Holland, Amsterdam, 1983.

[32] Winston, P.H., B.K.P. Horn, "LISP", Addision-Wesley, Reading, MA, 1983.

[33] Wirth, N. Comm ACM, 20, 11 /Nov. 1977) 822-823.

[34] Wirth, N. "The Programming Language PASCAL". Acta Informatica, 1, 1 (1971), 35-63.

[35] Wrzeszcz Z., "Potrzeby medycyny a możliwości krajowego przemysłu komputerowego", IMM - opracowanie wewnętrzne, 1985.

[36] Programming Language ADA Reference Manual, vol. 106, Lecture Notes in Computer Science, Springer-Verlag, N.York, 1981.

[37] Muroga, S., "VLSI System Design."When and How to Design Very-Large-Scale Integrated Circuits". Wiley-Interscience, New York, 1982.

[38] Batorcev, K.B., "Analogii i Modeli v Poznanii" (Analogies and Models in Scientific Cognition), "Nauka", Novosybirsk, 1981.

Appendix A

The Backus-Naur form(BNF) notation

⟨...⟩	Angular brackets enclose syntactic constructs denoted by English words.
⟨...⟩ ⟨...⟩	Concatenation of syntactic constructs.
{...}	Curly brackets express none or more repetitions
::=	To be read as "is defined as".
|	To be read as "or".
⟨empty⟩	Denotes null sequence of symbols.

Appendix B

Basic informations on LISP

Any symbolic construction, being a mathematical function, may take a form of LISP function. For instance, the function PLUS with two arguments A and B is written as PLUS A B . In the place of PLUS one may use any function say FUN which either exists in the function repertoire or must be defined in LISP.

The basic LISP objects are atoms. Using groups of atoms one may create objects similar to clauses, called lists. Creation of lists may follow using groups of lists. Atoms and lists together are called symbolic expressions or simply expressions.

Atom is an expression element and may pertain to a constant or a variable. One way of assigning a value to variable, say A, is to use the function SETQ

```
(SETQ  A  5)
  5
```

In the result the variable A acquires the value 5.

Let us consider the expression (SETQ A B). If B, as a variable, has the value e.g., 6, then B accepts the same value. But B may have a symbolic meaning e.g., a list. To pass such a meaning to A we must use the function QUOTE or shortly " ' ". Then the expression (SETQ A (Quote B)) assigns to A the symbol B despite that actual value of B may be 6.

As other languages, LISP contains reserved words, which cannot be used as variable identifiers. To the group belong, among others, the words T,F,NIL. The T and F are equivalent to 'true' and 'false'. But NIL has several applications and one of them is using it in place of 'false'.

LISP contains also a very useful construction, sort of procedural abstraction which allows to design new procedures (functions). As a general practice, the new procedures are created on the basis of the existing procedures. But the new ones, despite that they offer a saving of place and time in programming, they are purpose (task) oriented in the particular system design. The syntax of such a procedural abstaction has the following view

```
(DEFUN ⟨procedure  name⟩
   ( ⟨parameter1⟩ ...  ⟨parameter n⟩ )
     ⟨procedure body⟩ )
```

In place of conditional operators e.g., IF, WHEN etc., of languages such as PASCAL, the language LISP uses the conditional function COND. The function syntax is as follows

```
(COND (⟨test1⟩  ...  ⟨result1⟩)
      ------------------------
      (⟨test n⟩ ...  ⟨result n⟩))
```

The basic idea of the construction is to test the consecutive clauses (i.e., the lists) checking only the part 'test' as long as the result of checking is different from NIL. Only then the rest of the clause is returned as the value of the evaluated conditional form.

Function CONS belongs to a function group serving for manipulation on list components. First of all, in the group there are the functions CAR, CDR, CONS. As an argument of CAR function is taken a list and the CAR function value is the first element of the list. Assuming.

```
(SETQ A '(2  3  4))
  (2  3  4)
```

then

```
(CAR A)
  2
```

The function CDR also accepts a list as its argument, but the function value is a rest of the list left there after with drawing the first element.

```
(CDR A)
  (3 4)
```

Function CONS is used to insert the first element into a list. Using CONS one may, for instance, receive back the earlier form of the decomposad list A

```
( CONS ' (CAR A) ' (CDR  A))
  (2  3   4 )
```

The Author hopes that the above presented elements of LISP may give a help for these readers of chapter 7, which being not permanent users of LISP do not want to spend hours studying the full LISP reference texts.

STRESZCZENIE

Myśl przewodnia budowy modelu systemu sterowania pracą złożoną (system CJC) dotyczy dopasowania struktury komputerowych środków sterujących do struktury działalności podlegającej sterowaniu, bez względu na to z jaką dziedziną ta działalność się wiąże. Punktem wyjścia jest abstrakcyjna forma operacji, będącej pierwotnym elementem działalności. Operacja oraz zasada czasowego uporządkowania są konstytuentami takich struktur jak działalność prosta i działalność złożona. Wprowadzono następnie pewien abstrakcyjny przyrząd pod nazwą OpDevice, realizujący operację. Konstrukcja OpDevice jest pomostem prowadzącym do procesu sekwencyjnego, tj. do formy sterowania operacją, a stąd do formy sterowania działalnością. Dzięki temu proces sekwencyjny staje się reprezentacją działalności prostej (pracy prostej) zaś działalność złożona jest niczym innym jak zbiorem kooperujących procesów sekwencyjnych. Tak więc opisy zachowania się systememu CJC można formułować stosując odpowiednie języki komputerowe. Jako podstawę dla języka CJC przyjęto koncepcję CSP Hoare'a. Podano następnie kilka przykładów działalności złożonej zapisanych za pomocą języka CJC. Przykłady te zostały tak dobrane by pokazać tworzenie zarządzania systemem. System złożony zawiera:

- proces CJ - zespół skoordynowanych procesów roboczych,
- proces IF - pewien sprzęg pomiędzy procesem CJ oraz USERem,
- USER - specjalista zarządzający systemem,
- DSS - kontroler typu RBS, zawierający mechanizmy sztucznej inteligencji, sterujący kolejny akcjami procesu CJ.

Niniejszy artykuł zawiera krótką prezentację powyższych elementów systemu. Budowa modelu systemu zasadza się na filozofii, zgodnie z którą procesy CJ są odpowiedzialne za realizację zadań rutynowych zaś inteligentna, nadzorcza część sterowania jest przypisana specjaliście zarządzającemu systemem (USER) wyposażonemu w DSS.

РЕФЕРАТ

Идея создания модели системы управления сложным трудом (система CJC) заключается в приспособлении структуры компьютерных управляющих средств к структуре деятельности, подвергаемой управлению, независимо от того, к какой области относится эта деятельность. Исходной является абстрактная форма операции, являющейся первичным элементом деятельности. Операция а также принцип её упорядочения по времени являются составными частями таких структур, как простая и сложная деятельность.

Затем был введен некоторый абстрактный прибор под названием OpDevice осуществляющий операцию. Конструкция OpDevice является связывающим звеном, ведущим к последовательному процессу, то есть к виду управления операцией, а следовательно и к форме управления деятельностью. Благодаря этому, последовательный процесс становится представителем простой деятельности (простого труда), а сложная деятельность является ничем иным, как набором кооперирующих последовательных процессов. Таким образом, описания поведения системы CJC могут быть составлены с применением соответствующих компьютерных языков. В качестве основы для создания языка CJC была принята концепция CSP Hoare . Приведено несколько примеров сложной деятельности, описанных при помощи CJC. Примеры подобраны таким образом, чтобы показать процесс создавания управления системой.

Сложная система содержит:

- процесс CJ - комплекс координированных рабочих процессов,
- процесс IF - некоторая связь между процессом CJ и USER
- USER - специалист, заведующий системой,
- DSS - контроллёр типа RBS, содержащий механизмы искусственного интеллекта, управляющий очередными акциями процесса CJ.

В настоящей статье в сокращённой форме описаны вышеуказанные элементы системы. Строение модели основано на соображениях, согласно которым процессы CJ отвечают за осуществление нормальных задач, а интеллигентная, контрольная часть управления, остаётся в руках специалиста, заведующего системой (USER), оснащённого контроллером DSS.

Prace naukowo-badawcze Instytutu Maszyn Matematycznych

Preemptive Scheduling for Two-Processor Systems

by Andrzej Rowicki

Abstract

The purpose of the paper is to consider an algorithm for preemptive scheduling for two-processor systems with identical processors. Computations submitted to the systems are composed of dependent tasks with arbitrary execution times and contain no loops and have only one output. We assume that preemptions times are completely unconstrained and preemptions consume no time. Moreover, the algorithm determines the total execution time of the computation. It has been proved that this algorithm is optimal, that is, the total execution time of the computation (schedule length) is minimized.

1. Introduction

In general, if preemptions are permitted in executing of programs in multiprocessor computing systems, it is often possible to reduce the computation times of programs. This can be especially important for programs or computations which are executed many times, particularly in real-time applications. In these cases, the problem of optimal scheduling multiprocessor computer systems is of great importance.

In contrast to the problem considered in [1] , the purpose of the paper is to consider an algorithm for optimal preemptive schedeling of two-processor systems. We shall consider systems in which there are two identical processors and assume that the computations submitted to the systems are composed of tasks with arbitrary execution times. These computations contatin no loops and have one output. Mereover, we assume that preemption times are completely unconstrained. It means that it is permitted to interrupt any processor at any time and reasign it to a different task in such a way as to not violate the precedence relations among the tasks. Of course, it is not permitted to assign more than one task to a processor at any time.

It should be noted that we have assumed that the execution time for each task and the precedence relation are known a priori. This means that we only consider the static problem. From a more general point of view it is clear that the static scheduling model we have introduced is most applicable to those case in which a program or compuoation is to be executed many times, particularly in real time situations.

As a model of computation we assume a weighted directed graph with arbitrary node weights which is connected and contain no circuits and has one terminal vertex. This model can be interpreted as follows: The attachement of tasks of the computation to the vertices of the graph is such that the order of execution of the tasks is in accordance with succession relation of the vertices of the graph. A weight is associated with each node of the graph and determines the execution time of the corresponding task.

In [2] and [3] have been considered algorithms for optimal preemptive scheduling of two-processor systems for arbitrary acyclic computation and for optimal preemptive scheduling of multiprocessor systems for tree structured computations, respectively. It has assumed that all tasks of the computation have the mutually commensurable execution times which means that is permitted to interrupt any processor at stated times. On the other hand, this assumption means that all tasks of the computation are decomposed into chains of the tasks with the same execution times and thus the scheduling problem is of greater size in the size of the original problem. Of course, this influences on time complexity of the algorithm in question. The problem of determining the total execution time of the computation (schedule lenght) and time complexity of the algorithms presented in [2] and [3] has not been considered.

In [4] has been considered an extension of the preemptive algorithm for tree structured computations [3] to arbitrary acyclic computations for multiprocessor systems of uniformly different speeds. With the exeption of scheduling problems on two-processor systems or computations with independent tasks the algorithms considered in [4] produce suboptimal schedules. Moreover, in [5] has been considered an extension of optimal nonpreemptive algorithm for two-processor systems for preemptive case by assumption that all tasks of the acyclic computation have the mutually commensurable execution times. It seems that the resolution of a problem of preemptive scheduling by reducing to a problem of nonpreemptive scheduling for acyclic computations consisting of tasks with equal execution times, as was considered in [5] , aparat from the fact that not always this reduction is possible, cannot be find satisfactory. Obviously, the reduction based on decomposing all tasks into chains of tasks with equal execution times yields problems which are of greater size in the size of the original problems. In some cases this approach yields suboptimal solutions.

In this paper, we shall consider an algorithm for optimal preemptive scheduling of two-processor systems which has at most time complexity $O(n^2)$. We assume that the tasks of acyclic computation have arbitrary execution times and preemption times are completely unconstrained and thus we have no constraints on execution time and preemption time given in [2] and [5] . Moreover, this algorithm determines the total execution time of the computation (schedule length) provided that preemptions consume no time. Obviously, if scheduling or computation is completed then by counting the number of assignments of processors to the tasks and relevant processing times we can determine the execution time of the computation before completing the scheduling. This algorithm is optimal in the sense that the total execution time is minimized.

Description of the algorithm considered in this paper is strongly dependent on the structure of the computation. This fact allows us to inquire into structure of the computation and recognize the limitations in realization of the computation.

The algorithm presented in this paper is so general that can be extended on multiprocessor systems [6] and can be treated as an extension of the algorithm considered [1] . The algorithm considered is a modified and more compact version of the algorithm considered in [7,8,9].

2. Basic notions and definitions

Let us consider a weighteed graph $G=\langle X, \Gamma, t, x_0 \rangle$, where X is a finite set of vertices, Γ is a relation defined on the set X , t is a function which associates with every vertex of the set X a positive number and x_0 is a terminal vertex of the graph G, that is such a vertex that $\Gamma(x_0) = \emptyset$ where $\emptyset$ is a void set. In further considerations the values of the function t will be interpreted as the execution time of the tasks associated with the arguments of the function t.

The graph $G = \langle X , \Gamma , t , x_0 \rangle$ will be called a network if and only if is connected, contains no circuits and has one terminal vertex.

In further considerstions we confine ourselves to networks. The networks will be denoted by a capital S .

Let $S = \langle X , \Gamma , t , x_0 \rangle$ be an arbitrary but fixed network. Let $L'(i)$ for $i = 1,2,\ldots,n$ be a subset of X defined by induction in the following way:

i) $L'(1) = \{x_0\}$

ii) if $x \in Z_i$ and $t(x) = \min_{y \in Z_i} \{t(y)\}$ then $x \in L'(i)$

where Z_i is a subset of the set X defined as follows:

$$Z_i = \left\{ x \in X : x \in X - \bigcup_{j=1}^{i-1} L'(j) \wedge \Gamma(x) \subset \bigcup_{j=1}^{i-1} L'(j) \right\}$$

Let p be a natural number satisfying the relation $\bigcup_{j=1}^{p} L'(j) = X$, then for $1 \leq i \leq p$ a subset L(i) of the set X defined in the folloing way:

$$L(i) = L'(p+1-i)$$

will be called the i-th level of the network S.

It is easy to verify that the family of sets $\{L(i)\}_{i=1,2,..,p}$ is a disjoint partition of the set X of the network S, that is, for every $1 \leq i,j \leq p$ if $i \neq j$ then $L(i) \cap L(j) \neq \emptyset$ and $\bigcup_{j=1}^{p} L(j) = X$.

The network $S' = \langle X', \Gamma', t', x_0' \rangle$ will be called a reduct of the network $S = \langle X , \Gamma , t , x_0 \rangle$ if and only if

i) $X' \subset X$ and $\Gamma' \subset \Gamma$ and $t' \subset t$

ii) if $x \in X'$ then $\Gamma(x) \subset X'$

It is easy to see that if the network $S' = \langle X', \Gamma , t', x_0' \rangle$ is the reduct of the network $S = \langle X , \Gamma , t , x_0 \rangle$ if and only if

i) $X - X' \neq \emptyset$

ii) $\Gamma(X - X') \subset X'$

The sequence of the networks $S_1,\ldots,S_i,\ldots,S_k$ will be called a reduction of the network S (in symbols $R(S)$) if and only if

i) $S_1 = S$

ii) for every $i > k$, S_{i+1} is the reduct of S_i

iii) for S_k there exists no direct reduct

The reduction $R(S) = S_1,\ldots,S_i,\ldots,S_k$ is said to be a direct reduction of the network S if and only if for every $i < k$, S_{i+1} is the direct reduct of S_i.

Let $S' = \langle X', \Gamma', t', x_0' \rangle$ and $S = \langle X , \Gamma , t , x_0 \rangle$ be arbitrary but fixed networks such that $X \subset X'$. The network S' is said to be an extension of the network S (in symbols $S \subset S'$) is said to be an extension of the network S in symbols $S \; S'$ if and only if there exist such two sets Z and Z^* that $Z \subset X \cap X'$ and $Z^* \subset X' - X$ satysfying the following conditions:

i) if $s \in Z$ then there exists such $s^* \in Z^*$ that $\Gamma'(s) = s^*$ and $\Gamma(s) = \Gamma'(s)^*$

ii) $t(s) = t'(s) + t'(s)$

iii) if $x \in X - Z$ then $\Gamma(x) = \Gamma'(x)$

The elements of the set Z will be called singular vertices of the network S'with respect to the network S . If the network S'contain only the one singular vertex then the network S'is said to be a direct extension of the network S.

It should be noted that if the network S´is any extension of the network S then the network S is not a reduct of the network S´and if $x_o' \notin Z$ then $x_o = x_o'$.

3. General idea of the scheduling algorithm

To describe the algorithm some notions will be needed. To make the understanding of the essential notions easier we will sketch the basic step of the algorithm. Certain details will be omitted here, so as not to obscure the basic idea. Informally speaking, the heart of the scheduling algorithm consists in searching for sets of the mutually independent vertices of the network in question and splitting some tasks if it is necessary.

We now proceed to an informal description of the algorithm. Let the network S_o be a model of computation and p_o denote the number of levels of the network S_o. In the first step we analyse the elements of the sequence

(1) $L_o(1),\ldots,L_o(n),\ldots,L_o(p_o)$

of levels of the network S_o until we determine the smallest $1 \leq n \leq p_o$ satisfying the following relation:

$$\Gamma_o\Big(\bigcup_{j=1}^{n} L_o(j)\Big) \cap L_o(n+1) \neq \emptyset$$

Next, we create a set M(0,n) determined as follows:

$$M(0,n) = \bigcup_{j=1}^{n} L_o(j)$$

If the cardinality of the set M(0,n) is greater than 1 then we disinguish a subset X_s of the set M(0,n) determined as follows:

$$X_s = \left\{ x \in M(0,n) : \Gamma_o(x) \cap L_o(n+1) = \emptyset \right\}$$

If the set $X_s \neq \emptyset$ then we create an extension S_o^M of the network S_o in such a way so as the set X_s to be a set of singular vertices of the network S_o with respect to network S_o^M satisfying the following relation:

if $x \in X_s$ then $t_o^M(x) = \tau_o(x) - \tau_o(y) + t_o(y)$

where y is an arbitrary vertex from the set $M(0,n) - X_s$ and $\tau_o(x)$ is the sum of values of the fuction for the vertices being on the longest path leading from the vertex x to the vertex x_{oo}.

In the case where $X_s = \emptyset$, then for the sake of convenience the network S_o will be denoted by S_o^M .

Next, we create a disjoint partition $P = \{P_1, P_2\}$ of the set M(0,n) in such a way so as to satisfy following relation:

(2) $t_o^M(P_1) = t_o^M(P_2)$

If there exists no disjoint partition P satisfying the relation (2) then we split one of the tasks corresponding to the set M(0,n) in such a way so as to obtain the partition P satisfying relation (2). The disjoint partition P determines the sets of vertices P_1 and P_2 in such a way that the tasks corresponding to these sets can be concurrently executed.

In the case where the cardinality of the set M(0,n) is equal to 1 then we do continue analysing the terms of the sequence (1) until we determine the smallest $n < m \leq p_o$ satisfying the following relation:

(3) $L_o(m) - \Gamma_o\Big(\bigcup_{j=1}^{m-1} L_o(j)\Big) \neq \emptyset$

If there exists the smallest $n<m\leqslant p_o$ satisfying the relation (3) then we create a set D(0) determined in the following way:

$$D(0) = L_o(m) - \Gamma_o\Big(\bigcup_{j=1}^{m-1} L_o(j)\Big)$$

If there exists no $n<m\leqslant p_o$ satisfying the relation(3) then we set $D(0) = \emptyset$. In this case the task corresponding to the one element set M(0,n) is executed by one processor and the remaining processor is idle in this period.

If the set $D(0) \neq \emptyset$ then we chose an arbitrary but fixed vertex $x_d$from the set D(0) and, if it is necessary, we create a direct extension S_o^p of the network such that

$$t_o^p(M(0,n)) = t_o^p(x_d)$$

Obviously, the tasks corresponding to M(0,n)and x_d can be concurrently executed by different processors.

Of course, if we split any task then the model of computation varies. In this case a new model will be an extension of the network S_o. The next steps of the algorithm proceed in a way similar to the first step just described provided that relevant models of computation have been used.

4. Description of the algorithm

To describe the algorithm some additional notions will be needed. These notions can be defined formally as follows:

Let $S = \langle X, \Gamma, t, x_o\rangle$ be an arbitrary but fixed network and let $S_i = \langle X_i, \Gamma_i, t_i, x_{oi}\rangle$ be a network assigned to the network S as a result of executing i-th step of the scheduling algorithm. Let F(i) be a subset of the set X_i created as a result of executing i-th step of the scheduling algorithm. In the set X_i we distinguish a subset M(i,n) defined in the following way:

$$M(i,n) = \bigcup_{j=1}^{n} L_i(j) - \bigcup_{j=0}^{i} F(j)$$

where $L_i(j)$ is the j-th level of the network S_i.

Let n for $0\leqslant i\leqslant p_i$ be a function defined as follows:

$$n(i) = \begin{cases} \text{the smallest } i<n<p_i \text{ such that} \\ \Gamma i(M(i,n)) \cap L_i(n+1) \neq \emptyset \\ i \quad \text{otherwise} \end{cases}$$

where p_i is a natural number satisfying the following relation $\bigcup_{j=1}^{p_i} L_i(i) = X_i$; that is, p_i determines the number of levels of the network S_i.

Let $S' = \langle X', \Gamma', t', x_o'\rangle$ be any extension of the network $S = \langle X_i, \Gamma_i, t_i, x_{oi}\rangle$. Let X_s denote a set of singular vertices of the network S'with respect to the network S_i. The network S'is said to be an extension of the network S_i induced by the set M(i,n (i)) (in short m-extension) if and only if :

i) $X_s = \{x \in M(i,n(i)) : \Gamma(x) \cap L_i(n(i)+1) = \emptyset\}$

ii) if $x \in X_s$ then $t'(x) = \gamma_i(x) - \gamma_i(y) + t_i(y)$

where y is an arbitrary vertex from the set M(i,n(i)) satisfying the following relation:

$$\Gamma_i(y) \cap L_i(n(i)+1) \neq \emptyset$$

and γ_i is a function defined as follows:

$$\gamma_i(x) = \begin{cases} t_i(x) & \text{if } x = x_{oi} \\ t_i(x) + \max\limits_{y\in\Gamma_i(x)}\{\gamma_i(y)\} & \text{if } x \neq x_{oi} \end{cases}$$

It is easy to verify that not for every network S_i there exists an extension induced by the set $M(i,n(i))$.

It can be shown that if S' is m-extension of the network S_i and X_s is the set of singular vertices of the network S' with respect to the network S_i then

$$L'(n(i)+1) = L_i(n(i)+1) \cup \Gamma'(X_s)$$

For the sake convenience, by S_i^M will be denoted an m-extension of the network S_i or the network S_i so far as there exists no m-extension for it.

A set $L(i,n)$ for $0 \leq i \leq p_i$ and $0 \leq n \leq p_i$ is said to be a restriction of the set $L_i(n)$ if it is defined as follows:

$$L(i,n) = L_i(n) - \bigcup_{j=0}^{i} F(j)$$

Let g for $0 \leq i \leq p_i$ be a function defined as follows:

$$g(i) = \begin{cases} \text{the smallest } n(i) < n < p_i \text{ such that} \\ L(i,n) - \Gamma_i(M(i,n-1)) \neq \emptyset \\ n(i) \qquad \text{otherwise} \end{cases}$$

Basing on the function g, we define a set $D(i)$ in the following way:

$$D(i) = L(i,g(i)) - \Gamma_i(M(i,g(i)-1))$$

It is easy to see that if $g(i) = n(i)$ then $D(i) = \emptyset$.

In order to reduce the number of the tasks to be split, we distinguish in the set $D(i)$ a subset $D'(i)$ defined as follows:

$$D'(i) = \left\{ x \in D(i) : t_i(x) = \max_{y \in D(i)} \left\{ t_i(y) \right\} \right\}$$

Let us assume that x_d is an arbitraty but fixed vertex belonging to the set $D'(i)$. By this assumption we define a set $M^*(i)$ in the following way:

$$M^*(i) = \begin{cases} M(i,n(i)) & \text{if } \|M(i,n(i))\| > 1 \\ M(i,n(i)) \cup \{x_d\} & \text{if } \|M(i,n(i))\| = 1 \end{cases}$$

where $\|X\|$ denotes the cardinality of the set X.

Of, course, if we are not interested in reducing the number of the tasks to be split, then we can take as x_d an arbitrary but fixed vertex from the set $D(i)$.

For the sake of conciseness of description of the algorithm we introduce some additional notions.

Let a sequence $P(M^*(i)) = P_1(i), P_2(i), P_3(i)$ be a disjonit partition of the set $M^*(i)$ defined in the following way:

i) $t_i^M(P_j(i)) \leq T(i)$ for $j = 1,2$

ii) $P_3(i) = M^*(i) - (P_1(i) \cup P_2(i))$

iii) if $x \in P_j(i)$ and $j \neq n$ then $t_i^M(P_n(i)) + t_i^M(x) > T(i) > T(i)$ for $j = 1,2,3$ and $n = 1,2$

where T is a function defined on the set X_i^M as follows:

$$T(i) = 1/2 \sum_{x \in M^*(i)} t_i^M(x)$$

It is easy to verify that $\|P_3(i)\| \leq 1$ and if $t_i^M(P_3(i)) > T(i)$ then one of the sets $P_1(i)$ or $P_2(i)$ is empty.

Under assumption that $P_3(i) \neq \emptyset$, we define a direct extension $S_i^p = \langle X_i^p, \Gamma_i^p, t_i^p, x_{oi}^p \rangle$ of the network $S_i^M = \langle X_i^M, \Gamma_i^M, t_i^M, x_{oi}^M \rangle$ induced by the disjoint partition $P(M^*(i))$ in the following way:

If $t_i^M(P_3(i)) < T(i)$ then $\{s_i\} = P_3(i)$ and

$$t_i^p(s_1) = T(i) - t_i^M(P_1(i))$$

$$t_i^p(s_1^*) = T(i) - t_i^M(P_2(i)) = t_i^M(s_1) - t_i^p(s_1)$$

If $t_i^M(P_3(i)) > T(i)$ then $\{s_i\} = P_3(i)$ and

$$t_i^p(s_1) = t_i^M(P_1(i) \cup P_2(i))$$

$$t_i^p(s_1^*) = t_i^M(s_1) - t_i^p(s_1)$$

where s_1 is only the one singular vertex of the network S_i^p with respect to the network S_i^M and s_i^* is an associated vertex with the vertex s_1.

We are now in a position to describe the process of creation the sets $F(i)$ and assignment of extensions S_i to the network S. This process will be called an initial sequencing algorithm.

The initial sequencing algorithm is defined by induction as follows:

1. Basic step
   $F(0) = \emptyset$ and $S_0 = S$
2. Induction step

   Case 1

   If $\|M^*(i)\| = 1$ then $F(i+1) = M^*(i)$ and $S_{i+1} = S_i$

   Case 2

   If $\|M^*(i)\| > 1$

   a) if $P_3(i) = \emptyset$ then $F(i+1) = M^*(i)$ and $S_{i+1} = S_i^p$

   b) if $t_i^M(P_3(i)) > T(i)$ then $F(i+1) = M^*(i)$ and $S_{i+1} = S_i^p$

   c) if $t_i^M(P_3(i)) < T(i)$ then $F(i+1) = M^*(i) \cup \{s_i^*\}$ and $S_{i+1} = S_i^p$

Let a sequence $S_1, S_2, \ldots, S_k$ be the sequence of networks assigned to the network $S = \langle X, \Gamma, t, x_o \rangle$ as a result of acting of the initial sequencing algorithm. The network S_k is said to be a terminal extension of the network S if the following relation

$$X_k = \bigcup_{j=1}^{k} F(j)$$

holds and the number k will be called a range of extension of the network S_k.

In the following considerations, the terminal extension of the network S will be denoted by a capital S^*.

It should be pointed out that if as a result of the action of the initial sequencing algorithm none of the tasks have been split then $S = S^*$.

As a result of the action of the initial sequencing algorithm we obtain a disjoint partition $\{F(i)\}_{i=1,2,\ldots,k}$ of the set X^* of the network S^*, that is, for every $i,j \leq k$ if $i \neq j$ then $F(i) \cap F(j) = \emptyset$ and $\bigcup_{j=1}^{k} F(j) = X$. If there exist no such a number $1 \leq i \leq k$ that $s_i^* \in F(i+1)$ then the disjoint partition $\{F(i)\}_{i=1,2,\ldots,k}$ detemines the sets of mutually independent tasks of the computation which is represented by the network S^* and thus the tasks corresponding to the sets $F(i)$ can be concurrently executed. On the other hand in the case where $s_i^* \in F(i+1)$

then between vertices s_i and s_i^* the relation $\Gamma^*(s_i) = s_i^*$ holds. This means that the tasks corresponding to the vertices s_i and s_i^* are dependent and thus the task corresponding to the vertex s_i^* ought to be executed after the execution od the task corresponding to the vertex s_i.

The initial sequencing algorithm acts by assumption that all tasks of the computation can be split in an arbitrary way. The starting point of acting of the initial sequencing algorithm are levels of the network. In any step of the algorithm some tasks of the computation can be split. If we split any task then the model of computation varies. This new model is needed to perform next step of the algorithm. In connection with there is necessity for determing the levels of the network representing new model of the computation. It is easy to verify that the process of determination of levels in the network being the model of computation in question reduce to adding to one of existing levels the successors of singular vertices or to introducing only one new level consisting of the one successor of the singular vertex.

It remains now to solve a problem of assignment the tasks to the different processors, that is, the problem of creation of a disjoint partition of the sets P(i) in such a way so as to minimize the total execution time of the computation and do not violate the precedence relation among the tasks. This process will be called a partitioning algorithm.

Let r be a range of extension of the network S^*. In the set X^*, for $1 \leq i \leq r$ and $j = 1,2$ we distinguish some subsets $P(i,j)$ defined in the following way:

Case 1

If $\|F(i+1)\| = 1$ then $P(i,j) = \begin{cases} F(i+1) & \text{if } j = 1 \\ \emptyset & \text{if } j = 2 \end{cases}$

Case 2

If $\|F(i+1)\| > 1$ then

1. if $P_3(i) = \emptyset$ then $P(i,j) = P_j(i)$ for $j = 1,2$
2. if $t_1^M(P_3(i)) > T(i)$ then
$$P(i,j) = \begin{cases} P_1(i) \cup P_2(i) & \text{if } j = 1 \\ \{s_i\} & \text{if } j = 2 \end{cases}$$
3. if $t_1^M(P_3(i)) < T(i)$ then
$$P(i,j) = \begin{cases} P_1(i) \cup \{s_i\} & \text{if } j = 1 \\ P_2(i) \cup \{s_i\} & \text{if } j = 2 \end{cases}$$

Let P_j for $j = 1,2$ be a subset of the set X^* defined in the following way:

$$P_j = \bigcup_{i=1}^{r} P(i,j)$$

It is easy to verify that the sets P_1 and P_2 create a disjoint partition of the set X^*, that is $P_1 \cap P_2 = \emptyset$ and $P_1 \cup P_2 = X^*$. The sets P_1 and P_2 determine the tasks which will be assigned to the different processors. If $P(i,j) \neq \emptyset$ for $j = 1,2$ then the tasks corresponding to the set $P(i,1)$ are executed by the first processor and the tasks corresponding to the set $P(i,2)$ are executed by the second processor. If $P(i,1) \neq \emptyset$ and $P(i,2) = \emptyset$ then the tasks corresponding to the set $P(i,1)$ are executed by the first processor and the second processor is idle in this period. If $P(i,j) \neq \emptyset$ for $j = 1,2$ then the tasks corresponding to the sets $P(i,j)$ can be executed in arbitrary order provided that the relations $s_i \in P(i,1)$ and $s_i^* \in P(i,2)$ do not hold. In the case where $s_i \in P(i,1)$ and $s_i^* \in P(i,2)$ then the task corresponding to the vertex s_i^* ought to be executed after the execution of the task corresponding to the vertex s_i. Obviously, first ought to be executed the task corresponding to the set $P(i,j)$ and subsequently the tasks corresponding to the set $P(i+1,j)$.

Finally, the total execution time of the computation which is represented by the network S is given by the formula

$$T(S) = \sum_{i=1}^{r} T^*(i)$$

where T^* is a function defined as follows:

$$T^*(i) = \begin{cases} 1/2\, t^* F(i) & \text{if} \quad \|F(i)\| > 1 \\ t^* F(i) & \text{if} \quad \|F(i)\| = 1 \end{cases}$$

From definition of the function T^* it follows that the total execution time T(S) of the computation can be determined after completing the action of the initial sequencing algorithm. It means that to determine the total execution time T(S) there is no need to use the partitioning algorithm. In other words, the total execution time T(S) can be determined before completing the sequencing of the tasks of the computation in question.

5. Optimality and interpretation

In this section we shall prove some fundamental facts concerning the scheduling algorithm and its optimality. On the ground of these facts we shall interpret basic properties of the algorithm considered.

Theorem 1

Let S^M be the terminal extension of the network S and let k be the range of extension of the network S^*. Let R_i for $1 \leq i < k$ be a graph defined in the following way:

$$(*) \quad R_i = \left\langle \bigcup_{j=i}^{k} F(j) \;,\; \Gamma^* / \bigcup_{j=i}^{k} F(j) \;,\; t^* / \bigcup_{j=i}^{k} F(j) \;,\; x_o^* \right\rangle$$

If there exists such $0 \leq i < k$ that $s_i^* \in F(i+1)$, then the sequence

$$(**) \quad R_p(S) = R_1, \ldots, R_1, \ldots, R_k$$

is only the reduction of the network S^* but is not the direct reduction of the network S^*. In the case where there is no such $0 \leq i < k$ that $s_i^* \in F(i+1)$ then the sequence (**) is the direct reduction of the network S^*.

Proof.

From definition of terminal extension of the network S it follows that $R_1 = S^*$. Since $R_k = \langle F(k) \;,\; \Gamma / F(k) \;,\; t / F(k) \;,\; x_o \rangle$ and thus in order to prove the theorem it is necessary to show that for every $i < k$ if $s_{i-1}^* \in F(i)$ then the graph R_{i+1} is the reduct of the network R_i and if $s_{i-1}^* \in F(i)$ then the graph R_{i+1} is the direct reduct of the network R_i. We shall show now by induction on $i < k$ that the graph R_{i+1} is as the case may be the reduct or the direct reduct of the network R_i.

1. Basic step. Let $i = 1$. By virtue of (*) we obtain that

$$(1) \quad R_2 = \left\langle X^* - F(1) \;,\; \Gamma^* / X^* - F(1) \;,\; t^* / X^* - F(1) \;,\; x_o^* \right\rangle$$

Let a network $S_1 = \langle X_1 \;,\; \Gamma_1 \;,\; t_1 \;,\; x_{01} \rangle$ be the network assigned to the network S as a result of executing first step of the initial sequencing algorithm. If $S_o^* \in F(1)$ then by initial sequencing algorithm the network S_1 is a direct extension of the network S^M and

$$(2) \quad F(1) = \Gamma^*(0) \cup \{s_o^*\} \qquad \text{and} \quad \Gamma_1(s_o) = s_o^*$$

On the other hand, if there exists no such $i = 1$ that $s_o^* \in F(1)$ then by initial sequencing algorithm the network S_1 is the direct extension of the network S^M or identical with the network S and

(3) $F(1) = M^{*}(0)$

Since the network S_1 is the direct extension of the network S^{M} or identical with the network S^{M} or identical with the network S then from (2), (3) and definition of the set $M^{*}(i)$ it follows that

(4) $\Gamma_1(H(1)) \subset X_1 - F(1)$

where H(1) is a set defined in the following way

$$H(i) = \begin{cases} F(i) & \text{if } s^{*}_{i-1} \notin F(i) \\ F(i) - \{s^{*}_{i-1}\} & \text{if } s_1 - 1 \in F(i) \end{cases}$$

Since the network S^{*} is terminal extension of the network S then in virtue of (4) we obtain

(6) $\Gamma^{*}(H(1)) \subset X^{*} - F(1)$

Since $R_1 = S^{*}$ and $\Gamma^{*}(s_o) = s^{*}_o$ then from (6) in virtue of (5) and (1) it follows that if $s^{*}_o \in F(1)$ then the graph R_2 is the direct reduct of the network R_1 and if $s^{*}_o \in F(1)$ then the graph R_2 is the reduct of the network R_1 and is not the direct reduct of the network R_1.

2. Induction step. Let as assume that induction hypothesis holds for $i = n < k$, that is, that the network

(7) $R_n = \langle \bigcup_{j=n}^{k} F(j) , \Gamma^{*}/ \bigcup_{j=n}^{k} F(j) , t^{*}/ \bigcup_{j=n}^{k} F(j) , x^{*}_o \rangle$

is the reduct of the network R_{n-1} if $s^{*}_{n-2} \in F(n-1)$ and is the direct reduct of the network R_{n-1} if $s^{*}_{n-2} \notin F(n-1)$.

To prove the theorem, it is necessary to show that if $s^{*}_{n-1} \in F(n)$ then the graph R_{n+1} is the direct reduct of the network R_n.

Since

(8) $R_{n+1} = \langle \bigcup_{j=n+1}^{k} F(j) , \Gamma^{*}/ \bigcup_{j=n+1}^{k} F(j) , t^{*}/ \bigcup_{j=n+1}^{k} F(j), x^{*}_o \rangle$ and $\Gamma^{*}(s_{n-1}) = s^{*}_{n-1}$

then from (8) by induction hypothesis (7) it follows that the graph R_{n+1} is as the case may be the reduct of the network R_n or the direct reduct of the network R_n if the following relation is satisfied:

(9) $\Gamma^{*}(H(n)) \subset \bigcup_{j=n+1}^{k} F(j)$

where H(n) is defined by (5).

We shall show, now, by leading to a contradiction that the relation (9) holds. Suppose the contrary: the relation (9) does not hold, that is,

(10) $\Gamma^{*}(H(n)) \not\subset \bigcup_{j=n+1}^{k} F(j)$

Since the network S^{*} is the terminal extension of the network S then from (10) in virtue of (5) we have

(11) $\Gamma^{*}(H(n)) \cap \bigcup_{j=1}^{n} F(j) \neq \emptyset$

Since the network S^{*} is the terminal extension of the network S then by definition of initial sequencing algorithm and by definition of the set $M^{x}(i)$ we obtain

(12) $\Gamma^{*}(H(n)) \cap \bigcup_{j=1}^{n} F(j) = \emptyset$

In virtue of (11) and (12) we get a contradiction, and thus the relation (9) holds which means that if $s^*_{n-1} \in F(n)$ then the graph R_{n+1} is the reduct of the network R_n and if $s^*_{n-1} \notin F(n)$ then the graph R_{n+1} is the direct reduct of the network R_n.

In order to prove the theorem it is necessary to show that for the network R_k there exists no reduct. In virtue of (12) we have

$$(13) \quad \Gamma^*(H(k)) \cap \bigcup_{j=1}^{k} F(j) = \emptyset$$

From the fact that S^* is the terminal extension of the network S it follows that $H(k) = F(k)$ and

$$(14) \quad F(k) = X^* - \bigcup_{j=1}^{k-1} F(j)$$

From (13) and (14) it follows that for the network R_k there exists no reduct. Thus the theorem has been proved.

In further considerations, the reduction of the network S^* induced by the sets F(i), that is, the sequence

$$R_f(S) = R_1, .., R_i, ..., R_k$$

will be called f-reduction of the network S.

From theorem 1 it follows that if $x \in F(n)$ and $y \in F(m)$ and $n < m$ then there exists no path leading from the vertex y to the vertex x and moreover, if $x, y \in F(i)$ and $s^*_i \notin F(i)$ then neither $x \in \Gamma^*(y)$ nor $y \in \Gamma^*(x)$.

In connection with, the disjoint partition $\{F(i)\}_{i=1,2,..,k}$ of the set X^* of the network S can be interpreted as follows: If $x, y \in F(i)$ and $s^*_i \notin F(i)$ the tasks corresponding to the vertices x and y can be concurrently executed and conversely, if $s^*_i \in F(i)$ than the tasks corresponding to vertices s_i and s^*_i cannot be concurrently executed. Obviously, first ought to be executed the task corresponding to the vertex s_i and subsequently the task corresponding to the vertex s^*_i. Moreover, if $x \in F(n)$ and $y \in F(m)$ and $n < m$ then the task corresponding to the vertex y ought to be executed after completing the task coresponding to the vertex x.

Let us introduce a partial ordering relation $\prec$ defined on the set X^* in the following way:

$$x \prec y \iff x \in F(n) \wedge y \in F(m) \wedge n < m$$

We shall now show that the sequencing of the tasks corresponding to the set of vertices X^* of the terminal extension of the network S based on the partial ordering relation $\prec$ is optimal, provided that the rules of assigning the processors to the tasks of the computation induced by the sets P(i,j) have not been violated.

Theorem 2.

Let $S' = \langle X', \Gamma', t', x'_0 \rangle$ be such an extension of the network $S = \langle X, \Gamma, t, x_0 \rangle$ that there exists a reduction $R(S') = R'_1, \ldots R'_n$ satisfying the following condition:

$$(*) \quad \begin{array}{l} \text{if } \|X'_i - X'_{i+1}\| = 2 \text{ then } \max_{x \in X'_i - X'_{i+1}} \{t'x\} = 1/2 \sum_{x \in X'_i - X'_{i+1}} t'(x) \\ \text{if } \|X'_i - X'_{i+1}\| > 2 \text{ then } \max_{x \in X'_i - X'_{i+1}} \{t'x\} = 1/2 \sum_{x \in X'_i - X'_{i+1}} t'(x) \end{array}$$

Let T′ be a function defined for the network S′as follows:

$$T(i) = \begin{cases} 1/2 \sum_{x \in X'_{i+1}} t'(x) & \text{if } \|X'_i - X'_{i+1}\| > 1 \\ t'(x) & \text{if } x = x'_o \text{ or } \{x\} = X'_i - X'_{i+1} \end{cases}$$

There exists no such extension S′ of the network S that

$$\sum_{j=1}^{n} T'(j) < T(S)$$

Proof.

We prove the theorem by leading to a contradiction. Suppose the contrary: there exists such an extension S′ of the network S satisfying the condition (*) and there exists such a reduction

(1) $R(S') = R'_1, \ldots, R'_i, \ldots, R'_n$

that the following relation

(2) $$\sum_{i=1}^{n} T'(i) < T(S)$$

holds. Let S^* be terminal extension of the network S and let the sequence

(3) $R_f(S) = R_1, \ldots R_k$

be f-reduction of the network S^*.

If the relation (2) holds then there exists at least such a reduct R_{i+1} that $\|F(i)\| > 1$. Let us assume that the reduct R_{i+1} is the first (in the sense of the numeration of the reducts in f-reduction (3)) from the reducts sytisfying the following condition

(4) $\|F(i)\| > 1$

Let $F(i) = \{x_i\}$. If the condition (4) holds then from initial sequencing algorithm it follows that

(5) $M(i-1, n(i-1)) = \{x_i\}$ and $D(i-1) = \emptyset$

which means if $x \neq x_i$ then

(6) $$\Gamma^{*-1}(x) \not\subset \bigcup_{j=1}^{i-1} F(j)$$

The question arises whether for the network R_i can be created such a direct reduct R_{i+1} for which the following relation

(7) $$\|\bigcup_{j=1}^{k} F(j) - X^*_{i+1}\| > 1$$

holds. We shall now show that the answer for the question given above is negative.

If there exists direct reduct R^*_{i+1} of the network R_i satisfying the relation (7) then from definition of direct reduct it follows that there exists a vertex $x \neq x_i$ such that $x \in \bigcup_{j=1}^{k} F(j)$ and $\Gamma^{*-1}(x) \subset \bigcup_{j=1}^{i-1} F(j)$ which leads to a contradiction with (7). And thus there exists no such direct reduct R^*_{i+1} of the network R_i satisfying the relation(7)

Let us assume that S' is such an extension of the network S that there exists such reduction (1) and the reduct R'_j satisfying the following relation

$$(8) \qquad X'_j = \bigcup_{j=i}^{k} F(j)$$

From (8) and definitions of the function T and T' it follows that

$$(9) \qquad \sum_{n=1}^{j-1} T'(n) \geq \sum_{n=1}^{i-1} T(n)$$

Let us assume that the reduct R_{i+1} is the only one reduct of the network S satisfying the relation (4). Since for the network R_1 there exist no direct reduct R^*_{i+1} satisfying the relation (7) , and thus from 8 it follows that for the reduct R'_{j+1} of the network S'the following relation $X'_j - X'_{j+1} = F(i)$ holds, which by definitions of the function T' and T yields

$$(10) \qquad \sum_{m=j}^{n} T'(m) \geq \sum_{m=i}^{k} T(m)$$

From (9) and (10) we obtain

$$(11) \qquad \sum_{i=1}^{n} T'(i) \geq T(S)$$

In virtue of (2) (11) we get a contradiction, and thus the theorem has been proved.

From this theorem it follows that the value of the function T(S) determines the smallest possible total execution time of the computation (schedule length) which is represented by the network S. On the other hand, from dedinition of the function T it follows that the total execution time of the computation can be determined after completing the acting of the initial sequencing algorithm. To execute the computation is needed to complete acting of the initial sequencing algorithm and the partitioning algorithm. These facts mean that the total execution time T (S) of the computation can be determined before completing the sequencing of the tasks of the computation in question.

6. Concluding remarks

In spite of simplicity of formulation many of scheduling problems are very hard problems. Most of them are known to be NP-complete [10] . This means that the number of steps of the scheduling algorithm grows exponentialy with the size of the problem and thus in some cases these problems are intractable.

Let us consider a partition problem. The partition problem can be stated as follows:

Let $(C_1, C_2, \ldots, C_n) \in Z^n$ be arbitrary n-tuple of positive real numbers. Is there a set of indices $J \subset \{1,2,..,n\}$ such that

$$\sum_{s \in J} C_s = \sum_{s \notin J} C_s$$

This simple formulated problem is very good known to be NP-complete [10] . It is easy to see that we can find the solution of this problem by finite enumeration of all sets of indices J but the solutions is exponential in the size of the problem.

Referenoes

[1] Rowicki, A. Nonpreemptive Soheduling for Two-Prooessor Systems Praoe naukowo-badawoze IMM nr 1, rok XXVI (1984) , pp 41-58.

[2] Muntz, R.R. and Coffman, E.G., Jr. Optimal Preemptive Soheduling on Two-Proocessor Systems. IEE Transactions on Computers, vol. C-18, no. 11 (1969) , 1014-1020.

[3] Muntz, R.R. and Coffman E.G., Jr. Preemptive Scheduling of Real-Time Tasks on Multiproocessor Systems. J. of ACM, vol. 17, no. 2, (1970) , 324-338.

[4] Horvath, E.C., Lam, S. and Sethi, R. A Level Algorithm for Preemptive Soheduling J. of ACM, vol. 24, no. 1 (1977), 32-43.

[5] Coffman, E.G., Jr and Graham, R.L. Optimal Soheduling for Two-Proocessor Systems. Acta Informatioa, vol. 1, no. 3 (1972) , 200-213

[6] Rowioki, A. On Optimal Preemptive Scheduling for Multiprooessor Systems. Bull. Aoad. Poln. Soi., Ser., Soi, Math. Astronom. Phys. vol. 26, no. 7 (1978) , 651-660.

[7] Rowioki, A. A Note on Optimal Preemptive Soheduling for Two-Proocessor Systems. Information Proocessing Letters, vol. 6, no. 1 (1977), 25-28.

[8] Rowioki, A. On Optimal Preemptive Scheduling for Two-Proocessor Systems. Bull. Aoad. Polon. Soi., Ser., Soi. Math. Astronom. Phys., vol. 25, no 7 (1977), 697-706.

[9] Rowioki, A. Planowanie obliczeń zależnyoh podzielonyoh dla systemów dwuprooesorowyoh. Praoe Naukowo-Badawoze IMM, Warszawa 1980, 100-117.

[10] Karp, R., Reduoibility Amonh Combinatorial Problems. Complexity of Computer Computations, R. Miller, J. Thatcher, eds., Plenum Press, New York, pp. 85-103.

Prace naukowo-badawcze Instytutu Maszyn Matematycznych

Metoda minimalizacji funkcji logicznych

Marian Ligmanowski
Instytut Łączności Gdańsk

Opracowana metoda minimalizacji funkcji logicznych obejmuje dwa etapy. W pierwszym etapie znajduje się minimalne pokrycie funkcji, natomiast w drugim - proste implikanty zapewniające to pokrycie.

W znanej metodzie Quine'a Mc-Cluskey'a minimalizacji funkcji logicznych znajdują się najpierw wszystkie proste implikanty danej funkcji, a następnie minimalny podzbiór prostych implikantów, zapewniający pokrycie funkcji (np. [1, 2]). Może być interesujące pytanie czy możliwy jest porządek odwrotny: najpierw znalezienie minimalnego pokrycia funkcji za pomocą implikantów wspólnych dla pewnej liczby konstytuentów, a dopiero w drugim etapie wyznaczenie prostych implikantów (już nie wszystkich), przez które mogą być zastąpione implikanty wyznaczone w pierwszym etapie. Okazuje się, że oparta na tej koncepcji metoda może być użyteczna dla funkcji w pełni określonych, a szczególnie dla niecałkowicie określonych.

Niech dwuwartościowa funcja logiczna $f(x_1, \ldots, x_n)$, której argumentami są zmienne dwuwartościowe $x_1, \ldots, x_n$, jest zadana za pomocą dwóch tablic T1 i T2. Tablica T1 zawiera zbiór ciągów wartości argumentów $x_1, \ldots, x_n$ - liczb binarnych odpowiadających konstytuentom, dla którego funkcja przyjmuje wartość 1, tablica T2 - analogiczny zbiór, dla którego funkcja ma wartość 0. Niech K_1, K_2 oznaczają odpowiednio liczbę wierszy (konstytuentów) tablicy T1 i T2.

Konstytuenty zapisane w wierszach i_1, i_2 tablicy T1 mają wspólny implikant p, jeżeli zawiera się on w obu wierszach i_1, i_2 tej tablicy i nie występuje w żadnym wierszu tablicy T2.

Przykład - n=6. Niech konstytuenty występujące w tablicy T1 mają postać: wiersz i_1 - 101011 (oznacza to zbiór wartości argumentów $x_1=x_3=x_5=x_6=1$, $x_2=x_4=0$), wiersz i_2 - 001110. Koniunkcja -01---$(\bar{x}_2 x_3)$ jest implikantem funkcji f jeżeli nie zawiera się w żadnym wierszu tablicy T2. Koniunkcja ta pokrywa wymienione konstytuenty.

Chcąc wyznaczyć minimalną postać funkcji f można zastosować następujący tok postępowania.

1) Wybór wiersza i_1 tablicy T1.

Znalezienie wszystkich możliwych sposobów pokrycia konstytuentów z udziałem wiersza i_1.

2) Znalezienie wszystkich implikantów zawierających się w wierszu i_1, wspólnych dla pewnej liczby wierszy tablicy T1 /stanowiących pokrycie odpowiadających im konstytuentów/.

3) Powtórzenie p.1-2 dla tych wierszy tablicy T1, które nie weszły do pokrycia otrzymanego w p. 2.

4) Spośród różnych wariantów pokrycia wyznaczonych w p. 2, przeprowadzenia wyboru minimalnego pokrycia wszystkich konstytuentów tablicy T1, to jest zawierającego najmniejszą liczbę implikantów.

5) Znalezienie prostych implikantów dla pokrycia minimalnego wybranego w p. 4 i ocena całkowitej liczby liter pokrycia.

6) Powtórzenie p.4-5 - wyznaczenie ewentualnego innego pokrycia minimalnego i sprawdzenie, że pokrycie o mniejszej liczbie liter nie istnieje.

Dla znalezienia implikantów zawierających się w wierszu i_1 (p. 2) należy rozpatrzyć pary wierszy i_1, i_2 /$i_2 \neq i_1$/ tablicy T1. Pokrycie wierszy i_1, i_2 stanowi część wspólna liczb występująca w obu wierszach. Dla konstytuentów 101011 i 001110 częścią wspólną jest koniunkcja k= -01-1-. Wspólny implikant dla wierszy i_1, i_2 istnieje, jeżeli wyznaczona część wspólna nie zawiera w żadnym wierszu tablicy T2.

Po wyznaczeniu implikantów dla wszystkich par wierszy i_1, i_2 (i_1 ustalone zgodnie z p.1, $i_2=1, \ldots, K_1$, $i_2 \neq i_1$), znajduje się implikanty pokrywające trzy i więcej konstytuentów jako część wspólną par implikantów. Jeżeli otrzymuje się przy tym koniunkcję o mniejszej liczbie liter (która jest równa łącznej liczbie zer i jedynek części wspólnej), należy sprawdzić czy nie występuje ona w tablicy T2 i tylko wówczas jest implikantem funkcji f. Proces tworzenia nowych implikantów zostaje zakończony jeżeli z otrzymanego zbioru implikantów nie można utworzyć żadnego nowego implikantu. W przypadku gdy dla wszystkich rozpatrzonych par konstytuentów i_1, i_2 nie został znaleziony żaden implikant, jako implikant należy przyjąć konstytuent zapisany w wierszu i_1 tablicy T1.

P r z y k ł a d.

Funkcja $f(x_1, \ldots, x_6)$ jest przedstawiona za pomocą tablic T1, T2.

T1

nr wiersza	konstytuent
1	001 110
2	100 001
3	010 100
4	001 101
5	110 110
6	101 011
7	011 100

T2

nr wiersza	konstytuent
1	110 001
2	101 110
3	001 000
4	110 101
5	011 011
6	100 111

T3

pokrycie	implikant
1,3	0— 1-0
1,4	001 1—
1,7	0-1 1-0

T4

pokrycie	implikant
1,3,4,7	0— 1—

Niech $i_1=1$. W tablicy T3 zestawiono implikanty otrzymane dla par wierszy $i_1 = 1$ $i_2=2, \ldots,7$, tablicy T1. Dla par nie wpisanych do T3 implikant nie istnieje. Z kolei tworząc możliwe pary implikantów z tablicy T3 otrzymuje się jeden implikant o liczbie liter m=2, pokrywający cztery konstytuenty (tablica T4).

Rozpatrując wiersz $i_1=2$ (nie występuje w znalezionym pokryciu - tabl. T4) otrzymuje się implikanty zestawione w tablicy T5. Można przy tym pominąć parę 2, 1.

T5

pokrycie	implikant
2,4	-0- -01
2,6	10- 0-1
2,4,6	x

T6

pokrycie	implikant
5,3	-10 1-0
5,7	-1- 1-0
5,3,7	-1- 1-0

W tablicach T4, T5 nie występuje konstytuent oznaczony nr 5. Pomijając pary 5,1 i 5,2 (już rozpatrzone) otrzymuje się ostatecznie jeden implikant pokrywający ten konstytuent (ostatni wiersz tablicy T6).

Oznaczając znalezione implikanty przez p_1=0—1—, p_2=-1-1-0, p_3=10-0-1 można zauważyć, że tworzą one minimalny zbiór pokrywający wszystkie konstytuenty 1-7 z tablicy T1; nie istnieje zbiór o mniejszej liczbie implikantów. Z uwagi na to, że w tablicy T2 występują cyfry 1 i 0

w każdej kolumnie, nie mogą istnieć implikanty jednoliterowe. Implikant p_1 jest dwuliterowy, jest więc zarazem prostym implikantem. Implikanty p_2 i p_3 zawierają większą liczbę liter niż dwie, mogą zatem istnieć pokrywające je implikanty o mniejszej liczbie liter.

Wyznaczenie prostych implikantów dla minimalnego pokrycia funkcji p.5 jest możliwe oddzielenie dla każdego otrzymanego implikantu. Wykonuje się następujące operacje:

1) Znalezienie największej części wspólnej w implikantu p i konstytuentów zapisanych w tablicy T2.

2) Dla znalezionej części wspólnej w i implikantu p tworzy się koniunkcje m-literowe, które nie są zawarte w części wspólnej w, ale zawierają się w implikancie p, przy czym $m_{min}=2$ (jeżeli można wykluczyć implikanty jednoliterowe), $m_{max}=m_p-1$ (m_p - liczba liter implikantu p).

3) Utworzona koniunkcja m-literowa nie występująca w tablicy T2 jest prostym implikantem, jeżeli nie istnieje koniunkcja (m-1)- literowa, która nie jest zawarta w T2.

W rozpatrywanym przykładzie dla p_2 istnieje część wspólna w_2=——1-0 (wiersz 2 T2). Możliwe są dwie koniunkcje dwuliterowe k_1=-1-1— i k_2=-1——0, pokrywające implikant p_2 i nie występujące w w_2. Tylko koniunkcja k_2 nie występuje w żadnym wierszu T2, a zatem implikant p_2 może być zastąpiony przez prosty implikant p_2' =-1——0.

Dla pierwszego wiersza T2 i implikantu p_3 część wspólna w_3=1—0-1. Jeżeli istnieje implikant dwuliterowy pokrywający p_3, to musi on zawierać jedną cyfrę występującą w w_3 oraz 0 na drugim miejscu jak w p_3. Otrzymane w ten sposób koniunkcje nie są jednak implikantami, bo są zawarte w T2. Istnieją trzy koniunkcje trzyliterowe: 10-0—, 10——1, -0-0-1, z których tylko druga nie jest implikantem. Zatem p_3 można zastąpić przez proste implikanty:
p_3' =10-0— lub p_3'' =-0-0-1.

Minimalna postać funkcji zawiera więc dwie koniunkcje dwuliterowe i jedną koniunkcję trzyliterową, razem 2.2+3=7 liter. Gdyby każdy z konstytuentów 2, 6 mógł być zastąpiony innym implikantem dwuliterowym, to zamiast p_3 o trzech literach trzeba by stosować dwie koniunkcje dwuliterowe o łącznej liczbie czterech liter. Świadczy to o tym, że otrzymane postacie funkcji są minimalne:

$$f' = p_1 \vee p_2' \vee p_3' = \bar{x}_1 x_4 \vee x_2 \bar{x}_6 \vee x_1 \bar{x}_2 \bar{x}_4$$

lub $$f'' = p_1 \vee p_2' \vee p_3'' = \bar{x}_1 x_4 \vee x_2 \bar{x}_6 \vee \bar{x}_2 \bar{x}_4 x_6$$

Chociaż nie istnieją postacie funkcji o mniejszej liczbie liter niż 7, to jednak mogą istnieć postacie równoważne o tej samej liczbie liter. Wybierając implikant zapisany w pierwszym tablicy T6, p_2'' = -101-0 (konstytuent nr 7 jest pokryty przez p_1), można uzyskać prosty implikant p_2''' = —0—0, a zatem innymi postaciami minimalnymi są:

$$f''' = p_1 \vee p_2''' \vee p_3'$$

lub $$f^{IV} = p_1 \vee p_2''' \vee p_3''$$

Pewnego omówienia wymaga jeszcze p.6 metody.

Należy stwierdzić, że metoda bezpośrednio zapewnia otrzymanie postaci minimalnej funkcji, która zawiera minimum liter przy minimalnej liczbie wyrażeń / koniunkcji - dla postaci alternatywnej normalnej/. W zależności od przyjętego kryterium minimalizacji funkcji, np. minimalna liczba z góry określonych elementów, które mogą być użyte do realizacji funkcji - metoda wymaga pewnych modyfikacji.

Gdy kryterium minimalizacji stanowi całkowita liczba liter funkcji, ciąg operacji według p.1 - 5 nie zapewnia w każdym wypadku otrzymania postaci minimalnej. Należy wówczas sprawdzić (p.6) czy można otrzymać proste implikanty o mniejszej liczbie liter niż wyznaczone w p. 5 i jeżeli to zachodzi, czy może istnieć inne pokrycie funkcji, dla którego całkowita liczba liter ulegnie dalszemu zmniejszeniu. Podstawę wyboru stanowią zawsze implikanty otrzymane w p. 2.

Punkt 6 staje się zbędny, jeżeli jako kryterium minimalizacji przyjąć minimum liter przy minimalnej liczbie wyrażeń funkcji. W takim razie w p. 5 należy uwzględnić wszystkie pokrycia o tej samej minimalnej liczbie implikantów (jeżeli istnieje więcej niż jedno pokrycie). Punkt 6 może być również pominięty, gdy kryterium minimalizacji jest określone jako minimum całkowitej liczby liter, ale postać minimalna może być określona z pewnym przybliżeniem do tego minimum.

Minimalną postać koniunkcyjną funkcji można otrzymać w ten sam sposób zmieniając jedynie między sobą role tablic T1 i T2 oraz stosując operację negacji do otrzymanej postaci minimalnej alternatywnej.

Metodę można łatwo rozszerzyć na minimalizację zbioru funkcji logicznych, uzyskując minimum liter dla określonego zbioru funkcji, a nie tylko dla każdej funkcji rozpatrywanej oddzielnie.

Z uwagi na to, że większość operacji dotyczy działań na liczbach binarnych, metoda może być dostosowana do obliczeń maszynowych, co zwiększy jej efektywność. W porównaniu do metody Quine'a Mc-Cluskey'a wymaga często mniej operacji, gdyż nie uwzględnia konstytuentów niewykorzystanych, dla których funkcja nie jest określona i których liczba jest zwykle znaczna.

Wykaz literatury:

1. Głuszkow W.M.: Synteza automatów cyfrowych. Warszawa 1968
2. Siwiński J.: Układy przełączające w automatyce. Warszawa 1980

The method of logical function minimization

Summary

The elaborated method of logical function minimization contains two parts. In first part a minimal function covering is determined, while in second - prime implicants insuring this covering.

U W A G A !

Instytut Maszyn Matematycznych przekaże nieodpłatnie do eksploatacji sprzętowo niezależny system programowania grafiki komputerów PSG zaimplementowany na minikomputerze MERA 400.

Bliższe informacje: Pracownia Grafiki Komputerowej, tel. 21-84-41 w. 271, 388, 428.

Informacja o cenach i warunkach prenumeraty na 1987 r.
– dla czasopism Instytutu Maszyn Matematycznych

● Cena prenumeraty rocznej

Techniki Komputerowe – Biuletyn Informacyjny	2280.– dwum.
Przegląd Dokumentacyjny – Nauki i Techniki Komputerowe	1860.– dwum.
Informacja Ekspresowa – Nauki i Techniki Komputerowe	4200.– mies.
Prace naukowo-badawcze Instytutu Maszyn Matematycznych	1800.– 3x w roku

● Warunki prenumeraty

1/ dla osób prawnych – instytucji i zakładów pracy:
- instytucje i zakłady pracy zlokalizowane w miastach wojewódzkich i pozostałych miastach, w których znajdują się siedziby oddziałów RSW "Prasa-Książka-Ruch" zamawiają prenumeratę w tych oddziałach;
- instytucje i zakłady pracy zlokalizowane w miejscowościach, gdzie nie ma oddziałów RSW "Prasa-Książka-Ruch" i na terenach wiejskich opłacają prenumeratę w urzędach pocztowych i u doręczycieli;

2/ dla osób fizycznych – prenumeratorów indywidualnych:
- osoby fizyczne zamieszkałe na wsi i w miejscowościach, gdzie nie ma oddziałów RSW "Prasa-Książka-Ruch" opłacają prenumeratę w urzędach pocztowych i u doręczycieli;
- osoby fizyczne zamieszkałe w miastach – siedzibach oddziałów RSW "Prasa-Książka-Ruch" opłacają prenumeratę wyłącznie w urzędach pocztowych nadawczo-oddawczych właściwych dla miejsca zamieszkania prenumeratora. Wpłaty dokonują używając "blankietu wpłaty" na rachunek bankowy miejscowego oddziału RSW "Prasa-Książka-Ruch";

3/ Prenumeratę ze zleceniem wysyłki za granicę przyjmuje RSW "Prasa-Książka-Ruch", Centrala Kolportażu Prasy i Wydawnictw, ul. Towarowa 28, 00-958 Warszawa, konto NBP XV Oddział w Warszawie nr 1153-201045-139-11. Prenumerata ze zleceniem wysyłki za granicę pocztą zwykłą jest droższa od prenumeraty krajowej o 50% dla zleceniodawców indywidualnych i o 100% dla zlecających instytucji i zakładów pracy.

● Terminy przyjmowania prenumeraty na kraj i za granicę:

- do dnia 10 listopada na I kwartał, I półrocze roku następnego oraz na cały rok następny,
- do dnia 1 – każdego miesiąca poprzedzającego okres prenumeraty roku bieżącego.

Zamówienia na prenumeratę "Prac naukowo-badawczych Instytutu Maszyn Matematycznych" przyjmuje Dział Sprzedaży Wysyłkowej Ośrodka Rozpowszechniania Wydawnictw Naukowych PAN, Warszawa, Pałac Kultury i Nauki, tel. tel. 20-02-11 w. 2516. Egzemplarze pojedyncze Prac są do nabycia w księgarni ORWN PAN, Warszawa, Pałac Kultury i Nauki, tel. 20-02-11 w.2105.